6.006
Study

5/21

Start of review of # theory

Use 6.042 notes as well

- study of integers
- a divides b if integer k so $ak = b$
- gcd(a, b)
  - greatest common divisor of a and b

  gcd(18, 24) = 6

  everything has one
- gcd of a and b is the linear combo of a and b

  $gcd(a,b) = sa + tb$

  (I seem like I understood 6.042 much better now I always like that 2nd time around)

(2)

Prime - divisible by itself and 1 only

Every ⊛ # has a unique prime factorization
esp if sort by size (since can easily reorder)
$12 = 2 \cdot 2 \cdot 3$

Modular arithmetic

$a \equiv b \pmod{n}$

ie $29 \equiv 15 \pmod{7}$

since $7 \mid (29-15)$

*preserved by addition and multiplication

So can do math by remainder
[add, multi, sub

1. replace every int by remainder mod n
2. immed: replace any result outside of range w/ remainder on div by n

(3)

$+_n$ = notation for add then imm. dividing by n and taking the remainder

$\times_n$ = same for mult.

So the set of int for which this applies = $\mathbb{Z}_n$
Ohhhh!!!
└ "the ring of integers mod n"

---

relative prime - integers that have no prime factors in comon

Same as $\gcd(a, b) = 1$

Canceling factors does not work
└ normally $rs = st$

(4)

Ok back to 6.006

Z_n = the additive group of integers modulo n

└ defn matches

Two integers rel prime $\gcd(a,b) = 1$

Z_n^* = int $< n$ and <u>rel prime</u> to n

every els $a \in Z_n^*$ has an inverse that is easy to compute

<u>Multiplicative inverse</u>

$x \cdot x^{-1} = 1$

$\frac{1}{3} \cdot \frac{1}{3} = 1$

Every # $\frac{n}{m}$ has an inverse $\rightarrow \frac{m}{n}$

~~Ot~~ only $1, -1$ has inverse in Z_m

In Z_{15} $2 \cdot 8 = 1$

↑ are mult inverses

(5)

But 3 does not

Since $\exists$ is $j = 1$

"no j exists"

is an inverse if the #s are rel prime (aka coprime)

"How to find inverse"

Extended Euclidean algo - finds the GCD

"the table thing"

eg gcd(120, 23)

Dividend	Divisor	Quotient	Remainder	
120	23	5	5	
23	5	4	3	
5	3	1	2	
3	2	1	1	GCD = 1
2	1	2	0	

$a^{-1} \equiv x \pmod m$

"inverse of a is x"

$ax \equiv a a^{-1} \equiv 1 \pmod m$

When found $\rightarrow$ it is unique

But how to actually find

$ax + by = \gcd(a, b)$

↑ a, b are given

Sol to

$ax \equiv 1 \pmod{m}$

So

$ax - 1 = qm$

$ax - qm = 1$

x = inverse

q = int that discarded

$\gcd(a, m) = 1$ is pre det

So what do ya run?

Runs in $O(\log(m)^2)$

Also: Eulers Theorm

If a rel prime to m

$a^{\varphi(m)} \equiv 1 \pmod{m}$

(7)

Found some Pseudo code

```
Mod inv (a, m);
    g, x, y = egcd (a, m)
    if g != 1;
        return None (inv doesn't exist)
    else
        return x % m
```

So calc the Ecter gcd (a, m)

then check if the gcd is 1

if it is return x % m

$ax + by = g = \gcd(a, b)$

So test. Want inverse 8 mod 15 Should be 2

egcd (8, 15)

8 15 0 8

(my method doesn't track x, y)

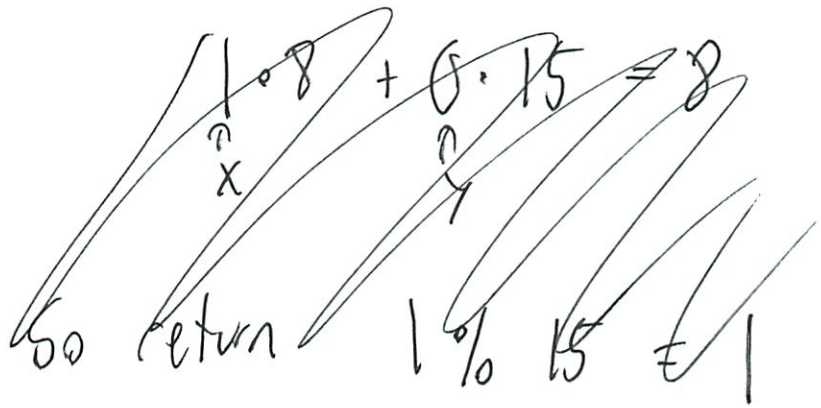
guess $-8 + -15 = 8$

Or can you do more

(8)

15 8 1 7

8 7 1 ① → gcd = 1



So return 1 % 15 ≡ 1

$$.8 + .15 = 1$$

Need to do the table

↓ rem x quotient

$7 = 15 - 1 \cdot 8$ ← y

$1 = 8 - 1 \cdot 7$

$= 8 - 1(15 - 1 \cdot 8)$ ← plug in for

Combine

$2 \cdot 8 - 1 \cdot 15$

So $1 = 2 \cdot 8 + (-1) \cdot 15$

↑x ↑y return x % m = 2 (mod 15) = 2

9

So after that long digression, I think I can find math. inverses,

Try it again — inverse of 8 mod 15

a	b	quo	rem
8	15	0	8
15	8	1	7 = 15 − 1·8
8	7	1	1 = 8 − 1·7

$1_{gcd} = 8 - 1 \cdot 7$

$= 8 - 1(15 - 1 \cdot 8)$

$= 2 \cdot 8 - 1 \cdot 15$

$x \quad y$

return $2 \% 15 = 2$ ✓

So where were we?

Z_n^* = multip. group of int $< n$ and rel prim to n

inverse easy (since gcd = 1)

Z_p^* is a special case when n = prime

(10)

Z_p^* is a cyclic group of order $p-1$

Cyclic group - a group that can be generated from a single element (g)
So that ever el is a power of g

---

ie $G = \{g^0, g^1, g^2, g^3, g^4, g^5\}$ Set defined as
Then $g^6 = g^0$ and its cyclic
Tau to when set defined as i
This is isomophic to $\{0, 1, 2, 3, 4, 5\} \bmod 6$

For every $\oplus$ int n, there is one group whose order is n

---

Order (group theory)

= Cardinality = # of items inside set
Sometimes also the period

So Z_p^* is $\{g, g^2, \ldots, g^{p-1}\}$

Example 2 is a generator of Z_{11}^*

2^1	2
2^2	4
2^3	8
2^4	5
2^5	10
2^6	9
2^7	7
2^8	3
2^9	6
2^{10}	1

mod 11

Notice: unique!

; How to find generators

↳ knew we talked about in class

(2)

Fermat's Little Theorm

$a^{p-1} \equiv 1 \mod p$ for all $a \in Z_p^*$

So for 2 which is in Z_{11}^*

$2^{11-1} \stackrel{?}{\equiv} 1 \mod p$

$2^{10} = 1$ ✓

How about $2^7 \equiv 7$

$7^{11-1} \pmod{11} = 1$ ✓

(I remember this from 6.042 – never understood...)

$\phi(n)$ is Euler tolient function

$\phi(n) = |Z_n^*|$

= The # of ints that are rel prime to n

How to find again?

(13)

Plus $k^{\phi(n)} \equiv 1 \pmod{n}$

$\phi(mn) = \phi(m)\phi(n)$

gives the order of $\mathbb{Z}_n^*$

$\varphi(n) = n \prod_{p|n} \left(1 - \frac{1}{p}\right)$

think thats all we need to know

When p is prime all int are rel prime to p

└ kinda by def'n

$\phi(p) = p - 1$

If p^k is a prime power then all int b/w 1 and p^k are rel prime to p^k except $p, 2p, 3p, \ldots p^{k-1}p$

$\phi(p^k) = p^k - p^{k-1}$

(14)

Mod exponentiation

↓ Bt actual problem

Given a, x, n Compute $a^x \bmod n$

Niave

1. set $y = 1$
2. Repeat x times $y = y \cdot a$
3. Return $y \bmod n$

But ~~[illegible]~~ that propety cumulative?

1. set $y = 1$
2. Repeat x times $y = (y \cdot a) \bmod n$
3. Return y

But still has x multiplications

So Repeated Squaring

Uses $(a^k)^2 = a^{2k}$

So $\left(a^{(2i)}\right)^2 = a^{2 \cdot 2i} = a^{2(i+1)}$

So we can obtain a^{2^i} in only i multiplications

? How does this work exactly ?

Example ? Think saw one in lecture

Was this Fast exponentiation in 6.042 ?

Algorithmist example

Say want 3^{13}

Know

3^1	3
3^2	$3 \cdot 3 = 9$
3^4	$9 \cdot 9 = 81$
3^8	$81 \cdot 81 = 6561$

So $3^{13} = 3^8 \times 3^4 \times 3^1$

$[8 + 4 + 1 = 13]$

$= 6561 \cdot 81 \cdot 3$

$= 1594323$

But under exp by squaring on WP - lots of examples

So this works in $\log n$ multiplications

BA only if it's a power of 2!

If x is not a power of 2, can write as

$$x = \sum_{i=1}^{k} b_i 2^i$$

where $b_k b_{k-1} \dots b_0$ is the binary representation of x

there $k = \lfloor \log_2(x) \rfloor$

$$a^x = a^{\sum_{i=1}^{k} b_i 2^i}$$

$$= \prod_{i=1}^{k} a^{b_i 2^i}$$

$$= \prod_{i=1}^{k} \left(a^{(2^i)}\right)^{b_i}$$

(why are doing this: binary)

(17)

So this is prod of $a^{(2^i)}$ for i where $b_i = 1$

So $O(k) = O(\log x)$ multiply

1. Make a matrix $A[i] = a^{2^i}$
   - i Set $A[0] = a$
   - ii For $i = 1, \ldots, k$ $A[i] = (A[i-1])^2 \bmod n$
2. Obtain the binary repesentation
3. $y = 1$
4. For $i = 1, \ldots k$: If $b_i = 1$, set $y = (A[i] \cdot y) \bmod n$
5. Return y

So what is this binary rep?

Is just the # in binary

Im guessing b_k is # of digits

└ yeah since that is $\lfloor \log_2(x) \rfloor$

(18)

Reloaded over lecture notes

The binary says where to square

$b = 101 \ldots$

$= 2^k + 2^{k-2} + \ldots$

Quadratic residues modulo p

We say that a is a quadratic residue (square) mod p if exists $x \in \mathbb{Z}_p^*$ such that $a \equiv x^2 \bmod p$

Note that $\mathbb{Z}_p^*$ has generator g

$x \in \mathbb{Z}_p^*$ as $x = g^k$

Set $\{g^1, \ldots, g^{p-1}\}$ is $\{1, \ldots, p-1\}$

the set of els of $\mathbb{Z}^*_p$

Deciding if a # is a quadratic reside mod p

Have a generator g

What els are quadratic residues?

Quadratic reside

if q is congruent to perfect square mod n

$$x^2 \equiv q \pmod{n}$$

if there exists an x s.t.

For a given n

then a list of q.r. may be obtained by squaring $0, 1, \ldots, n-1$

So for 5 qr are 1, 4

6 1, 4, 5, 9, 16

18 1, 4, 7, 9, 10, 13, 16

Since $9^2 \pmod{18} = 9$

think this is a coinc.

We know the set is $\{g^1, \ldots, g^{p-1}\}$

Thus the set of squares is $g^2, g^4, \ldots, g^{2p-2}$

But are counting els ttwice

So $g^2, g^4, \ldots g^{p-3}, g^{p-1} = g^0$

So are even powers of g

(where is this leading?)

Eulers' criterion The el a is square mod p iff $a^{\frac{p-1}{2}} \equiv 1 \bmod p$

Moreove a is not a square mod p iff $a^{\frac{p-1}{2}} \equiv -1 \bmod p$

(21)

Let me go through lecture notes again

Is "a" a square mod p^r?

binary search

all generators distinct

Any el to power of group size $(p-1) = 1$

---

ex

is g^{2i} square

g^i is in square

So $(g^i)^2$ is as well

$$\left(g^{\frac{p-1}{2}}\right)^2 = g^{p-1} = 1$$

(oh thats where that comes from!

(22)

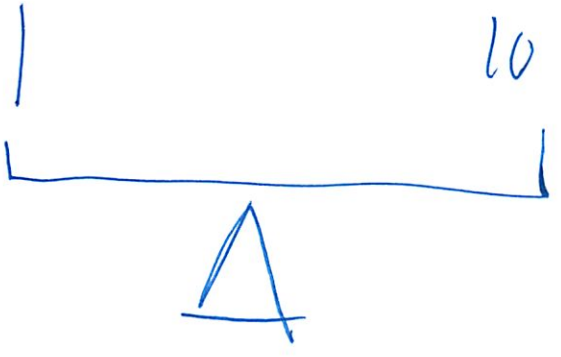
g^{2i}

g^i $\quad$ $\left(g^{i+\frac{p-1}{2}}\right)^2$

Any quad has exactly 2 roots

g^{2i+1} $\quad$ $g^{\text{odd \#}}$ could be ?

? is square



← if sq has 2 roots

1 $\quad$ 10 $\quad$ p-1

Can't be root of 2 diff groups

This is totally confusing!

(23)

Recitation

Given a prime, find a generator

$\exists g$ s.t. $\{g, g^2 \dots g^{p-1}\} = Z_p^*$

ie a Z_7^* is generated by 3

$3 = 3$
$3^2 = 2$
$3^3 = 6$
$3^4 = 4$
$3^5 = 5$
$3^6 = 1$

kinda proof

$g^{p-1} = 1$ for any g

Then a bunch of iterations of code

- simplest try all $2 \to p$
  $O(n^2)$ time

This has nothing to do w/ rest of 6.006

Generators mod p are randomly evenly distributed

└ One every log n #s

generators $\approx \frac{p}{\log p}$

$= \Omega\left(\frac{p}{\log p}\right)$

└ I need to remember symbol

Symbols

Θ tight bound

c_i is a Θ constant

$f(n) = \Theta(g(n))$

O upper bound

Ω lower bound

o upper bound - not asy tight

ω lower bound - not asy tight

Lecture

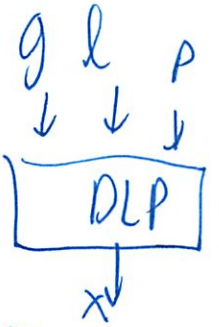
Non square mod p iff $a \equiv g^{2i+1}$

How to decide if square or not?

1. Find generator
2. Find discrete log that gives exponent
3. Even or odd?

How to effiently find a generator?

DLP gives $g, p, e \in \mathbb{Z}_p^*$



But very hard to actually solve

↳ that means we don't need to know

26

(Still really confused...!)

In recitation again - pick a random # - test if generator

---

p = prime

g = gen mod p

Given a we can compute g^a mod p

But given g^a we can't compute a

↳ I get this → st. Crypto stuff

We can't easily find a g for a given p

But can compute a g and p at same time randomly

↳ Don't need to know

Crypto uses this

(27)

A tells B p, g

A picks a, announces g^a

B " b " g^b

$g^{ab} = (g^b)^a = (g^a)^b$

Diffe-Hellman

What qu will they ask here?

Lecture: NP completeness

tractable = NP complete
= solvable in poly. time

intractable can't solve in polynomial

28

Next lecture

This was this the "weird" lecture
Where he had a fn called weird

Testing if something is composite or not

Think going to start from top

(skipping basic stuff) —

Vector Space model

Peak Finding

Core concepts

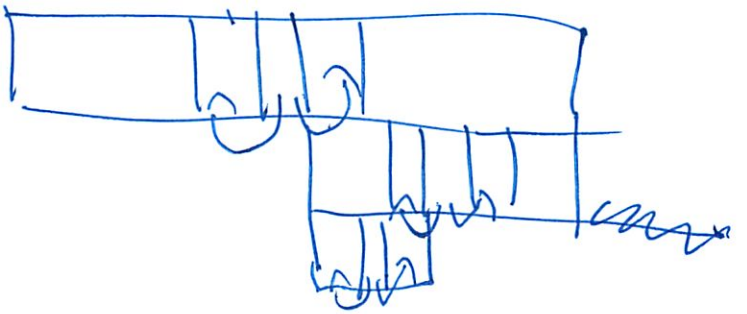
- Linked Data structures
- Divide + conquer
- Hashing
- Sorting
- Graphs
- DP — number theory

29

1D Peak finding

Naive: scan whole list

Or compare middle w/ neighbors
go that direction



$T(n) = T(n/2) + \theta(1)$

$= \underbrace{\theta(1) + \cdots + \theta(1)}_{\log n \text{ ties}}$

? I understand where this comes from much better!

2D

Recycle 1D

By doing each col ind.

Or (exact details NOT important!)

$$T(n,m) = T(n, m/2) + \Theta(n)$$

$$= \underbrace{\Theta(n) + \cdots}_{\log m \text{ times}}$$

$$= \Theta(n \log m)$$

Master Method

$$T(n) = a\,T(n/b) + f(n)$$

1. $f(n) = O(n^{\log_b a - \epsilon})$
   then $T(n) = \Theta(n^{\log_b a})$

2. $f(n) = \Theta(n^{\log_b a})$
   then $T(n) = \Theta(n^{\log_b a} \log n)$

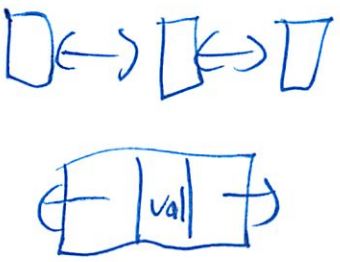
3. $f(n) = \Omega(n^{\log_b a + \epsilon})$

This is the hard way ...

(31)

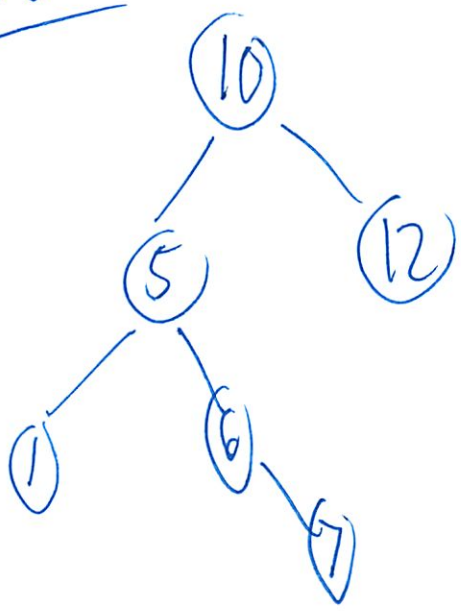
continue lectures - then find when see it

Linked List



blocks

BST



$\leq$ left

$\geq$ right

(note can be =

insert, etc all all $\log n$ avg case

n worst case

h the height actually

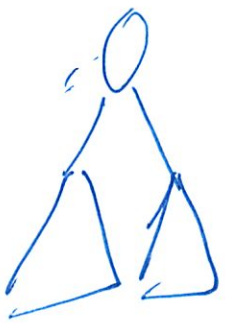
32

insert pretty straight forward
just walk down tree
till get to end of branch
Ti sure its always at end? Yes - thats the code

delete = very complex

---

red-black - a form of balanced
- makes sure



are = $\pm$ n-1
Ti correct

- forget exact balance mech, somewhat complicated

Goal for answering ovr runway problem
Everything $\leq$ or $\geq$ than a ~~ttt~~
w/ augment of # of subtres

Back to master theorm

$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n)$$

(same notation as before...)
thought it was diff

$a \geq 1$ branch factor

$b > 1$ how much to shrink on recurrence

$$a\,f\left(\frac{n}{b}\right) \leq c\,f(n)$$

In this class balanced = ~~AVM~~ AVL
is ± 1 allowable diff invariant
height = $\log n$

(34)

Here is the easy Master Theorm

$n^{\log_b a}$

$n^c \cdot (\ln n)^d$

1. Compare "c's"
   If "c"s diff - bigger c wins
   Polynomically diff

   eg $n(\lg n)^3 >> \sqrt{n}(\lg n)^3$

   $n^1 >> n^{1/2}$

2. Next compare d
   Only slightly diff

~~this doesn't~~ ~~to recurse~~

$n(\lg n)^3 < n(\lg n)^{10}$

35

Case 1 if sig diff
ans is bigger one

Case 2 slight diff
ans to recurrence is bigger one $\cdot \log n$

Oh so what are we comparing

$a\,T\left(\frac{n}{b}\right)$ — recursive runtime

$O(f(n))$ — top level runtime

$n^{\log_b a}$ $\qquad$ $f(n)$

eg

$$T(n) = T\left(\frac{n}{2}\right) + \Theta(1)$$

$n^{\log_2 1}$

$\log 1 = 0$

$n^0 = 1$ vs 1

Same, so $1 \cdot \log n = \log n$

(36)

$$T(n) = T\left(\frac{n}{2}\right) + O(n)$$

$n^{\log_b a}$

n

So $n^{\log_2 1}$

n^0

1 vs n

τ bigger $= n$ ✓

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

$n^{\log_2 2}$

n^1

n vs n

$n \log n$ ✓

$$T(n) = 25T\left(\frac{n}{2}\right) + n^2 \log_n 3$$

$n^{\log_2 25}$

$n^{4.6644}$ $\qquad$ n^2

τ bigger

(37)

I have not done that in a while!

Opps copy error

$$25\,T\left(\frac{n}{5}\right) + n^2 \lg^3 n$$

$$n^2 \quad \text{vs} \quad n^2 \lg^3 n$$

So compare d

bigger one : $\log n$

$$n^2 \lg^4 n$$

Oh didn't divide

$$\frac{n^2}{\lg^3 n}$$

So like

$d=0$ $\quad$ $d=-3$

↑ here

$n^2 \lg n$ ✓

Skipping Pg wisdom

Dictionaries

insert
delete
find

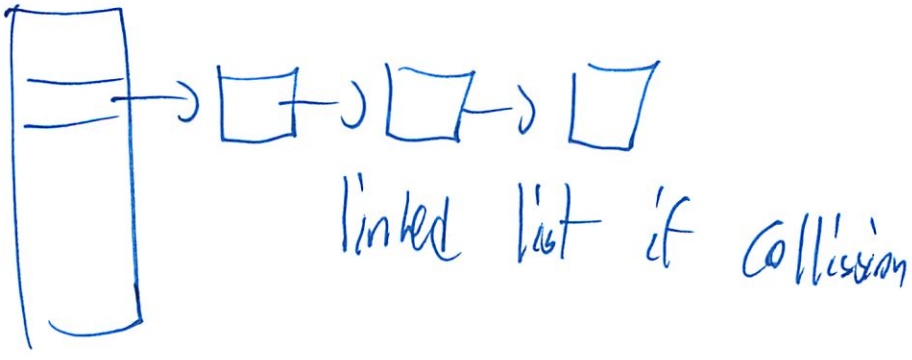
powered by hashing

Hashing

$h: U \to \{1, \ldots, m\}$

puts stuff in buckets

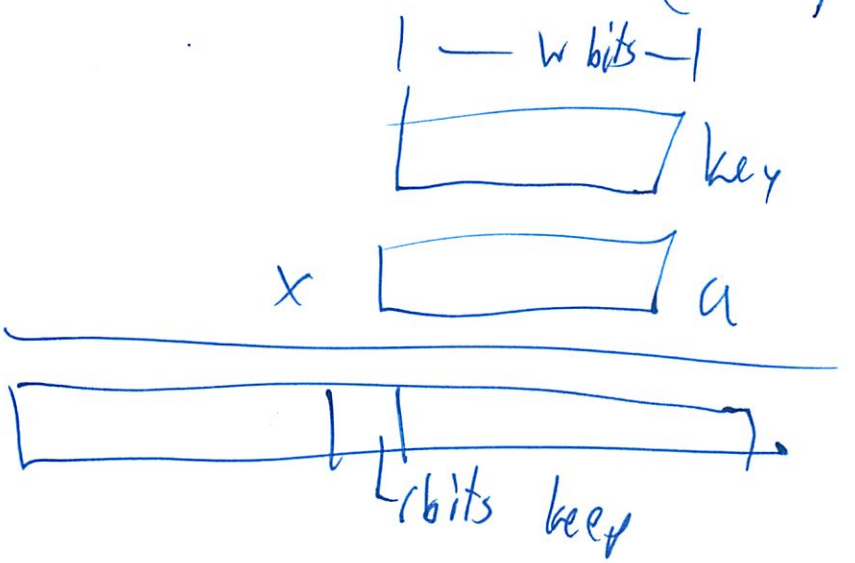
→ So insert/find is ~ lenght of key
= time to hash



Division Hash Fn

$h(k) = k \bmod m$

Collide if $k_1 = k_2 \pmod m$



(It seems that the actual hashing alg is a bit hardware)

---

p should be prime ...
for poly hash

$\frac{n}{p}$ collision prob

(4C)

Rolling Hash (Rabin-Karp)

Best for seeing if P is a substring of S

P has len L

S has len n

1. Hash P O(L)
2. Iterate through L O(nL)
3. Compare to verify no collisions

Initial

$$h(k) = (k[0]\,10^4 + k[1]\cdot 10^3 + k[2]\cdot 10^2 + k[3]\,10^1 + k[4]\,10^0) \bmod m$$

Then next

$$h(S_{i+1}) = [(h(S_i) - (10^5 \cdot \text{first digit } S_i)) \cdot 10 + \text{next digit after } S_i] \bmod m$$

1. Subtract out lst
2. Multiply all by 10 (mod of course)
3. Add on last digit

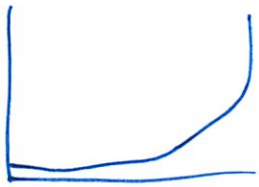
41

Resize when $n \geq 2m$

└ amortized analysis - avg over all cases

Open addressing - probe till find empty

SHUA - that hashing result is uniform

 = P (collision)

~~Uniform~~

Universal hashing family - keyed hash?

└ seems like it since pick a hash from a family of hashes

Sorting

Want $A[1] \leq A[2] \leq \ldots \leq A[n]$

~~Insertion~~

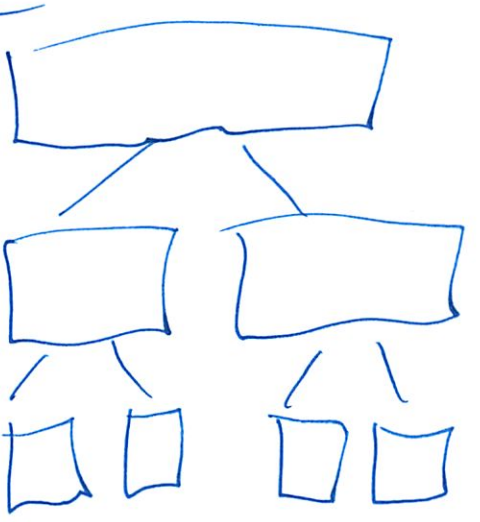
Insertion Sort

(the one at start of text book)

as go through deck of cards, insert 1 at a time to right place

$O(n^2)$

Merge Sort



Split in half + half again

Then reassemble in order

$O(n \lg n)$

(13)

Data structure: Priority queue

insert (S, x)
put x into S

~~increase~~ max

implemented w/ Heap

Heap

Actually an array
But shown as a binary tree

~~Max~~

min-heap property $A[parent(i)] \leq A[i]$

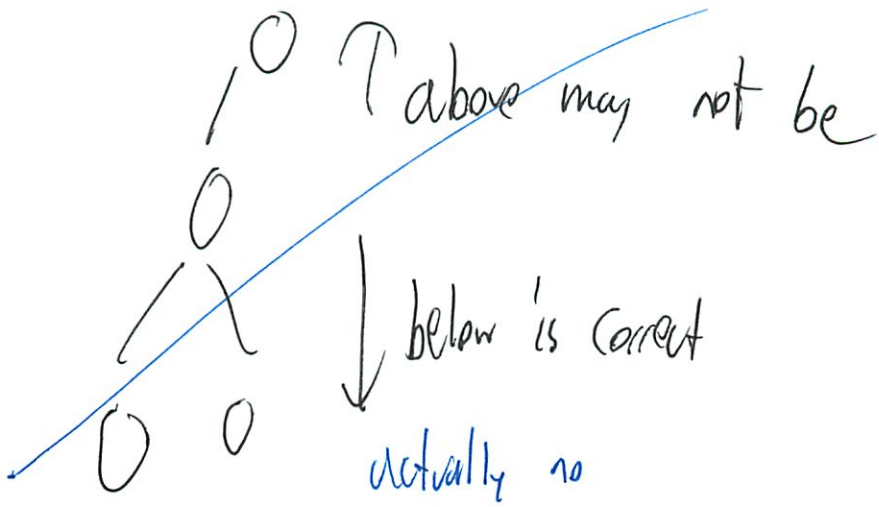
height = $\lg n$

Max heapify $O(\lg n)$ maintains property

assumes left(i) and right(i) are max heaps
but A[i] might be smaller than children

(14)

Basically



lets A[i] float down

↓moves it down

for a specific i

Build max heapify — does max heapify on all

$O(n \lg n)$

Heapsort

Sorts it in order by removing items (from end)

~~from the~~ remember heap sort is only

sorted ~~like~~ to the level

↳within level arbitrary

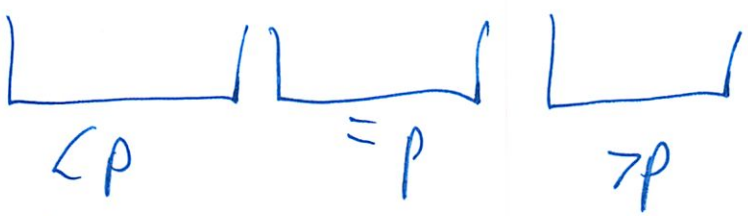
So it removes max
then calls max heapify on spot 1
repeats till all items in array
$O(n \lg n)$

Parent = $\lfloor \frac{i}{2} \rfloor$ left = $2i$ right = $2i+1$

Quicksort

Randomized $\Rightarrow O(n \lg n)$

picks a pivot



divide + conquer

quicksort (A, p, r)

quicksort (A, p, q-1) (A, q+1, r)

(46)

So slowly (1 by 1) sorts to < pivot or ≥ pivot

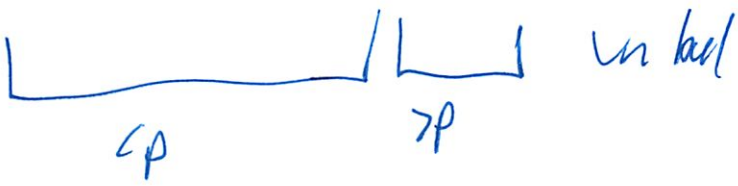
Then recurse on that remaining group

Can't be bad

Balanced → merge sort $O(n \lg n)$

Unbal → insertion sort $O(n^2)$

└ means



Randomize - to get good over all inputs

randomly picks a pivot

$O(n \lg n)$

47

Linear time sorts

Comparison sort

- the algs we saw befor
- $O(n \lg n)$
- decision tree model
  - go right or left based on comparison
  - so no better than $O(n \lg n)$ worst case

Counting sort

- Count the # of occurances of int
- $O(n + k)$
  - k # of possibilities
- is stable w/ satellite data
  - same order comes out

↑ How does this work?

Radix sort

Card machines

Most sig digit 1st

Graphs, Representations, Search

Adj list

- L for each vertex, list neighbors
- good for sparse graphs

incidence list

- for every vertex list edges

adj matrix

	vertices
vertices	1 if edge

good for calcing A^2 = # length 2 paths

- good for dense graphs

49

Breath First

Greedy

Coloring

queue

$O(n+m)$

Depth First

optimistic

like exploring a maze

~~Started~~ coloring as well

↳ start + finish times

Parentese structure

Stack

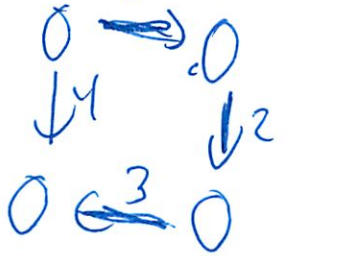
$O(n+m)$

Types of Edges

1. Tree edges
   - L regular
   - L in DFS forest
2. Back edge
   - L reach back to an ancestor

directed only:

3. Forward edge
   - Connect to a decendent (non tree)
   - from an edge other than the one that was explored
4. Cross edges
   - All others



Codes explored in

51

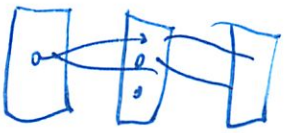
Topo sort

Call DFS
Record in order finished
For a DAG

Strongly Connected Components

all vertices are reachable from all other in the group

Spanning tree

build layers 

can't skip
but can do back

Topo Revers

(topo sort then reverse) from DFS
lets us find sinks

52

Dijstras

Single source shortest path

weighted graph — why you have to vs DFS

* all edge non neg — why you can vs B-F

extract min from queue

- greedy

then relax all edges from that

- see if we can update shortest path estimates

running time depends on how queue implemented

Remember having a lot of trouble w/ this

$O((V+E) \lg V)$ binary min-heap

$O(V \lg V + E)$ Fib heap

Should better understand how put together

53

Bellman Ford

Weighted graph
Allow ⊖

for ~~every~~ # of vertices
  for every edge ∈ in a specific order
    relax

neg weight cycles → return false
O(VE)

DAG

Topo Sort
Go through once along the order
Θ(V+E)

54

Heuristics

Google Maps is too slow w/ normal Dij

So bi-directional - start from each side
pick shortest from both

Planar
use as crows fly distance

A* has a potential h()
aka guess
heuristic
use a consistent heuristic

Can bi-dir A* - hard

5/22

Dynamic Programming

Fib as example

Basically save results (memoization) of subproblems
And use those results to make subproblems
easier to solve

Is both a top-down and bottom-up way to visualize

All-pairs shortest path

output is usually a table
Could run Dij for each node
Is stupid $O(V^3)$
Or Bellman Ford
$O(V^4)$

Uses adj matrix
also compute a predecessor matrix
$\Pi = \pi_{ij}$

DP approach to this

- each loop is very similar to matrix multip.

Remember the 4 steps

1. Char. the optimal structure
2. Recursively define the value of an optimal sol
3. Compute value of optimal sol bottom up
4. Construct an optimal sol

I never really understood this →

So recursive sol

$l_{ij}^{(m)}$ is min weight of any path i to j w/ at most m edges

So start w/

$$l_{ij}^{0} = \begin{cases} 0 & \text{if } i=j \\ \infty & i \neq j \end{cases}$$

Then $l_{ij}^{m} = \min l_{ij}^{(m-1)}$ and l_{ij}^{m} from looking at all possible predecessors

(57)

ℓ lengths notation!

So $l_{ij}^{m} = \min\left(l_{ij}^{m-1}, \min_{1 \le k \le n}\left\{l_{ik}^{(m-1)} + w_{kj}\right\}\right)$

$= \min_{1 \le k \le n}\left\{l_{ik}^{(m-1)} + w_{kj}\right\}$

↑ Since $w_{jj} = 0$ for all j

Basically grow one at a time

look back and find the shortest

$\delta(i,j)$ = actual shortest path weights

Can't have more than $n-1$ int. nodes

So now compute some matrices $L^1\ L^2\ L^3$

$L^m = l_{ij}^{m}$

L^{n-1} is final, has shortest path weights

L^1 is the basic one $= W$

Key is Extend-Shortest-Paths (L, W)

```
for i → 1 to n
    j → 1 to n
        k → 1 to n
            l_ij = min(l_ij, l_ik + w_kj)
```

$\Theta(n^3)$ Then do this for $m \to 2$ to $n-1$

→ Can see this is just matrix multiplication

And don't need each m → just $n-1$

So repeated squaring trick

$\Theta(n^3 \lg n)$

Floyd ~~Warshall~~ Warsall

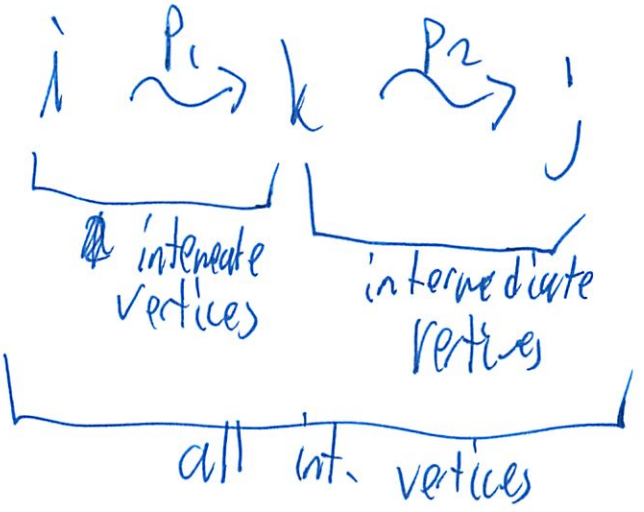
Diff DP formulation

$\Theta(V^3)$

neg weights allowed – but not neg cycles

Considers the intermediate vertices of a shortest path

So confusing set stuff



Recursive sol

same as before

$$d_{ij}^{0} = w_{ij}$$

$$d_{ij}^{k} = \begin{cases} w_{ij} & k = 0 \\ \min\left(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1}\right) & k \geq 1 \end{cases}$$

Code for bottom up

seems the same

So the diff seems to be the introduction of an "intermediate" vertex

Then comm for actually printing shortest path w/ Π the predecessor matrix

$$\Pi_{ij}^{0} = \begin{cases} nil & \text{if } i = j \quad w_{ij} = \infty \\ i & i \neq j \quad w_{ij} < \infty \end{cases}$$

$$\Pi_{ij}^{k} = \begin{cases} \Pi_{ij}^{k-1} & \text{if } d_{ij}^{k-1} \leq d_{ik}^{k-1} + d_{kj}^{k-1} \\ \Pi_{kj}^{k-1} & \text{if } d_{ij}^{k-1} > d_{ik}^{k-1} + d_{kj}^{k-1} \end{cases}$$

Transitive closure

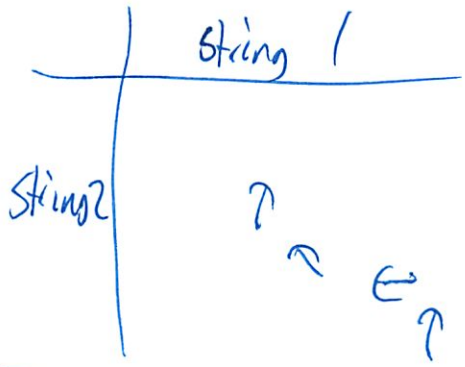
$$E^{*} = \{(i,j) \text{ there is path from } i \text{ to } j \text{ in } G$$

Too far afield now

(61)

Notes have that top down vs bottom up recursion thing
↳ save a lot by memoizing previous ans

LCS b/w 2 strings can use trail table



---

knapsack

not possible for arbitrary size

~~xxxxxx~~ given a set of items each w/ weight + value
find best combo so weight < limit and
value largest

(linear programming)

WP
- 0 or 1 of each item
- or up to x_i of each
- or unbounded

(62)

Or Text doc layout function

Or vertex cover

what was the point of this?

Then back to Number Theory

Also Java

5/22

Introduction to Algorithms
Massachusetts Institute of Technology
Professors Erik Demaine, Piotr Indyk, and Manolis Kellis

May 19, 2011
6.006 Spring 2011
Final Exam

Final Exam

- Do not open this exam booklet until directed to do so. Read all the instructions on this page.
- When the exam begins, write your name on every page of this exam booklet.
- You have 180 minutes to earn **180** points. Do not spend too much time on any one problem. Read them all through first, and attack them in the order that allows you to make the most progress.
- This exam booklet contains 20 pages, including this one. Two extra sheets of scratch paper are attached. Please detach them before turning in your exam at the end of the exam period.
- This exam is closed book. You may use **three** handwritten, ~~$8\frac{1}{2}'' \times 11''$ or A4 crib~~ sheets (both sides). No calculators or programmable devices are permitted. No cell phones or other communications devices are permitted.
- Write your solutions in the space provided. If you need more space, write on the back of the sheet containing the problem. Pages may be separated for grading.
- Do not waste time and paper rederiving facts that we have studied. It is sufficient to cite known results.
- Show your work, as partial credit will be given. You will be graded not only on the correctness of your answer, but also on the clarity with which you express it. Be neat.
- Good luck!

Problem	Parts	Points	Grade	Grader		Problem	Parts	Points	Grade	Grader
1	10	30				7	–	10		
2	8	40				8	–	10		
3	3	15				9	–	15		
4	–	10				10	–	10		
5	–	10				11	–	20		
6	–	10				Total		180		

Name: _____

Athena username: _____

Recitation:	Nick	Nick	Tianren	David	Joe	Joe	Michael
	WF10	**WF11**	**WF12**	**WF1**	**WF2**	**WF3a**	**WF3b**

Problem 1. True or false [30 points] (10 parts)

For each of the following questions, circle either T (True) or F (False). **Explain your choice.** (Your explanation is worth more than your choice of true or false.)

(a) **T** / **F** For all positive $f(n)$, $f(n) + o(f(n)) = \Theta(f(n))$.

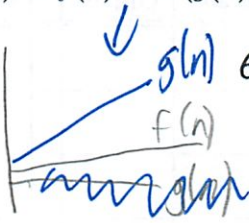
~~lower bound~~

Θ tight
O upper
Ω lower
o upper - not asy
ω - lower not asy

No - doesn't make any sense

True
(but not explained

(b) **T** / **F** For all positive $f(n)$, $g(n)$ and $h(n)$, if $f(n) = O(g(n))$ and $f(n) = \Omega(h(n))$, then $g(n) + h(n) = \Omega(f(n))$.



← the upper bound is g

So if add - since asy like $\frac{1}{2} + 2 = 1$

basically <1 and >1 are lower

don't think so

$f(n) = O(g(n)) \rightarrow g(n) = \Omega(f(n))$

still don't get ...

(c) (T) F Under the simple uniform hashing assumption, the probability that three specific data elements (say 1, 2 and 3) hash to the same slot (i.e., $h(1) = h(2) = h(3)$) is $1/m^3$, where m is a number of buckets.

So get multiples $\frac{1}{m} \cdot \frac{1}{m} \cdot \frac{1}{m}$

Only for fixed bucket
(a particular bucket)

$\frac{1}{m^2}$ ← how did they get this?

(d) (T) F Given an array of n integers, each belonging to $\{-1, 0, 1\}$, we can sort the array in $O(n)$ time in the worst case.

? is -1 or 0 or 1 ?

Yes → Counting Sort

✓ $O(n+k)$ $k=3$

add 1 to all #s

↳ why? I think it would work otherwise -- depends on specific alg

(e) **T** (**F**) The following array is a max heap: [10, 3, 5, 1, 4, 2].

Every thing is smaller than parents

× no 3, 4 ✓

(f) **T** (**F**) RADIX SORT does not work correctly (i.e., does not produce the correct output) if we sort each individual digit using INSERTION SORT instead of COUNTING SORT.

~~Does~~ Insertion Sort is stable ✓ it is

- I guess it could be quite easy,

(g) (T) **F** Given a directed graph G, consider forming a graph G' as follows. Each vertex $u' \in G'$ represents a strongly connected component (SCC) of G. There is an edge (u', v') in G' if there is an edge in G from the SCC corresponding to u' to the SCC corresponding to v'.
Then G' is a directed acyclic graph.

new graph

So SCC is collapsed to a node

I believe ya can do that

✓

(h) (T) **F** Consider two positively weighted graphs $G = (V, E, w)$ and $G' = (V, E, w')$ with the same vertices V and edges E such that, for any edge $e \in E$, we have $w'(e) = w(e)^2$. So 2nd graph just has weights squared
For any two vertices $u, v \in V$, any shortest path between u and v in G' is also a shortest path in G.

yeah doesn't change anything

Since pos thing ⊕ > 0

Oh paths not of = lenght

Duh – think!! I knew that!

(i) T (F) An optimal solution to a knapsack problem will always contain the object i with the greatest value-to-cost ratio v_i/c_i.

Wait on WP was a weight cutoff could be above that

Was that in our napsack problem?

√ Diff reason

Greedy doesn't work

if max weight = 2

$\frac{2}{1}$ vs $\frac{3}{2}$ & pick

Can you have multiple

(j) (T) F Every problem in NP can be solved in exponential time.

Forgot the def

I think NP is exp + ~~unsolvable~~

(those were kinda hit or miss tiring

Problem 2. Short answer [40 points] (8 parts) oh I dislike these

(a) Rank the following functions by increasing order of growth; that is, find an arrangement g_1, g_2, g_3, g_4 of the functions satisfying $g_1 = O(g_2)$, $g_2 = O(g_3)$, $g_3 = O(g_4)$. (For example, the correct ordering of n^2, n^4, n, n^3 is n, n^2, n^3, n^4.)

$$f_1 = (n!)^{1/n} \qquad f_2 = \log n^n \qquad f_3 = n^{n^{1/2}} \qquad f_4 = n \log n \log\log n$$

f_2 f_1
f_3 f_2
f_4 f_4
f_1 f_3

total guess totally wrong

(b) Solve the following recurrences by giving tight Θ-notation bounds. You do not need to justify your answers, but any justification that you provide will help when assigning partial credit.

i. $T(n) = 4T(n/2) + n^2 \log n$
ii. $T(n) = 8T(n/2) + n \log n$
iii. $T(n) = \sqrt{6006} \cdot T(n/2) + n^{\sqrt{6006}}$

Master theorem

$n^{\log_b a}$ vs f

$2^2 = 4$ 1. $n^{\log_2 4}$

n^2 vs $n^2 \log n \to n^2 \log^2 n$ ✓

$2^3 = 8$ 2. $n^{\log_2 8}$ vs $n \log n$

$n^3 \to n^3$ ✓

3. $n^{\log_2 \sqrt{6006}}$

n^{6006} $n^{6.27}$ vs n^{77} $= n^{6.27}$ n^{77}

(how supposed to do w/o calc I don't get why!

(c) Give a recurrence $T(n) = \cdots$ for the running time of each of the following algorithms, along with the asymptotic solution to that recurrence:

i. Insertion sort
ii. Merge sort
iii. 2D peak finding (the fastest algorithm we've seen)

oh

forgot to solve

Insertion sort $T(n) = T(n-1) + O(n) \checkmark = O(n^2)$

Merge $T(n) = 2T(\frac{n}{2}) + O(n) = O(n \lg n)$

2D ; $T(n^2) = T(\frac{n}{2}^2) + O(n) = O(n^2)$

Review these

(that was in the notes! the final one was never given!

(d) Could a binary search tree be built using $o(n \lg n)$ comparisons in the comparison model? Explain why or why not.

upper bound not asy tight

Yes - isn't this a red/black and AVL tree

No or can sort in $o(n \lg n)$ time by building a BST and then doing in-order tree walk in $O(n)$

But a AVL BST is $O(\lg n)$ for insertion

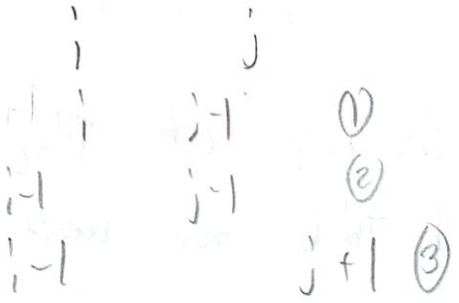
I don't get...

(g) In dynamic programming, we derive a recurrence relation for the solution to one subproblem in terms of solutions to other subproblems. To turn this relation into a bottom-up dynamic programming algorithm, we need an order to fill in the solution cells in a table, such that all needed subproblems are solved before solving a subproblem. For each of the following relations, give such a valid traversal order, or if no traversal order is possible for the given relation, briefly justify why.

i. $A(i,j) = F(A(i,j-1), A(i-1,j-1), A(i-1,j+1))$
ii. $A(i,j) = F(A(\min\{i,j\}-1, \min\{i,j\}-1), A(\max\{i,j\}-1, \max\{i,j\}-1))$
iii. $A(i,j) = F(A(i-2,j-2), A(i+2,j+2))$

solve smallest list

i)



or confused on format

Solve A(i,j) for i from 0 to n
for j from 0 to n

Extra ①

(h) Consider an array $A[1 \ldots n]$ of integers in the range $1 \ldots n^2$. A number a is a ***heavy hitter*** in A if a occurs in A at least $n/2$ times.

Give an efficient algorithm that finds all heavy hitters in a given array A.

reminds me of Parking wars

; Counting sort style list of integers
Count the # of of appearences
If > n/2, add to result
$O(n+n^2) = O(n^2)$

Radix sort + linear scan

Oh yeah that is better
I should have thought harder...

(e) Given n integers in the range $0 \ldots k$, describe how to preprocess these integers into a data structure that can answer the following query in $O(1)$ time: given two integers a and b, how many integers fall within the range $a \ldots b$?

BST augmented
- how many #s <
- then find augment b - augment a ✓

Same as counting sort they say
I think mine works

(f) Describe how any comparison-based sorting algorithm can be made stable, without affecting the running time by more than a constant factor.

(1) When find the right place, append at the end of that list of #s

Or augment to be decimal 7.1
7.2
or 2 part #s ✓

Extra ①

ii) $\min\{i,j\} - 1 \quad \min\{i,j\} - 1$

$\max\{i,j\} - 1 \quad \max\{i,j\}$

Solve $A(k,k)$ for k 0 to n

solve rest in any order

I don't get this at all...

iii) $A(i,j) = F(A(i-2, j-2), A(i+2, j+2))$

Impossible

Cyclic

Problem 3. You are the computer [15 points] (3 parts)

(a) Fill in the following grid with the correct subproblem solutions for this sequence alignment problem with these weights: 0 for mutation, 1 for insertion or deletion, and 3 for a match (the goal is to maximize the sum of the weights). Here "ATC" is the starting sequence and "TCAG" is the ending sequence.

What is the problem?

–	–	A	T	C
–	0	1	2	3
T	1	2	4	5
C	2	3	5	7
A	3	5	6	8
G	4	6	7	9

What is the optimal alignment?

(b) Draw a max-heap on the following set of integers: $\{2, 3, 5, 7, 11, 13, 17\}$. (You do not need to use the Build-Heap algorithm.)

17

13 11

7 5 3 2

no ans - but said almost everyone got this

(c) Compute $\sqrt[3]{6006}$ using two iterations of Newton's method, i.e., fill out the following table. Your entry for x_1 should be fully simplified. Your entry for x_2 can be left unsimplified.

i	x_i
0	1
1	
2	

Problem 4. Rotated array [10 points]

here is the tricky qu

Consider an array $A[1 \cdots n]$ constructed by the following process: we start with n distinct elements, sort them, and then rotate the array k steps to the right. For example, we might start with the sorted array $[1, 4, 5, 9, 10]$, and rotate it right by $k = 3$ steps to get $[5, 9, 10, 1, 4]$. Give an $O(\log n)$-time algorithm that finds and returns the position of a given element x in array A, or returns None if x is not in A. Your algorithm is given the array $A[1 \cdots n]$ but does *not* know k.

Nieve

Find min and find its pos $O(n)$
but not $\lg n$
essentially max heapify
Call max heapify on 1
see what pos the original # is

Then have k
return $x + k$

Sol 1 - modified binary search

2 - binary search on left and right
find smallest

Seems very similar

lol

Problem 5. Taxachusetts [10 points]

Suppose you are given a weighted graph $G = (V, E, w)$ of highways, and the state government has implemented a new tax rule whereby the cost of a path gets doubled as penalty if the number of edges in the path is greater than 10. Explain how to reduce finding the shortest-path weight between every pair of vertices (under this penalty) to the usual all-pairs shortest paths problem (as solved by Floyd-Warshall).

$d^{10} = d^9 \cdot 2$

d^9 as normal

just change the d^{10} step to double the cost of the d^9 step path

Or 10 is the # of nodes explored - we want actual - so not d^{10} but when $d^n = 10 \rightarrow$ double it

; anything more complex

1. Must augment to track path lengths
   - ah forgot that detail
   - d^n tracks weight sum not # edges totally missed

2. ~~Must~~ Must track weights for < 10 edges
   - ≥ 10 edges Why? Oh in case a len 9 is for higher weight w/o double - I see

Problem 6. Does this path make me look fat? [10 points]

Consider a connected weighted directed graph $G = (V, E, w)$. Define the ***fatness*** of a path P to be the maximum weight of any edge in P. Give an efficient algorithm that, given such a graph and two vertices $u, v \in V$, finds the minimum possible fatness of a path from u to v in G.

fatness = max weight

Go Dij - but track fatness

Can we do just by tracking fatness

No we need to track both weight fatness

Select by min fatness

but add to weight

Recalc fatness

✓

Don't forget correctness argument

Plus note neg weight edges normally break Dij - but not here!

Problem 7. Indiana Jones and the Temple of Algorithms [10 points]

While exploring an ancient temple, Prof. Jones comes across a locked door. In front of this door are two pedestals, and n blocks each labeled with its positive integer weight. The sum of the weights of the blocks is W. In order to open the door, Prof. Jones needs to put every block on one of the two pedestals. However, if the difference in the sum of the weights of the blocks on each pedestal is too large, the door will not open.

Devise an algorithm that Prof. Jones can use to divide the blocks into two piles whose total weights are as close as possible. To avoid a boulder rolling quickly toward him, Prof. Jones needs your algorithm to run in pseudopolynomial time.

Can he pick ind blocks?
Or are they ordered?

Like a balance tree
but w/ weights

Goal → W/2

Add 1 then add closest to diff $\frac{W}{2}$

1. Random on left
2. $\frac{W}{2}$ - left on right ← closest
3. Then again try to get closer

Greedy - prob not best
but how to avoid?

Smallest pile as close to $\frac{W}{2}$ w/o going over

DP! - not comfortable w/ knapsack-style problems

extra
2

Extra 2

Subproblem $P(i, w)$

└ true if pile $w \leq W/2$ using subset $0 \to i$
false otherwise

Initially $P(0, 0)$ true

$P(0, w)$ false for all $w \geq 0$

Recurrence $P(i, w) = P(i-1, w) \vee P(i-1, w - w_i)$

(what is this notation?

i = blocks subset 0 to i

W is target weight

So $P(i, w)$ is can we (T or F) make a group of blocks that is $w \leq W$
~~0 to~~ i
up to

So that fn is one less block or one less block and remove its weight

So this is work backwards (but really a trap)

Where do we start from?

Extra 3

$P(i-1, w-w_i)$ is false if $w_i > W$

so if that block is more than weight?

or if it goes 0 - makes sense

If $P(i, w)$ is true, record in table

L an d which is true

Find larges w for which $P(n, w)$ is true

all blocks

Then use B tables to find which blocks are in

$O(nW)$ subproblems $O(1)$ each $= O(nW)$

Tso for every weight → every block

Still don't get where you start from?

So use that formula for top down perspective

Extra 4

Say 5 blocks

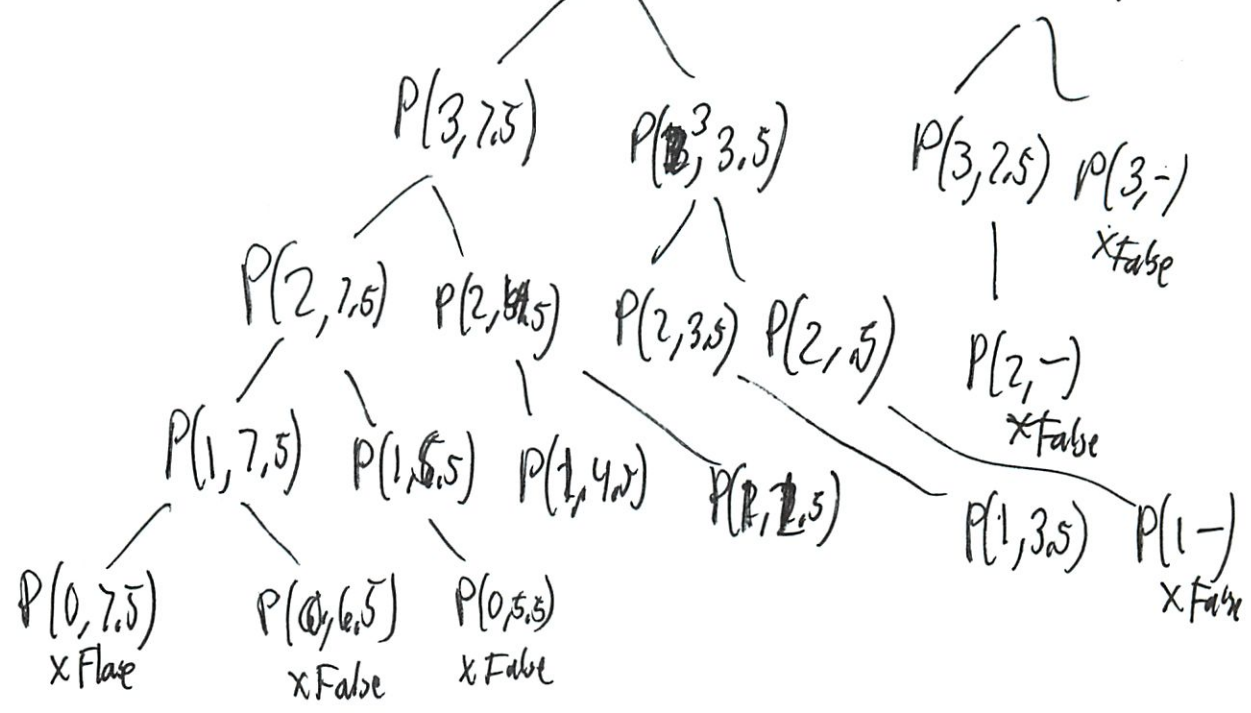
:	1	2	3	4	5
w	1	2	3	4	5

~~Want weight~~

So W = 15

~~Or~~ half = 7.5

Call P(5, 7.5) = P(4, 7.5) or P(4, 2.5)



but only true at P(0, 0) - so False

So need to try every weight, till true

Where is nW - just the tree - no that is $2^{T(n)+O(1)}$ I think

$n^{\lg_2 1}$

so 1 - but that is wrong

Still don't really get - but got TD vs BU in DP

extra 5

Yeah don't get recurrence / master theorm on DP problems

— was on a hw

That was how many subproblems

Problem 8. Claustrophobic chickens [10 points]

Prof. Tyson has noticed that the chickens in his farm frequently get claustrophobic. He wants to build a monitoring system that will alert him when this happens, allowing him to manually rearrange the chickens. After thorough research, Prof. Tyson determines that a chicken becomes claustrophobic if it is within 2 feet of at least 8 other chickens.

Prof. Tyson has installed a tracker on each chicken, which reports the (x, y) coordinates (measured in feet) of each of his chickens. Suppose that there are n chickens whose locations are represented as a sequence $(x_1, y_1), \ldots, (x_n, y_n)$. Prof. Tyson needs an efficient algorithm to determine whether there are any claustrophobic chickens (causing Prof. Tyson to go out and manually rearrange the chickens). Devise such an algorithm and save the chickens!

Oh boy!
- sort in order - DP?
- Or graph

Have no clue how to start

2x2 grid?

Nieve → n^2 - check every chicken w/ each other

- must be on int point?
  - could do hash grid

Square grid $\sqrt{2} \times \sqrt{2}$ square

Hash point into bucket

If ≥ 9 points → yes

Otherwise → calc distance to all pts in cell

Count those points

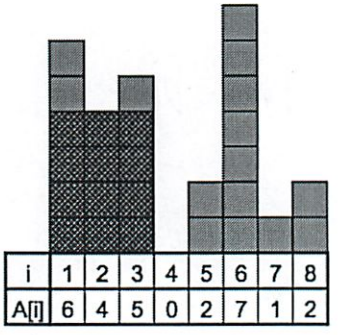
Extra 6

$O(n)$

Problem 9. Architects 'R Us [15 points]

You are assisting Prof. Gehry with designing the shape of a new room in the Stata Center. The professor has given you n columns, each of the same unit thickness, but with different heights: $A[1], A[2], \ldots, A[n]$. He asks you to permute the columns in a line to define the shape of the room. To make matters difficult, MIT wants to be able to hang a large rectangular picture on the columns. If j consecutive columns in your order all have a height of at least k, then we can hang a rectangle of size $j \cdot k$.

The example below contains 3 consecutive columns with heights of at least 4, so we can hang a rectangle of area 12 on the first three columns.



i	1	2	3	4	5	6	7	8
A[i]	6	4	5	0	2	7	1	2

(a) Give an efficient algorithm to find the largest area of a hangable rectangle for the *initial* order $A[1], A[2], \ldots, A[n]$ of columns.

This is the same as the football player problem,
Rewrite to practice
No wait is it - I think it is
Sort highest to lowest $O(n \lg n)$
Then say Max(cols 1, 2, 3, 4 etc)
include or not
but those don't recurse $O(n)$

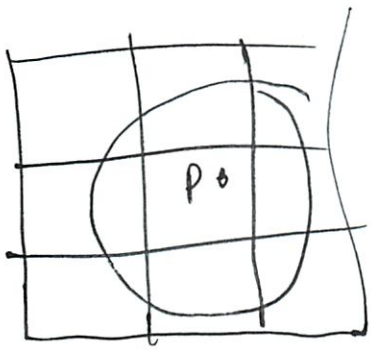
Extra 7

extra 6

So it does not have to be int?

Since check all neighboring cells

- calc distance to all neighboring cell



So this is not n^2 since just checking 9 cells?

instead of each

I see how this works

Kinda seems like cheating - but I guess it works...

Extra 9

Simplest ~~Klee's~~

~~Biggest rectangle is bounded by some col~~

Guess a col i

Go left + right counting cols

How do we guess this?

Total DS

Other

m[i,j] is min of interval A[i], ... A[j]

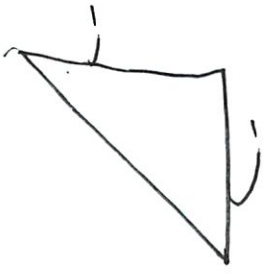
$m[i,j] = \min\{m[i,j-1], A[j]\}$

Now sol is $\{m[i,j] \cdot (j-i+1) : i \le j\}$

So was what I had it?

Also review notes on 2D DP

it was



So is this a 2D one? - Don't think so

extra 7

$O(n^2)$

worse than mine → guess I was wrong

Ohhh can't rearrange cols!

Let me rethink then

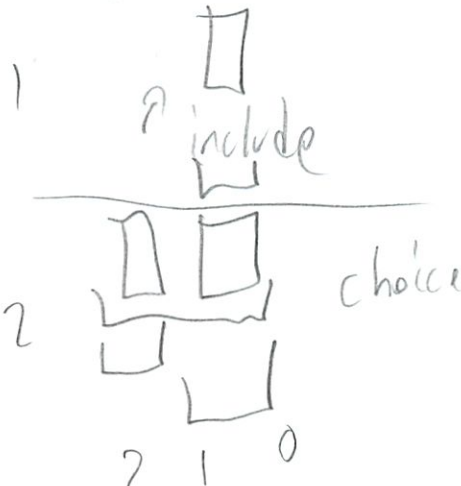
now standard DP

$DP(n) = \max$ (include or not)

from i to n

= Cmax on start can vary

Do from end



$DP_{2\to 0} = \max(2\to 1, 2\to 0, 1\to 0)$

$DP_3 = \max$

Extra 8

So 3 choices
just that
best of DP 2
Or all 3

$DP(3) = (A[3], DP(2), A[3] + DP(2))$

↑ from end

? what is this exactly
Can't just add
So just return

square $[A[3] \cdot A[2], A[1]$
? find min
multiply by 3

? Can we DP that?
Something about saving min
? so sap to save

But this
is cost

(b) Devise an efficient algorithm to permute the columns into an order that maximizes the area of a hangable rectangle.

So no we can move? Yes

Don't we want 

~~Test~~ sort ✓ sort $O(n \lg n)$ ✓

Wrote sol before

Compute hangable in $O(n)$

✓ So exactly as I had before

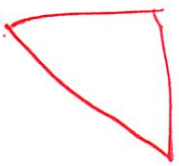
Problem 10. Guess Who? [10 points] These are all DPs...

Woody the woodcutter will cut a given log of wood, at any place you choose, for a price equal to the length of the given log. Suppose you have a log of length L, marked to be cut in n different locations labeled $1, 2, \ldots, n$. For simplicity, let indices 0 and $n+1$ denote the left and right endpoints of the original log of length L. Let d_i denote the distance of mark i from the left end of the log, and assume that $0 = d_0 < d_1 < d_2 < \ldots < d_n < d_{n+1} = L$. The ***wood-cutting problem*** is the problem of determining the sequence of cuts to the log that will cut the log at all the marked places and minimize your total payment. Give an efficient algorithm to solve this problem.

I don't get what it is asking...

Rt prob some sort of DP @

$$C(i,j) = \min_{i<k<j} \{ C(i,k) + C(k,j) + (d_j - d_i) \}$$



$O(n)$ each

$O(n^2)$ entries $\leq O(n^3)$

Greedy does not work

So it was the 2D - pa like the parenthesis qu

Problem 11. Varying variance [20 points]

For a set S of numbers, define its ***average*** to be $\bar{S} = \frac{1}{|S|}\sum_{s\in S} s$, and its ***variance*** to be $V(S) = \frac{1}{|S|}\sum_{s\in S}(s-\bar{S})^2$.

A ***segment variance data structure*** supports the following operations on an array $A[1\ldots n]$:

- **Assignment:** given an index i and a value a, set $A[i] = a$.
- **Segment variance:** given a pair of indices i and j, compute $V(\{A[i], A[i+1], \ldots, A[j]\})$.

Initially all entries in A are set to 0.

Design a data structure that supports both operations in $O(\log n)$ time.

Some sort of augmented tree

Or two

But ya can't avg 2 avgs

Or subtract

✓ but keep sum of all leaves and sum of squares of all leaves

Oh can subtract that

I was close...

SCRATCH PAPER

SCRATCH PAPER

Introduction to Algorithms
Massachusetts Institute of Technology
Professors Erik Demaine, Piotr Indyk, and Manolis Kellis

May 19, 2011
6.006 Spring 2011
Final Exam Solutions

Final Exam Solutions

Problem 1. True or false [30 points] (10 parts)

For each of the following questions, circle either T (True) or F (False). **Explain your choice.** (Your explanation is worth more than your choice of true or false.)

(a) T F For all positive $f(n)$, $f(n) + o(f(n)) = \Theta(f(n))$.

Solution: True.

(b) T F For all positive $f(n)$, $g(n)$ and $h(n)$, if $f(n) = O(g(n))$ and $f(n) = \Omega(h(n))$, then $g(n) + h(n) = \Omega(f(n))$.

Solution: True. This follows from $f(n) = O(g(n)) \Rightarrow g(n) = \Omega(f(n))$.

(c) T F Under the simple uniform hashing assumption, the probability that three specific data elements (say 1, 2 and 3) hash to the same slot (i.e., $h(1) = h(2) = h(3)$) is $1/m^3$, where m is a number of buckets.

Solution: False. The above formula only describes the probability of collision in a *fixed* bucket (say bucket number 1). The correct answer is $1/m^2$.

(d) T F Given an array of n integers, each belonging to $\{-1, 0, 1\}$, we can sort the array in $O(n)$ time in the worst case.

Solution: True. Use counting sort, e.g., after adding 1 to all numbers.

(e) T F The following array is a max heap: $[10, 3, 5, 1, 4, 2]$.

Solution: False. The element 3 is smaller than its child 4, violating the max-heap property.

(f) T F RADIX SORT does not work correctly (i.e., does not produce the correct output) if we sort each individual digit using INSERTION SORT instead of COUNTING SORT.

Solution: False. INSERTION SORT (as presented in class) is a stable sort, so RADIX SORT remains correct. The change can worsen running time, though.

3 points for correct answer and explanation
1 point for incorrect answer, but mentioning that radix sort needs to use a stable sort

(g) T F Given a directed graph G, consider forming a graph G' as follows. Each vertex $u' \in G'$ represents a strongly connected component (SCC) of G. There is an edge (u', v') in G' if there is an edge in G from the SCC corresponding to u' to the SCC corresponding to v'.
Then G' is a directed acyclic graph.

Solution: True. If there were any cycles in the graph of strongly connected components, then all the components on the cycle would actually be one strongly connected component.

3 points for correct answer

(h) T F Consider two positively weighted graphs $G = (V, E, w)$ and $G' = (V, E, w')$ with the same vertices V and edges E such that, for any edge $e \in E$, we have $w'(e) = w(e)^2$.
For any two vertices $u, v \in V$, any shortest path between u and v in G' is also a shortest path in G.

Solution: False. Assume we have two paths in G, one with weights 2 and 2 and another one with weight 3. The first one is shorter in G' while the second one is shorter in G.

3 points for correct answer - most people got this right

(i) T F An optimal solution to a knapsack problem will always contain the object i with the greatest value-to-cost ratio v_i/c_i.

Solution: False. Greedy choice doesn't work for the knapsack problem. For example, if the maximum cost is 2, and there are two items, the first with cost 1 and value 2, and the second with cost 2 and value 3, the optimal solution is to take just the second item.

3 points for correct answer

(j) T F Every problem in NP can be solved in exponential time.

Solution: True.

3 points for correct answer

Problem 2. Short answer [40 points] (8 parts)

(a) Rank the following functions by increasing order of growth; that is, find an arrangement g_1, g_2, g_3, g_4 of the functions satisfying $g_1 = O(g_2)$, $g_2 = O(g_3)$, $g_3 = O(g_4)$. (For example, the correct ordering of n^2, n^4, n, n^3 is n, n^2, n^3, n^4.)

$$f_1 = (n!)^{1/n} \qquad f_2 = \log n^n \qquad f_3 = n^{n^{1/2}} \qquad f_4 = n \log n \log\log n$$

Solution: The correct order if f_1, f_2, f_4, f_3

(b) Solve the following recurrences by giving tight Θ-notation bounds. You do not need to justify your answers, but any justification that you provide will help when assigning partial credit.

i. $T(n) = 4T(n/2) + n^2 \log n$

ii. $T(n) = 8T(n/2) + n \log n$

iii. $T(n) = \sqrt{6006} \cdot T(n/2) + n^{\sqrt{6006}}$

Solution:

i. Case 2 of the Master Method - $T(n) = n^2 \log^2 n$.

ii. Case 1 of the Master Method - $T(n) = n^3$.

iii. Case 3 of the Master Method to obtain $T(n) = \Theta(n^{\sqrt{6006}})$, checking that the regularity condition $af(n/2) = \sqrt{6006}\frac{n}{2}^{\sqrt{6006}} \leq cn^{\sqrt{6006}}$ for some $c < 1$. Condition holds for any $c > \sqrt{6006}/2^{\sqrt{6006}}$.

(c) Give a recurrence $T(n) = \cdots$ for the running time of each of the following algorithms, along with the asymptotic solution to that recurrence:

i. Insertion sort
ii. Merge sort
iii. 2D peak finding (the fastest algorithm we've seen)

Solution:

i. $T(n) = T(n-1) + O(n) = O(n^2)$
ii. $T(n) = 2T(n/2) + \Theta(n) = \Theta(n \lg n)$
iii. $T(n) = T(n/2) + \Theta(n) = \Theta(n)$

(d) Could a binary search tree be built using $o(n \lg n)$ comparisons in the comparison model? Explain why or why not.

Solution: No, or else we could sort in $o(n \lg n)$ time by building a BST in $o(n \lg n)$ time and then doing an in-order tree walk in $O(n)$ time.

(e) Given n integers in the range $0 \ldots k$, describe how to preprocess these integers into a data structure that can answer the following query in $O(1)$ time: given two integers a and b, how many integers fall within the range $a \ldots b$?

Solution: same preprocessing as for counting sort, then numbers in range is $C'(b) - C'(a)$.

(f) Describe how any comparison-based sorting algorithm can be made stable, without affecting the running time by more than a constant factor.

Solution: Tag elements with their original positions in the array, only increase by a factor of 2 at most

(g) In dynamic programming, we derive a recurrence relation for the solution to one subproblem in terms of solutions to other subproblems. To turn this relation into a bottom-up dynamic programming algorithm, we need an order to fill in the solution cells in a table, such that all needed subproblems are solved before solving a subproblem. For each of the following relations, give such a valid traversal order, or if no traversal order is possible for the given relation, briefly justify why.

i. $A(i,j) = F(A(i,j-1), A(i-1,j-1), A(i-1,j+1))$
ii. $A(i,j) = F(A(\min\{i,j\}-1, \min\{i,j\}-1), A(\max\{i,j\}-1, \max\{i,j\}-1))$
iii. $A(i,j) = F(A(i-2,j-2), A(i+2,j+2))$

Solution:

i. Solve $A(i,j)$ for(i from 0 to n: for(j from 0 to n))
ii. Solve $A(k,k)$ for(k from 0 to n) then solve rest in any order
iii. Impossible: cyclic.

(h) Consider an array $A[1 \ldots n]$ of integers in the range $1 \ldots n^2$. A number a is a ***heavy hitter*** in A if a occurs in A at least $n/2$ times.
Give an efficient algorithm that finds all heavy hitters in a given array A.

Solution: Radix-sort and linear scan.

Problem 3. You are the computer [15 points] (3 parts)

(a) Fill in the following grid with the correct subproblem solutions for this sequence alignment problem with these weights: 0 for mutation, 1 for insertion or deletion, and 3 for a match (the goal is to maximize the sum of the weights). Here "ATC" is the starting sequence and "TCAG" is the ending sequence.

–	–	A	T	C
–	0			
T				
C				
A				
G				

What is the optimal alignment?

Solution: The optimal alignment is to match the "TC" in both sequences.

-	-	A	T	C
-	0	1	2	3
T	1	2	4	5
C	2	3	5	7
A	3	5	6	8
G	4	6	7	9

3 points for correctly filled grid, 2 points for correct alignment of sequences

(b) Draw a max-heap on the following set of integers: $\{2, 3, 5, 7, 11, 13, 17\}$. (You do not need to use the Build-Heap algorithm.)

Solution: multiple possible solutions

5 points for a correct answer - (almost) everyone got this correct

(c) Compute $\sqrt[3]{6006}$ using two iterations of Newton's method, i.e., fill out the following table. Your entry for x_1 should be fully simplified. Your entry for x_2 can be left unsimplified.

i	x_i
0	1
1	
2	

Solution:

i	x_i
0	1
1	2002.6
2	$2/3x_1 + \frac{2002}{x_1^2}$

5 points for a correct answer

-2 for significant arithmetic/algebraic errors
-3 for errors that led to negative or very large solutions (should have been caught by simple sanity checks)

Problem 4. Rotated array [10 points]

Consider an array $A[1 \cdots n]$ constructed by the following process: we start with n distinct elements, sort them, and then rotate the array k steps to the right. For example, we might start with the sorted array $[1, 4, 5, 9, 10]$, and rotate it right by $k = 3$ steps to get $[5, 9, 10, 1, 4]$. Give an $O(\log n)$-time algorithm that finds and returns the position of a given element x in array A, or returns None if x is not in A. Your algorithm is given the array $A[1 \cdots n]$ but does *not* know k.

Solution: Solution I (to be fixed): You can perform a modified binary search.

```
SEARCH(A, i, j, x)
  first ← A[i]
  middle ← A[(i + j)/2]
  last ← A[j]
  if x ∈ {first, middle, last}:
    then
        return the corresponding index
    else :
        if (x < first and x > middle) or (x > first and x > middle): [xxx Isn't it equivalent to just x > m
          then
              return SEARCH(A, (i + j)/2, j, x)
          else :
              return SEARCH(A, i, (i + j)/2, x)
```

Solution II: Let $k \geq 0$ be the number of elements A was rotated by. We show how to identify the value of k, after which one can simply perform binary search in the "left part" $A[1 \ldots k]$ and the "right part" $A[k + 1 \ldots n]$. To this end, observe that all elements in the left part are not smaller than $A[1]$, while all elements in the right part are smaller than $A[1]$. By binary search we find the smallest $j > 1$ such that $A[j]$ is smaller than $A[1]$ (or set $j = 1$ if no such j exists). We then report $k = j - 1$.

Problem 5. Taxachusetts [10 points]

Suppose you are given a weighted graph $G = (V, E, w)$ of highways, and the state government has implemented a new tax rule whereby the cost of a path gets doubled as penalty if the number of edges in the path is greater than 10. Explain how to reduce finding the shortest-path weight between every pair of vertices (under this penalty) to the usual all-pairs shortest paths problem (as solved by Floyd-Warshall).

Solution: Augment to keep track of path lengths between each pair. We also augment so we keep track of the shortest-path weight between each pair using fewer than 10 edges and using greater than 10 edges.

Problem 6. Does this path make me look fat? [10 points]

Consider a connected weighted directed graph $G = (V, E, w)$. Define the ***fatness*** of a path P to be the maximum weight of any edge in P. Give an efficient algorithm that, given such a graph and two vertices $u, v \in V$, finds the minimum possible fatness of a path from u to v in G.

Solution: There are two good solutions to this problem.

We can see that that fatness must be the weight of one of the edges, so we sort all edge weights and perform a binary search. To test whether or not a path with a fatness no more than x exists, we perform a breadth-first search that only walks edges with weight less than or equal to x. If we reach v, such a path exists.

If such a path exists, we recurse on the lower part of the range we are searching, to see if a tighter fatness bound also applies. If such a path does not exist, we recurse on the upper part of the range we are searching, to relax that bound. When we find two neighboring values, one of which works and one of which doesn't, we have our answer. This takes $O((V + E) \lg E)$ time.

Another good solution is to modify Dijkstra's algorithm. We use "fatness" instead of the sum of edge weights to score paths, and the only change to Dijkstra itself that is necessary is to change the relaxation operation so that it compares the destination node's existing min-fatness with the max of the weight of the incoming edge (i, j) and the min-fatness of any path to i (the source of the incoming edge).

The correctness argument is almost precisely the same as that for Dijkstra's algorithm. A correct solution also had to note that negative-weight edges, which could be present here and normally break Dijkstra, don't do that here; adding negative numbers produces ever-more-negative path weights, but taking their max doesn't. This solution has the same time complexity as Dijkstra's algorithm.

Two less-efficient solutions were to use Bellman-Ford instead of Dijkstra (with the same modified relaxation step and an analagous correctnes argument) and to perform the iterative search linearly instead of in a binary fashion. These recieved partial credit.

Problem 7. Indiana Jones and the Temple of Algorithms [10 points]

While exploring an ancient temple, Prof. Jones comes across a locked door. In front of this door are two pedestals, and n blocks each labeled with its positive integer weight. The sum of the weights of the blocks is W. In order to open the door, Prof. Jones needs to put every block on one of the two pedestals. However, if the difference in the sum of the weights of the blocks on each pedestal is too large, the door will not open.

Devise an algorithm that Prof. Jones can use to divide the blocks into two piles whose total weights are as close as possible. To avoid a boulder rolling quickly toward him, Prof. Jones needs your algorithm to run in pseudopolynomial time.

Solution: Just consider the smaller of the two piles. The goal is to make the weight of this pile as close to $W/2$ as possible, while not exceeding that weight. Our guess for each block is whether the block is included in this smaller pile or not.

Our subproblems are $P(i, w)$, which has the value "true" if it is possible to make a pile of weight $w \leq W/2$ with some subset of blocks 1 through i, and the value "false" otherwise. Initially, let $P(0, 0)$ be true, and $P(0, w)$ be false for all values of $w > 0$.

Let the weight of block i be w_i. Use the following recurrence:

$$P(i, w) = P(i-1, w) \vee P(i-1, w-w_i)$$

(where $P(i-1, w-w_i)$ is considered to be false if $w_i > w$). If $P(i, w)$ is true, record in a separate table $B(i, w)$ which of $P(i-1, w)$ or $P(i-1, w-w_i)$ was true (choose arbitrarily if they are both true).

Find the largest value of w for which $P(n, w)$ is true. Then backtrack using the table B to determine which blocks were included in this pile to achieve this weight. This algorithm runs in $O(nW)$ time, because there are nW subproblems, each of which takes constant time.

Problem 8. Claustrophobic chickens [10 points]

Prof. Tyson has noticed that the chickens in his farm frequently get claustrophobic. He wants to build a monitoring system that will alert him when this happens, allowing him to manually rearrange the chickens. After thorough research, Prof. Tyson determines that a chicken becomes claustrophobic if it is within 2 feet of at least 8 other chickens.

Prof. Tyson has installed a tracker on each chicken, which reports the (x, y) coordinates (measured in feet) of each of his chickens. Suppose that there are n chickens whose locations are represented as a sequence $(x_1, y_1), \ldots, (x_n, y_n)$. Prof. Tyson needs an efficient algorithm to determine whether there are any claustrophobic chickens (causing Prof. Tyson to go out and manually rearrange the chickens). Devise such an algorithm and save the chickens!

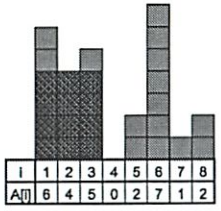
Solution: We shall solve this problem in a way similar to the close pair problem done in lecture, namely, partitioning the plane into a square grid and hashing points to corresponding buckets.

1. Impose a square grid onto the plane where each cell is a $\sqrt{2} \times \sqrt{2}$ square.
2. Hash each point into a bucket corresponding to the cell it belongs to.
3. If there is a bucket with ≥ 9 points in it, return YES.
4. Otherwise, for each point p, calculate distance to all points in the cell containing p as well as the neighboring cells. Return YES if the number of points within distance 2 is ≥ 8. Clearly the algorithm will find a clumped chicken if one exists, either in step 3 or in step 4. By using a hash table of size $O(n)$, we can make the above algorithm run in expected linear time.

Problem 9. Architects 'R Us [15 points]

You are assisting Prof. Gehry with designing the shape of a new room in the Stata Center. The professor has given you n columns, each of the same unit thickness, but with different heights: $A[1], A[2], \ldots, A[n]$. He asks you to permute the columns in a line to define the shape of the room. To make matters difficult, MIT wants to be able to hang a large rectangular picture on the columns. If j consecutive columns in your order all have a height of at least k, then we can hang a rectangle of size $j \cdot k$.

The example below contains 3 consecutive columns with heights of at least 4, so we can hang a rectangle of area 12 on the first three columns.



i	1	2	3	4	5	6	7	8
A[i]	6	4	5	0	2	7	1	2

(a) Give an efficient algorithm to find the largest area of a hangable rectangle for the *initial* order $A[1], A[2], \ldots, A[n]$ of columns.

Solution: The best algorithms we know run in $O(n^2)$ time.

The simplest $O(n^2)$ algorithm is the following. The biggest rectangle is bounded on top by some column. Guess that column i. So the height of the rectangle is $A[i]$. Now walk left until reaching a column of height $< A[i]$, and similarly walk to the right. Count the number k of traversed columns, and multiply by $A[i]$.

Another $O(n^2)$ algorithm is the following. Define $m[i,j]$ to be the minimum of the interval $A[i], \ldots, A[j]$. Then $m[i,j] = \min\{m[i,j-1], A[j]\}$, so by memoization, we can compute all $m[i,j]$'s in $O(n^2)$ time. Now the solution is $\max\{m[i,j] \cdot (j-i+1) : i \le j\}$, which takes $O(n^2)$ time to compute given the $m[i,j]$'s.

The easy brute-force algorithm already runs in $O(n^3)$ time (and was worth a base value of 4–5 out of 8 points for this part). Just use the computation above, but without memoizing the $m[i,j]$'s, so each takes $O(n)$ to compute, and we use $O(n^2)$ of them.

An $O(nh)$ algorithm, where $h = \max_i A[i]$, was worth a base value of 6 out of 8 points. We define one subproblem per (x,y) coordinate: $R(x,y)$ is the maximum possible area of a rectangle whose upper-right corner is at (x,y). There are $O(nh)$

such subproblems. To solve the subproblem, we can use the $O(1)$-time recurrence

$$R(x,y) = \begin{cases} R(x-1,y)+y & \text{if } A[x] \ge y \\ 0 & \text{otherwise.} \end{cases}$$

(b) Devise an efficient algorithm to permute the columns into an order that maximizes the area of a hangable rectangle.

Solution: The intended algorithm is to sort the columns in decreasing order, e.g., using merge sort in $O(n \lg n)$ time. This works because, if the correct height of a rectangle is k, then at best it can involve all columns with height $\ge k$, and these are consecutive in the sorted order. In fact, increasing order works just as well, as does a strange order (suggested by several students) of putting the maximum in the middle, then repeatedly placing the next smaller column alternately between the left and right sides of the construction so far.

We can compute the hangable rectangle in $O(n)$ additional time, though this was not necessary to receive full credit. For each prefix $B[1 \cdots i]$ of the sorted array, we'd like to compute $(\min B[1 \cdot i]) \cdot i$, and take the maximum over all i. But $\min B[1 \cdot i] = B[i]$, so this actually takes constant time per choice of i, for a total cost of $O(n)$ time.

2 out of 7 points were removed for lack of justification of sorting. 1 out of 7 points was removed for using counting (or radix) sort, which is not necessarily efficient given the setup of the problem.

Problem 10. Guess Who? [10 points]

Woody the woodcutter will cut a given log of wood, at any place you choose, for a price equal to the length of the given log. Suppose you have a log of length L, marked to be cut in n different locations labeled $1, 2, \ldots, n$. For simplicity, let indices 0 and $n+1$ denote the left and right endpoints of the original log of length L. Let d_i denote the distance of mark i from the left end of the log, and assume that $0 = d_0 < d_1 < d_2 < \ldots < d_n < d_{n+1} = L$. The ***wood-cutting problem*** is the problem of determining the sequence of cuts to the log that will cut the log at all the marked places and minimize your total payment. Give an efficient algorithm to solve this problem.

Solution: Dynamic programming. $c(i,j) = \min_{i<k<j}\{c(i,k)+c(k,j)+(d_j-d_i)\}$ where $c(i,j)$ is the min cost of cutting a log with left endpoint i and right endpoint j at all its marked locations. Start with $c(i,i+1)$ (consecutive cuts) and move outwards to $c(i,i+2)$, $c(i,i+3)$ until the maximal distance $c(1,n)$, which gives the optimal score for the whole wood. Remember pointers to k that gave max score at each step, and trace back pointers to construct optimal solution. Each iteration takes $O(n)$ (linear search between i and j) and there are $O(n^2)$ entries to fill (a triangle really, not a square). Greedy solutions that pick the maximum cut each time do not work. Similarly, heuristics like picking the point closest to the center do not work.

Problem 11. Varying variance [20 points]

For a set S of numbers, define its ***average*** to be $\bar{S} = \frac{1}{|S|}\sum_{s\in S} s$, and its ***variance*** to be $V(S) = \frac{1}{|S|}\sum_{s\in S}(s-\bar{S})^2$.

A ***segment variance data structure*** supports the following operations on an array $A[1\ldots n]$:

- **Assignment:** given an index i and a value a, set $A[i] = a$.
- **Segment variance:** given a pair of indices i and j, compute $V(\{A[i], A[i+1], \ldots, A[j]\})$.

Initially all entries in A are set to 0.

Design a data structure that supports both operations in $O(\log n)$ time.

Solution: We have $1/|S|\sum_{s\in S}(s-\bar{S})^2 = 1/|S|\sum_{s\in S} s^2 - \bar{S}^2$. Both terms can be maintained using an augmented data structure that, in each internal node, keeps (i) the sum of all leaves and (ii) the sum of squares of all leaves.

5/22 Pratice

Introduction to Algorithms
Massachusetts Institute of Technology
Professors Konstantinos Daskalakis and Patrick Jaillet

December 14, 2010
6.006 Fall 2010
Final Exam

Final Exam

- Do not open this quiz booklet until directed to do so. Read all the instructions on this page.
- When the quiz begins, write your name on every page of this quiz booklet.
- You have 180 minutes to earn 180 points. Do not spend too much time on any one problem. Read them all first, and attack them in the order that allows you to make the most progress.
- This exam is closed book. You may use **three** $8\frac{1}{2}'' \times 11''$ or A4 crib sheets (both sides). No calculators or programmable devices are permitted. No cell phones or other communications devices are permitted.
- Write your solutions in the space provided. If you need more space, write on the back of the sheet containing the problem. Pages may be separated for grading.
- Don't waste time and paper rederiving facts that we have studied. It is sufficient to cite known results.
- When writing an algorithm, a **clear** description will suffice. Pseudo-code isn't required.
- When asked for an algorithm, your algorithm should have the time complexity specified in the problem with a correct analysis. If you cannot find such an algorithm, you will generally receive partial credit for a slower algorithm **if you analyze your algorithm correctly**.
- Show your work, as partial credit will be given. You will be graded not only on the correctness of your answer, but also on the clarity with which you express it.

Problem	Parts	Points	Grade	Grader
1	2	2		
2	1	30		
3	2	30		
4	2	28		
5	3	30		
6	3	30		
7	2	30		
Total		180		

Name: _____

Friday Recitation:	Aleksander **11 AM**	Arnab **12 PM**	Alina **3 PM**	Matthew **4 PM**

Problem 1. What is Your Name? [2 points] (2 parts)

(a) [1 point] Flip back to the cover page. Write your name there.

(b) [1 point] Flip back to the cover page. Circle your recitation section.

Problem 2. Storing Partial Maxima [30 points] (1 part)

start, end

6.006 student, Mike Velli, wants to build a website where the user can input a time interval in history, and the website will return the most exciting sports event that occurred during this interval. Formally, suppose that Mike has a chronologically sorted list of n sports events with associated integer "excitement factors" $e_1, \ldots, e_n$. You can assume for simplicity that n is a power of 2. A user's query will consist of a pair (i, j) with $1 \leq i < j \leq n$, and the site is supposed to return $\max(e_i, e_{i+1}, \ldots, e_j)$.

clue!

Mike wishes to minimize the amount of computation per query, since there will be a lot of traffic to the website. If he precomputes and stores $\max(e_i, \ldots, e_j)$ for every possible input (i, j), he can respond to user queries quickly, but he needs storage $\Omega(n^2)$ which is too much.

↑ lower bound

In order to reduce storage requirements, Mike is willing to allow a small amount of computation per query. He wants to store a cleverer selection of precomputed values than just $\max(e_i, \ldots, e_j)$ for every (i, j), so that for any user query, the server can retrieve two precomputed values and take the maximum of the two to return the final answer. Show that now only $O(n \log n)$ values need to be precomputed.

BST
- bst interval
how deal w/ both
store max - but might be outside the range

Depends how large range is
- could store ~~max to~~ sorted list of e max → min
- seach through till find in range
O(n) worst case
thats actually pretty good

Extra!

Problem 3. Longest Simple Cycle [30 points] (2 parts)

Given an unweighted, directed graph $G = (V, E)$, a path $\langle v_1, v_2, ..., v_n \rangle$ is a set of vertices such that for all $0 < i < n$, there is an edge from v_i to v_{i+1}. A cycle is a path such that there is also an edge from v_n to v_1. A *simple path* is a path with no repeated vertices and, similarly, a *simple cycle* is a cycle with no repeated vertices. In this question we consider two problems:

- LONGESTSIMPLEPATH: Given a graph $G = (V, E)$ and two vertices $u, v \in V$, find a simple path of maximum length from u to v or output NONE if no path exists.
- LONGESTSIMPLECYCLE: Given a graph $G = (V, E)$, find a simple cycle of maximum length in G.

(a) [20 points] Reduce the problem of finding the longest simple path to the problem of finding the longest simple cycle. Prove the correctness of your reduction and show that it runs in polynomial time in $|V|$ and $|E|$.

But longest path might not be longest cycle
Cycle is just a special case ... longest path that folds back on itself
DFS $O(V+E)$

Path v

(b) [10 points] As we discussed in class, finding a longest simple path is NP-Hard. Therefore, there is no known algorithm that, on input u, v, and G returns a longest simple path from u to v in polynomial time. Using this fact (which you do not need to prove) and Part (a), show that there is no known polynomial time algorithm that can find a longest simple cycle in a graph.

Note: If you were unable to solve Part (a), you may assume an algorithm SIMPLEPATHFROMCYCLE for finding a longest simple path from u to v that runs in time polynomial in L, $|V|$, and $|E|$ where L is the running time of a black-box algorithm for solving LONGESTSIMPLECYCLE.

(That's what I thought

This seems pretty simple
A longest simple cycle is just a special case of path ✓
But write as contradiction "proof format"

extra 1

for each $1 \le i \le n/2$ store $\max(e_i, e_{i+1}, \ldots e_{n/2})$

So for every starting point store best up to $n/2$

for each $n/2 < j \le n$ same

Recurse sep on both lists

till list size 11

; What are we recursing on?

---

if $i \le n/2$ $j > n/2$ then return max of both

if $j \le \frac{n}{2}$ or $j > n/2$ ans found recursively

Again how?

- Does that mean normal way?

~~S(n) = 2S(n/2) + O(n)~~

$S(n) = 2S(n/2) + O(n)$

$= O(n \lg n)$ master theorem

I still like my sol better

extra 2

The Goal is way longer than the space to ans

Create a new graph G'

Copy all vertices + edges

Add |V| more ~~edge~~ vertices to G'

$w_1, \ldots, w_{|V|}$

For $0 < i < |V|$ create a directed edge from w_i to w_{i+1}

Also create directed edge $v \to w_1$

$w_{|V|}$ to u

Run longest simple cycle

That makes it more difficult — no

was not thinking that was possible

Problem 4. Closest pair [28 points] (2 parts)

We are interested in finding the closest pair of points in the plane, closest in the sense of the rectilinear distance (also called the Manhattan or L_1 distance). The rectilinear distance between two points p_1 and p_2 on the plane is defined as $d(p_1, p_2) = |x_1 - x_2| + |y_1 - y_2|$, where the x's and y's are the first and second coordinates, respectively. In the first part we consider a one-dimensional version of the problem as a warm-up. In both cases, coordinates of the points are real numbers with k significant digits beyond the decimal point, k constant.

So

(a) [8 points] Warm up - Provide an efficient way to find a closest pair among n points in the interval $[0, 1]$ on the line. Full credit will be given to the most efficient algorithm with a correct analysis.

So what are the points?
How are they paired
So smallest distance
Naive $O(n^2)$
On last exam this was grid + hash

Extra 3

(b) [20 points] Case of the plane - A divide and conquer approach for n points in the square $[0, 1] \times [0, 1]$. Here is a possible strategy. Divide the set of points into two sets of about half size: those on the right of the x-coordinate median, and those on the left. Recursively, find the closest pair of points on the right, and the closest pair of points on the left and let δ_r and δ_l be the corresponding distances. The overall closest pair is either the minimum of these two options, or corresponds to a pair where one point is on the right of the median and the other is on the left. Given δ_r and δ_l, there should be an efficient way to find the latter. Explain how; then write the full recurrence for the running time of this approach; and conclude with its overall running time.

very similar to what I had

don't forget the special case →

$O(n) = 2T(\frac{n}{2}) + O(n)$

$\log_2 2$

n vs n ← in worst cases all n pts on the strip

$O(n \log n)$

thought it was this

alright dividing on space - not points

Problem 5. APSP Algorithm for Sparse Graphs [30 points] (3 parts)

Let $G = (V, E)$ be a weighted, directed graph that can have some of the weights negative. Let $n = |V|$ and $m = |E|$, and assume that G is strongly connected, i.e., for any vertex u and v there is a path from u to v in G.

We want to solve all-pairs-shortest-path problem (APSP) in G, i.e., we want to either find all the vertex-to-vertex distances $\{\delta(v, u)\}_{v,u \in V}$, or report existence of a negative-length cycle. We will design an algorithm for this task that runs in $O(mn + n^2 \log n)$ time. (Note that when G is not dense, i.e., when m is $o(n^2)$, the running time of this algorithm is asymptotically better than the one of the Floyd-Warshall algorithm.)

(a) [5 points] Fix some vertex $t \in V$ and consider the vertex potential $\lambda_t(u) = \delta(u, t)$ where $\delta(u, t)$ is the shortest path from $u \in V$ to t. Give an algorithm for calculating $\lambda_t(u)$ for all $u \in V$ and analyze the running time.

(b) [5 points] Show $\lambda_t(u)$, the potential from Part (a), is a feasible potential even if some of the original weights are negative.

Extra 3

But needs to be some way to do diff in 2D

I guess it can be hash groups

But need right group size

Largest group = best

Or divide out

which half has more points

but that is $> \lg n$

$\frac{1}{2} + \frac{1}{4} + \frac{1}{8} = ?$ 1

So $O(n)$ sol

Radix sort $O(kn)$

Calc distance b/w it and one before $O(k)$

So total $O(n)$ since k fixed

I think my sol would have worked

↳ almost like binary search

(c) [20 points] Show how you use the vertex potential from Parts (a) and (b) to solve APSP in G in $O(mn + n^2 \log n)$ time, including the time it takes to calculate the vertex potential.

Problem 6. Airplane Scheduling [30 points] (3 parts) lecture

Consider the runway reservation system from PS2. Recall that we kept track of requested landing times from airplanes by storing them in a balanced binary search tree. In that problem set, we required, for safety, that no landing time be within three minutes of any other landing time in the tree. We showed we could insert into the tree, delete from the tree, and check the validity of a landing time in $O(\log n)$ time.

Sometimes severe weather hits, and the 3-minute window between flights just isn't safe, so a new window size is determined. In these cases, an extra runway might be opened up at a nearby airport to take flights which don't fit within the new window. For all parts, you may use data structure augmentations provided that you explain the augmentation. Its maintenance may not increase the asymptotic running time of other operations, but you are not required to prove this.

(a) [5 points] Provide a very fast (constant-time) algorithm to determine if there are any flights which are not valid with the new window so that the extra runway can start opening immediately. I don't fully remember the last alg

Basically used next largest

in order tree walk - checking every gap

Ston simple

(b) [15 points] Also provide a slower algorithm which locates which specific flights are not valid within the new window so they can be rescheduled. For example, assuming a window of 3 minutes with flights at times 28, 31, 35, 40, 43, 48, 53, 57, 60, if the window size were expanded to 4 minutes, the algorithm should return that 28, 31, 40, 43, 57, 60 are invalid.

Why is 28 invalid?

So both are canceled?

Wouldn't an in-order tree walk work?

Cool missing from prev)

track min window size of subtree

look at root if any window smaller than new window

ahh very clever - wanted O(1) time

Can traverse extra in O(n) - so sol to a works

(c) [10 points] Can the running time of this slower algorithm be improved if we assume bounds on the maximum size the window could become?

yes - can keep a side-structure

no extra details ;)

Problem 7. Traveling on a Budget [30 points] (2 parts)

too long

Arthur Dent has \$500 and 1000 hours to go from Cambridge, MA, to Berkeley, CA. He has a map of the States represented as a directed graph $G = (V, E)$. The vertices of the graph represent towns, and there is a directed edge $e = (A, B)$ from town A to town B if there is some means of public transportation connecting the two towns. Moreover, the edge is labeled with a pair (m_e, t_e), representing the cost $m_e \in \{0, 1, \ldots\}$ in dollars of transportation from A to B and the time $t_e \in \{0, 1, \ldots\}$ in hours that it takes to go from A to B.

oh geeze →

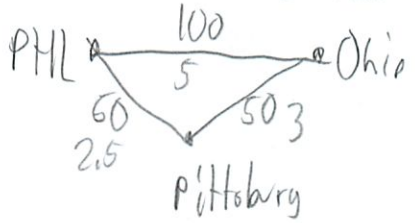
Arthur is interested in finding a path from Cambridge to Berkeley that does not cost more than \$500 and does not take more than 1000 hours, while also minimizing the objective $5M^2 + 2T^2$, where M is the cost of the trip in dollars and T is the duration of the trip in hours. He was looking for an algorithm that runs in time polynomial in $|V|$ and $|E|$ ...

(napsack don't be greedy DP 7 basic rules (things I learned studying)

(a) [10 points] Due to his lack of knowledge in algorithms, he gave up on the idea of respecting the budget and time constraints. At least he thought he could efficiently find the path minimizing the objective $5M^2 + 2T^2$. He tried modifying Dijkstra's algorithm as follows: If an edge e was labeled (m_e, t_e), he assigned it a weight $w_e = 5m_e^2 + 2t_e^2$, and ran Dijkstra's algorithm on the resulting weighted directed graph. Show that Arthur's algorithm may return incorrect results, i.e. return a path that does not minimize the objective $5M^2 + 2T^2$.

Something about 2 legs vs 1

Say



↙ this is 50050 ← when this is actually much better

12512 12518 = 25000

(pick smaller #s on actual exam)

(b) [20 points] Now provide an algorithm that solves Arthur's original problem in time polynomial in $|E|$ and $|V|$. Your algorithm should find the path that minimizes the objective $5M^2 + 2T^2$, while at the same time respecting the constraints $M \leq 500$ and $T \leq 1000$. Please describe your algorithm precisely, and justify its correctness and running time. More credit will be given to faster algorithms, provided that the analysis of the algorithm is correct.

[*Hint 1: Use dynamic programming.*]

[*Hint 2: For each town A, integer values $m \leq 500$ and $t \leq 1000$, either there is a path from Cambridge to A that requires cost m and time t, or there is not.*]

Oh how to do my

Cdan - I know that

Cost = min (
all edges
?

The fn

but how to avoid the problem we saw
- save the cost + time and re calc the function each time

506 x 15 matrix

- no further ans

but this is like the all pairs problem
- what I would have answered

SCRATCH PAPER

SCRATCH PAPER

Introduction to Algorithms
Massachusetts Institute of Technology
Professors Konstantinos Daskalakis and Patrick Jaillet

December 14, 2010
6.006 Fall 2010
Final Exam Solutions

Final Exam Solutions

Problem 1. What is Your Name? [2 points] (2 parts)

(a) [1 point] Flip back to the cover page. Write your name there.

(b) [1 point] Flip back to the cover page. Circle your recitation section.

Problem 2. Storing Partial Maxima [30 points] (1 part)

6.006 student, Mike Velli, wants to build a website where the user can input a time interval in history, and the website will return the most exciting sports event that occurred during this interval. Formally, suppose that Mike has a chronologically sorted list of n sports events with associated integer "excitement factors" $e_1, \ldots, e_n$. You can assume for simplicity that n is a power of 2. A user's query will consist of a pair (i, j) with $1 \leq i < j \leq n$, and the site is supposed to return $\max(e_i, e_{i+1}, \ldots, e_j)$.

Mike wishes to minimize the amount of computation per query, since there will be a lot of traffic to the website. If he precomputes and stores $\max(e_i, \ldots, e_j)$ for every possible input (i, j), he can respond to user queries quickly, but he needs storage $\Omega(n^2)$ which is too much.

In order to reduce storage requirements, Mike is willing to allow a small amount of computation per query. He wants to store a cleverer selection of precomputed values than just $\max(e_i, \ldots, e_j)$ for every (i, j), so that for any user query, the server can retrieve two precomputed values and take the maximum of the two to return the final answer. Show that now only $O(n \log n)$ values need to be precomputed.

Solution: We are given the list $e_1, \ldots, e_n$. For each $1 \leq i \leq n/2$, store $\max(e_i, e_{i+1}, \ldots, e_{n/2})$, and for each $n/2 < j \leq n$, store $\max(e_{n/2+1}, \ldots, e_j)$. Recurse separately on the two lists $e_1, \ldots, e_{n/2}$ and $e_{n/2+1}, \ldots, e_n$. Stop the recursion when the list size becomes 1.

If the user's query is (i, j) with $i \leq n/2$ and $j > n/2$, then we can return $\max(\max(e_i, e_{i+1}, \ldots, e_{n/2}), \max(e_{n/2+1}, \ldots, e_j))$. If both $i, j \leq n/2$ or $i, j > n/2$, then the answer is found recursively.

Let $S(n)$ be the number of values stored for a list of length n. By construction, $S(n) = O(n) + 2S(n/2)$, and therefore, $S(n) = O(n \log n)$.

Problem 3. Longest Simple Cycle [30 points] (2 parts)

Given an unweighted, directed graph $G = (V, E)$, a path $\langle v_1, v_2, ..., v_n \rangle$ is a set of vertices such that for all $0 < i < n$, there is an edge from v_i to v_{i+1}. A cycle is a path such that there is also an edge from v_n to v_1. A *simple path* is a path with no repeated vertices and, similarly, a *simple cycle* is a cycle with no repeated vertices. In this question we consider two problems:

- LONGESTSIMPLEPATH: Given a graph $G = (V, E)$ and two vertices $u, v \in V$, find a simple path of maximum length from u to v or output NONE if no path exists.
- LONGESTSIMPLECYCLE: Given a graph $G = (V, E)$, find a simple cycle of maximum length in G.

(a) [20 points] Reduce the problem of finding the longest simple path to the problem of finding the longest simple cycle. Prove the correctness of your reduction and show that it runs in polynomial time in $|V|$ and $|E|$.

Solution: Create a new graph G'. Copy all vertices and edges from G to G'. Then add $|V|$ more vertices to G', $w_1, ..., w_{|V|}$. For $0 < i < |V|$, create a directed edge from w_i to w_{i+1}. Also create a directed edge from v to w_1 and a directed edge from $w_{|V|}$ to u. Run LONGESTSIMPLECYCLE on G'. If the longest simple cycle involves vertices u and v, return the path from u to v in the cycle. Otherwise, output NONE.

Running Time: Creating the new graph takes $O(|V| + |E|)$. We run LONGESTSIMPLECYCLE once. Therefore, the running time of the algorithm is $O(|V| + |E| + L)$ where L is the running time of LONGESTSIMPLECYCLE.

Correctness: If a path from u to v exists then $c = \langle u, ..., v, w_1, ..., w_{|V|} \rangle$ is a cycle in G'. The cycle c contains at least $|V|+2$ vertices and, by construction, is the only cycle containing the w vertices. Therefore, all other cycles must have length at most $|V|$. Hence LONGESTSIMPLECYCLE will return this cycle if it exists and our algorithm will return $p = \langle u, ..., v \rangle$. Since the cycle is simple, p must contain only vertices in V and is thus the longest path from u to v in G. If the cycle does not exist, there is no path from u to v and the algorithm correctly outputs NONE.

Note: This is only one possible reduction. People came up with others and that's fine so long as they are correct and polynomial.

Grading:

- 20/20 For any polynomial time reduction. The reduction did not have to be linear to receive full credit.
- 19.5/20 if you added a weighted edge from v to u rather than a number of vertices (the graph was unweighted)
- 18/20 If you gave the reduction of adding an edge from v to u, finding the connected cluster containing v and u and running LSC on that. The problem is that the connected cluster containing v and u might have a simple cycle that does not involve v and u longer than the one that does involve v and u. The whole connected cluster is a cycle, but not a simple one.
- 17/20 For removing back edges
- 17/20 For removing all nodes u can't reach
- 14/20 If you tried to remove edges one at a time, which does not work, but the algorithm was still polynomial.
- 10/20 If you returned the longest path in the graph, but not necessarily from u to v.
- 10/20 If you did the reduction correctly in the wrong direction
- From the above grades points were also subtracted for:
  - -1 for no runtime analysis
  - -2 for no correctness proof
  - -1 if you forgot to add an edge from v to u and your reduction required one.

(b) [10 points] As we discussed in class, finding a longest simple path is NP-Hard. Therefore, there is no known algorithm that, on input u, v, and G returns a longest simple path from u to v in polynomial time. Using this fact (which you do not need to prove) and Part (a), show that there is no known polynomial time algorithm that can find a longest simple cycle in a graph.

Note: If you were unable to solve Part (a), you may assume an algorithm SIMPLEPATHFROMCYCLE for finding a longest simple path from u to v that runs in time polynomial in L, $|V|$, and $|E|$ where L is the running time of a black-box algorithm for solving LONGESTSIMPLECYCLE.

Solution: If L is polynomial than the algorithm outlined in Part (a) gives a polynomial time algorithm for finding the longest simple path in a graph. Contradiction.

Grading:

- 10/10 For anything like above proof. It was also fine to say that the reduction means LSC must be at least as hard as LSP since we stressed that formulation in class.
- 9/10 If you said you can use LSC to solve LSP and therefore by definition of NP-Hard, LSC is also NP-Hard. The definition of NP-Hard is that all NP optimization problems can be reduced to some problem in NP-Hard. So you need an extra step just tying all the ends together, but it's very close.

Problem 4. Closest pair [28 points] (2 parts)

We are interested in finding the closest pair of points in the plane, closest in the sense of the rectilinear distance (also called the Manhattan or L_1 distance). The rectilinear distance between two points p_1 and p_2 on the plane is defined as $d(p_1, p_2) = |x_1 - x_2| + |y_1 - y_2|$, where the x's and y's are the first and second coordinates, respectively. In the first part we consider a one-dimensional version of the problem as a warm-up. In both cases, coordinates of the points are real numbers with k significant digits beyond the decimal point, k constant.

(a) [8 points] Warm up - Provide an efficient way to find a closest pair among n points in the interval $[0, 1]$ on the line. Full credit will be given to the most efficient algorithm with a correct analysis.

Solution: Use radix sort on the n numbers (k digits representation). Then go through the sorted list and calculate the distance between each point and the next one, keeping a running minimum along the way. Radix sort will take $O(kn)$, and distance calculations $n - 1$ time $O(k)$, so an overall linear time $O(n)$, since k is fixed. (Half points only given to an $O(n \log n)$ answer with otherwise correct analysis.)

(b) [20 points] Case of the plane - A divide and conquer approach for n points in the square $[0, 1] \times [0, 1]$. Here is a possible strategy. Divide the set of points into two sets of about half size: those on the right of the x-coordinate median, and those on the left. Recursively, find the closest pair of points on the right, and the closest pair of points on the left and let δ_r and δ_l be the corresponding distances. The overall closest pair is either the minimum of these two options, or corresponds to a pair where one point is on the right of the median and the other is on the left. Given δ_r and δ_l, there should be an efficient way to find the latter. Explain how; then write the full recurrence for the running time of this approach; and conclude with its overall running time.

Solution: Searching for closest pair of points that cross the median x-coordinate can be limited to the vertical strip around the median line of width +/- δ where $\delta = \min\{\delta_r, \delta_l\}$. By having the points belonging to this strip sorted according to their y-coordinates, one can go through of all them bottom up, and for each check distances to points above them within a vertical bound of δ (one can show that only the 7 points above need to be checked for each point). In the worst case, all n points are in this strip, so we end up with a recurrence for the running time of $T(n) = 2T(n/2) + O(n)$ and so $T(n) = O(n \log n)$.

Problem 5. APSP Algorithm for Sparse Graphs [30 points] (3 parts)

Let $G = (V, E)$ be a weighted, directed graph that can have some of the weights negative. Let $n = |V|$ and $m = |E|$, and assume that G is strongly connected, i.e., for any vertex u and v there is a path from u to v in G.

We want to solve all-pairs-shortest-path problem (APSP) in G, i.e., we want to either find all the vertex-to-vertex distances $\{\delta(v, u)\}_{v,u \in V}$, or report existence of a negative-length cycle. We will design an algorithm for this task that runs in $O(mn + n^2 \log n)$ time. (Note that when G is not dense, i.e., when m is $o(n^2)$, the running time of this algorithm is asymptotically better than the one of the Floyd-Warshall algorithm.)

(a) [5 points] Fix some vertex $t \in V$ and consider the vertex potential $\lambda_t(u) = \delta(u, t)$ where $\delta(u, t)$ is the shortest path from $u \in V$ to t. Give an algorithm for calculating $\lambda_t(u)$ for all $u \in V$ and analyze the running time.

Solution: Bellman-Ford in $O(|V||E|)$ time.

(b) [5 points] Show $\lambda_t(u)$, the potential from Part (a), is a feasible potential even if some of the original weights are negative.

Proof of the feasibility of λ:

Consider $w^*(u, v)$ for any $u, v \in V$. By definition

$$w^*(u, v) := w(u, v) - \lambda(u) + \lambda(v) = w(u, v) - \delta(u, t) + \delta(v, t).$$

Since $\delta(u, v) \leq w(u, v)$, we have that

$$w^*(u, v) = w(u, v) - \delta(u, t) + \delta(v, t) \geq \delta(u, v) - \delta(u, t) + \delta(v, t).$$

But, by triangle inequality we have that $\delta(u, t) \leq \delta(u, v) + \delta(v, t)$, thus

$$w^*(u, v) \geq \delta(u, v) - \delta(u, t) + \delta(v, t) \geq 0,$$

as desired.

(c) [20 points] Show how you use the vertex potential from Parts (a) and (b) to solve APSP in G in $O(mn + n^2 \log n)$ time, including the time it takes to calculate the vertex potential.

Solution: Choose any vertex t in G and run Bellman-Ford algorithm from it on the transposed graph G to either compute all the distances $\delta(u, t)$ for every $u \in V$, or detect a negative-length cycle. If such a cycle is detected in the transposed graph then it also exists in the original graph, so just report its existence and terminate.

Otherwise, define the vertex potential $\lambda(u) := \delta(u, t)$ and look at the resulting reduced weight $w^*(u, v) := w(u, v) - \lambda(u) + \lambda(v)$ in G. Since G is strongly connected (and we now know it does not have negative-length cycles), all distances $\delta(u, t)$ are finite and thus all $\lambda(u)$'s are finite as well. From the recitations (and pset 5B) we know that this implies that λ is a feasible potential, i.e., for every $u, v \in V$ $w^*(u, v)$ is non-negative (for the sake of completeness we include the proof of feasibility below).

Since the reduced weight w^* is non-negative, one can run Dijkstra's algorithm in G from every node $u \in V$ to compute all the vertex-to-vertex distances $\{\delta^*(u, v)\}_{u,v \in V}$ with respect to w^*. Next, extract the true vertex-to-vertex distances in G by computing $\delta(u, v) = \delta^*(u, v) - \lambda(v) + \lambda(u)$ for all $u, v \in V$, and output them.

The running time of the above algorithm is dominated by one execution of Bellman-Ford algorithm – which is $O(mn)$ time – and n executions of Dijkstra's algorithm – which is $O(n(m + n \log n)) = O(mn + n^2 \log n)$ time using the implementation of Dijkstra with Fibonacci heaps. Thus the total running time is $O(mn + n^2 \log n)$, as desired.

Problem 6. Airplane Scheduling [30 points] (3 parts)

Consider the runway reservation system from PS2. Recall that we kept track of requested landing times from airplanes by storing them in a balanced binary search tree. In that problem set, we required, for safety, that no landing time be within three minutes of any other landing time in the tree. We showed we could insert into the tree, delete from the tree, and check the validity of a landing time in $O(\log n)$ time.

Sometimes severe weather hits, and the 3-minute window between flights just isn't safe, so a new window size is determined. In these cases, an extra runway might be opened up at a nearby airport to take flights which don't fit within the new window. For all parts, you may use data structure augmentations provided that you explain the augmentation. Its maintenance may not increase the asymptotic running time of other operations, but you are not required to prove this.

(a) [5 points] Provide a very fast (constant-time) algorithm to determine if there are any flights which are not valid with the new window so that the extra runway can start opening immediately.

missing sol!

(b) [15 points] Also provide a slower algorithm which locates which specific flights are not valid within the new window so they can be rescheduled. For example, assuming a window of 3 minutes with flights at times 28, 31, 35, 40, 43, 48, 53, 57, 60, if the window size were expanded to 4 minutes, the algorithm should return that 28, 31, 40, 43, 57, 60 are invalid.

Solution: Like question 2d on the problem set, but instead of tracking for the presence of a particular window size, track the minimum window of the subtree. Constant-time lookup at the root to see if any window is smaller than the new window value. Traversing the tree is both $O(n)$ and $O(k \log n)$ to find all newly invalid times. With bounds on the window size, we could keep a side-structure with all flights belonging to a given window.

(c) [10 points] Can the running time of this slower algorithm be improved if we assume bounds on the maximum size the window could become?

no sol here either

Problem 7. Traveling on a Budget [30 points] (2 parts)

Arthur Dent has \$500 and 1000 hours to go from Cambridge, MA, to Berkeley, CA. He has a map of the States represented as a directed graph $G = (V, E)$. The vertices of the graph represent towns, and there is a directed edge $e = (A, B)$ from town A to town B if there is some means of public transportation connecting the two towns. Moreover, the edge is labeled with a pair (m_e, t_e), representing the cost $m_e \in \{0, 1, \ldots\}$ in dollars of transportation from A to B and the time $t_e \in \{0, 1, \ldots\}$ in hours that it takes to go from A to B.

Arthur is interested in finding a path from Cambridge to Berkeley that does not cost more than \$500 and does not take more than 1000 hours, while also minimizing the objective $5M^2 + 2T^2$, where M is the cost of the trip in dollars and T is the duration of the trip in hours. He was looking for an algorithm that runs in time polynomial in $|V|$ and $|E|$ ...

(a) [10 points] Due to his lack of knowledge in algorithms, he gave up on the idea of respecting the budget and time constraints. At least he thought he could efficiently find the path minimizing the objective $5M^2 + 2T^2$. He tried modifying Dijkstra's algorithm as follows: If an edge e was labeled (m_e, t_e), he assigned it a weight $w_e = 5m_e^2 + 2t_e^2$, and ran Dijkstra's algorithm on the resulting weighted directed graph. Show that Arthur's algorithm may return incorrect results, i.e. return a path that does not minimize the objective $5M^2 + 2T^2$.

(b) [20 points] Now provide an algorithm that solves Arthur's original problem in time polynomial in $|E|$ and $|V|$. Your algorithm should find the path that minimizes the objective $5M^2 + 2T^2$, while at the same time respecting the constraints $M \leq 500$ and $T \leq 1000$. Please describe your algorithm precisely, and justify its correctness and running time. More credit will be given to faster algorithms, provided that the analysis of the algorithm is correct.

[*Hint 1: Use dynamic programming.*]

[*Hint 2: For each town A, integer values* $m \leq 500$ *and* $t \leq 1000$*, either there is a path from Cambridge to A that requires cost* m *and time* t*, or there is not.*]

Solution sketch: We construct a 501-by-15 sized matrix M_v at each vertex v so that the (i, j)'th entry of M_v is 1 if there's a path from the node for Cambridge to v of total cost i and total duration j and is 0 otherwise. $M_{\text{Cambridge}}$ is set to be 1 at entry $(0, 0)$ and 0 everywhere else.

Final Exam

- Do not open this quiz booklet until directed to do so. Read all the instructions on this page.
- When the quiz begins, write your name on every page of this quiz booklet.
- You have 180 minutes to earn **200** points. Do not spend too much time on any one problem. Read them all through first, and attack them in the order that allows you to make the most progress.
- This quiz booklet contains 12 pages, including this one. Two extra sheets of scratch paper are attached. Please detach them before turning in your quiz at the end of the exam period.
- This quiz is closed book. You may use **three** $8\frac{1}{2}'' \times 11''$ or A4 crib sheets (both sides). No calculators or programmable devices are permitted. No cell phones or other communications devices are permitted.
- Write your solutions in the space provided. If you need more space, write on the back of the sheet containing the problem. Pages may be separated for grading.
- Do not waste time and paper rederiving facts that we have studied. It is sufficient to cite known results.
- Show your work, as partial credit will be given. You will be graded not only on the correctness of your answer, but also on the clarity with which you express it. Be neat.
- Good luck!

Problem	Parts	Points	Grade	Grader		Problem	Parts	Points	Grade	Grader
1	8	24				5	1	24		
2	4	32				6	1	24		
3	3	24				7	1	24		
4	3	24				8	1	24		
						Total		200		

Name: _____

Problem 1. True or False [24 points] (8 parts)

For each of the following questions, circle either T (True) or F (False). **Explain your choice.** (No credit if no explanation given.)

(a) **T F** There exists an algorithm to build a binary search tree from an unsorted list in $O(n)$ time.
Explain:

(b) **T F** There exists an algorithm to build a binary heap from an unsorted list in $O(n)$ time.
Explain:

(c) **T F** To solve the SSSP problem for a graph with no negative-weight edges, it is necessary that some edge be relaxed at least twice.
Explain:

(d) **T F** On a connected, directed graph with only positive edge weights, Bellman-Ford runs asymptotically as fast as Dijkstra.
Explain:

(e) **T** **F** A Givens rotation requires $O(1)$ time.
Explain:

(f) **T** **F** In the worst case, merge sort runs in $O(n^2)$ time.
Explain:

(g) **T** **F** There exists a stable implementation of merge sort.
Explain:

(h) **T** **F** An AVL tree T contains n integers, all distinct. For a given integer k, there exists a $\Theta(\lg n)$ algorithm to find the element x in T such that $|k - x|$ is minimized.
Explain:

Problem 2. Short Answer [32 points] (4 parts)

(a) The *eccentricity* $\epsilon(u)$ of a vertex u in a connected, undirected, *unweighted* graph G is the maximum distance from u to any other vertex in the graph. That is, if $\delta(u, v)$ is the shortest path from u to v, then $\epsilon(u) = \max_{v \in V} \delta(u, v)$.

Give an efficient algorithm to find the eccentricity of a given vertex s. Analyze its running time.

(b) What is the asymptotic cost of solving a linear system of equations with $n-1$ equations of the form

$$a_{i,i}x_i + a_{i,i+1}x_{i+1} = b_i \quad i = 1, \ldots, n-1$$

and one equation of the form

$$a_{n,1}x_1 + a_{n,2}x_2 + \cdots + a_{n,n}x_n = b_n$$

(none of the a's in this equation are zero). The a's and b's are given numbers and the x's are unknowns. Assume that we use Givens rotations to reduce the coefficient matrix to a triangular form. Justify your answer.

(c) Suppose a hash function h maps arbitrary keys to values between 0 and $m-1$, inclusive. We hash n keys, $k_1, k_2, \ldots, k_n$. If $h(k_i) = h(k_j)$, we say that k_i and k_j collide. Assuming simple uniform hashing, how should we choose m (in terms of n) such that the expected number of collisions is $O(1)$? Justify your answer.

(d) Suppose you have a directed acyclic graph with n vertices and $O(n)$ edges, all having nonnegative weights. Propose an efficient method of finding the shortest path to each vertex from a single source, and give its running time in terms of n.

Problem 3. Judge Jill [24 points] (3 parts)

Judge Jill has created a web site that allows people to file complaints about one another. Each complaint contains exactly two names: that of the person who filed it and that of the person he/she is complaining about.

Jill had hoped to resolve each complaint personally, but the site has received so many complaints that she has realized she wants an automated approach.

She decides to try to label each person as either good or evil. She only needs the labeling to be consistent, not necessarily correct. A labeling is consistent if every complaint labels one person as good and the other person as evil, and no person gets labeled both as good and evil in different complaints.

(a) [8 points] Propose a way to model the consistent labeling problem as a graph problem.

(b) [10 points] Propose an efficient algorithm to consistently label all the names as good or evil, or to decide that no such classification exists. Use the graph model you proposed in the previous part of the problem. Analyze the running time of the algorithm.

(c) [6 points] Later, Judge Jill wants to be more thorough. She will interview some people to figure out who is good and who is evil. She can always determine whether a person is good or evil by interviewing him or her. Assuming that one person in every complaint is good and the other is evil, what is the minimum number of people she needs to interview to correctly classify all the people named in the complaints?

Problem 4. Bitdiddle Bins [24 points] (3 parts)

Ben Bitdiddle has devised a new data structure called a Bitdiddle Bin. Much like an array or a set, you can INSERT values into it, and you can LOOKUP values to see if they are contained in the structure. (He'll figure out DELETE later.)

A Bitdiddle Bin is implemented as a pair of lists (arrays), designated the *neat list* and the *messy list*, with these properties:

- The *neat list* is always in sorted order. (The messy list may or may not be sorted.)
- The *messy list* has a size of at most $\sqrt{n}$, where n is the total number of values in the entire Bitdiddle Bin.

The LOOKUP algorithm for a Bitdiddle Bin is as follows:

1. Use binary search to look for the value in the neat list.
2. If it wasn't in the neat list, iterate over the entire messy list to look for the value.

The INSERT algorithm is as follows:

1. Append the value to the messy list.
2. If the messy list is now too big, CLEANUP.

This CLEANUP subroutine is run whenever the messy list grows beyond $\sqrt{n}$ items:

1. Sort the messy list.
2. Merge the messy list with the neat list.
3. The merge result is the new neat list. The new messy list is empty.

(a) [4 points] What is the worst-case asymptotic runtime of LOOKUP on a Bitdiddle Bin?

(b) [8 points] What is the worst-case asymptotic runtime of INSERT on a Bitdiddle Bin? Explain.

(c) [12 points] What is the *amortized* asymptotic runtime of each INSERT operation, when inserting n values into an empty Bitdiddle Bin? Explain.

Problem 5. Local Minimum [24 points]

Consider an array A containing n distinct integers. We define a *local minimum* of A to be an x such that $x = A[i]$, for some $0 \leq i < n$, with $A[i-1] > A[i]$ and $A[i] < A[i+1]$. In other words, a local minimum x is less than its neighbors in A (for boundary elements, there is only one neighbor). Note that A might have multiple local minima.

As an example, suppose $A = [10, 6, 4, 3, 12, 19, 18]$. Then A has two local minima: 3 and 18.

Of course, the absolute minimum of A is always a local minimum, but it requires $\Omega(n)$ time to compute.

Propose an efficient algorithm to find some local minimum of A, and analyze the running time of your algorithm.

Problem 6. Straits [24 points]

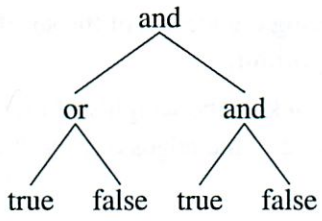
Let G be a connected, weighted, undirected graph. All the edge weights are positive. An edge e is a *strait* between vertices u and v if it has weight ℓ, is on a path from u to v whose other edges weigh ℓ or more, and every path between u and v contains an edge of weight ℓ or less. In other words, to go from u to v you must cross an edge of weight ℓ, but you do not need to cross edges lighter than ℓ.

Describe an efficient algorithm for finding the weight of the strait between every pair of vertices in G. Analyze the running time of the algorithm.

Hint: Use a $|V|$-by-$|V|$ table to keep track of the weights of all the straits. Initialize it so that it is correct for a graph with no edges. Then add the edges one by one.

Problem 7. The Cake Is a Lie! [24 points]

At Aperture Bakeries, every cake comes with a binary boolean-valued tree indicating whether or not it is available. Each leaf in the tree has either a *true* or a *false* value. Each of the remaining nodes has exactly two children and is labeled either *and* or *or*; the value is the result of recursively applying the operator to the values of the children. One example is the following tree:



If the root of a tree evaluates to *false*, like the one above, the cake is a lie and you cannot have it. Any *true* cake is free for the taking. You may modify a tree to make it *true*; the only thing you can do to change a tree is to turn a *false* leaf into a *true* leaf, or vice versa. This costs \$1 for each leaf you change. You can't alter the operators or the structure of the tree.

Cake is good. Cheap cake is even better. Describe an efficient algorithm to determine the minimum cost of a cake whose tree has n nodes, and analyze its running time.

Problem 8. I Am Locutus of Borg, You Will Respond To My Questions [24 points]

Upon arrival at the planet Vertex T, you and Ensign Treaps are captured by the Borg. They promptly throw the ensign out of the airlock. You had better solve their problem, lest you share his fate.

The Vertex T parking lot is an $n \times n$ matrix A. There are already a number of spaceships parked in the lot. For $0 \leq i, j < n$, let $A[i][j] = 0$ if there is a ship occupying position (i, j), and 1 otherwise.

The Borg want to find the ***largest square parking space*** in which to park the Borg Cube. That is, find the largest k such that there exists a $k \times k$ square in A containing all ones and no zeros. In the example figure, the solution is 3, as illustrated by the 3×3 highlighted box.

	0	1	2	3	4
0	0	1	1	1	0
1	1	1	1	1	1
2	1	1	1	1	1
3	1	1	0	0	0
4	1	1	1	0	1

Describe an efficient algorithm that finds the size of the largest square parking space in A. Analyze the running time of your algorithm.

Hint: Call $A[0][0]$ be the *top-left* of the parking lot, and call $A[n-1][n-1]$ the *bottom-right*. Use dynamic programming, with the subproblem $S[i, j]$ being the side length of the largest square parking space whose bottom-right corner is at (i, j).

SCRATCH PAPER

SCRATCH PAPER

Introduction to Algorithms
Massachusetts Institute of Technology
Professors Sivan Toledo and Alan Edelman

May 21, 2009
6.006 Spring 2009
Final Exam Solutions

Final Exam Solutions

Problem 1. True or False [24 points] (8 parts)

For each of the following questions, circle either T (True) or F (False). **Explain your choice.** (No credit if no explanation given.)

(a) T F There exists an algorithm to build a binary search tree from an unsorted list in $O(n)$ time.
Explain:

Solution: False. Because an in-order walk can create a sorted list from a BST in $O(n)$ time, the existence of such an algorithm would violate the $\Omega(n \lg n)$ lower bound on comparison-based sorts.

(b) T F There exists an algorithm to build a binary heap from an unsorted list in $O(n)$ time.
Explain:

Solution: True. The standard BUILD-HEAP algorithm runs in $O(n)$ time.

(c) T F To solve the SSSP problem for a graph with no negative-weight edges, it is necessary that some edge be relaxed at least twice.
Explain:

Solution: False. Dijkstra's algorithm solves the SSSP problem relaxing each edge at most once.

(d) T F On a connected, directed graph with only positive edge weights, Bellman-Ford runs asymptotically as fast as Dijkstra.
Explain:

Solution: False. Bellman-Ford requires $\Theta(VE)$, regardless of the edge weights. Dijkstra runs in $\Theta(E + V \lg V)$. Because the graph is connected, $E = \Omega(V)$, so $\Theta(VE) = \Omega(V^2)$, which is clearly worse than Dijkstra.

(e) T F A Givens rotation requires $O(1)$ time.
Explain:

Solution: False. It may take $O(1)$ time if the matrix is sparse, but in general a Givens rotation requires $O(n)$ for a matrix with n columns.

(f) T F In the worst case, merge sort runs in $O(n^2)$ time.
Explain:

Solution: True. Merge sort runs in $O(n \lg n)$ time which is $O(n^2)$.

(g) T F There exists a stable implementation of merge sort.
Explain:

Solution: True. If the items in the two lists being merged are equal, choose the item in the "left" half (the half that came first in the original array).

(h) T F An AVL tree T contains n integers, all distinct. For a given integer k, there exists a $\Theta(\lg n)$ algorithm to find the element x in T such that $|k - x|$ is minimized.
Explain:

Solution: True. INSERT k, then find the PREDECESSOR and SUCCESSOR of k. Return the one whose difference with k is smaller. All three methods take $\Theta(\lg n)$ time.

Problem 2. Short Answer [32 points] (4 parts)

(a) The *eccentricity* $\epsilon(u)$ of a vertex u in a connected, undirected, *unweighted* graph G is the maximum distance from u to any other vertex in the graph. That is, if $\delta(u, v)$ is the shortest path from u to v, then $\epsilon(u) = \max_{v \in V} \delta(u, v)$.
Give an efficient algorithm to find the eccentricity of a given vertex s. Analyze its running time.

Solution: Run BFS starting at s. Keep track of how many "levels" have been explored. The number of levels required to explore the entire graph is equal to $\epsilon(s)$. This requires $O(V + E)$ time.

(b) What is the asymptotic cost of solving a linear system of equations with $n-1$ equations of the form

$$a_{i,i}x_i + a_{i,i+1}x_{i+1} = b_i \quad i = 1, \ldots, n-1$$

and one equation of the form

$$a_{n,1}x_1 + a_{n,2}x_2 + \cdots + a_{n,n}x_n = b_n$$

(none of the a's in this equation are zero). The a's and b's are given numbers and the x's are unknowns. Assume that we use Givens rotations to reduce the coefficient matrix to a triangular form. Justify your answer.

Solution: If we form a matrix and apply Givens rotations, eliminating the first column will fill in all the elements below the main diagonal, so we will need to perform $\Theta(n^2)$ rotations at a total cost of $\Theta(n^3)$. If we move the last equation (the one without zero coefficients) to the top, we only need to perform one Givens rotation per column, so the total cost reduces to $\Theta(n^2)$.

(c) Suppose a hash function h maps arbitrary keys to values between 0 and $m-1$, inclusive. We hash n keys, $k_1, k_2, \ldots, k_n$. If $h(k_i) = h(k_j)$, we say that k_i and k_j collide. Assuming simple uniform hashing, how should we choose m (in terms of n) such that the expected number of collisions is $O(1)$? Justify your answer.

Solution: Any two keys collide with probability $1/m$. There are $\binom{n}{2} = O(n^2)$ pairs of elements, so the expected number of collisions is $O(n^2/m)$. Thus, let $m = \Omega(n^2)$.

(d) Suppose you have a directed acyclic graph with n vertices and $O(n)$ edges, all having nonnegative weights. Propose an efficient method of finding the shortest path to each vertex from a single source, and give its running time in terms of n.

Solution: The given conditions allow us to use either the DAG shortest paths algorithm or Dijkstra's. DAG shortest paths runs in $\Theta(V + E) = \Theta(n)$, and Dijkstra's runs in $O((V + E) \lg V)$, or $O(V \lg V + E)$ if using a Fibonacci heap. In either case, Dijkstra's runs in $O(n \lg n)$. DAG shortest paths is faster in this case.

Problem 3. Judge Jill [24 points] (3 parts)

Judge Jill has created a web site that allows people to file complaints about one another. Each complaint contains exactly two names: that of the person who filed it and that of the person he/she is complaining about.

Jill had hoped to resolve each complaint personally, but the site has received so many complaints that she has realized she wants an automated approach.

She decides to try to label each person as either good or evil. She only needs the labeling to be consistent, not necessarily correct. A labeling is consistent if every complaint labels one person as good and the other person as evil, and no person gets labeled both as good and evil in different complaints.

(a) [8 points] Propose a way to model the consistent labeling problem as a graph problem.

Solution: We can model the problem using an undirected graph in which every edge represents one complaint and every vertex represents one name (one person).

(b) [10 points] Propose an efficient algorithm to consistently label all the names as good or evil, or to decide that no such classification exists. Use the graph model you proposed in the previous part of the problem. Analyze the running time of the algorithm.

Solution: The problem is quivalent to testing whether we can color the vertices of graph with two colors, or equivalently, whether the graph is bipartite. This can be done using DFS in $Theta(V+E)$ time. We label the root of DFS traversals arbitrarily good or bad. When we first visit a vertex, we label it differently than its parent. When we encounter back or cross edges, we check for consistency; if we find an inconsistency, the graph cannot be labeled consistently.

(c) [6 points] Later, Judge Jill wants to be more thorough. She will interview some people to figure out who is good and who is evil. She can always determine whether a person is good or evil by interviewing him or her. Assuming that one person in every complaint is good and the other is evil, what is the minimum number of people she needs to interview to correctly classify all the people named in the complaints?

Solution: She needs to interview one person in each connected component of the complaint graph.

Problem 4. Bitdiddle Bins [24 points] (3 parts)

Ben Bitdiddle has devised a new data structure called a Bitdiddle Bin. Much like an array or a set, you can INSERT values into it, and you can LOOKUP values to see if they are contained in the structure. (He'll figure out DELETE later.)

A Bitdiddle Bin is implemented as a pair of lists (arrays), designated the *neat list* and the *messy list*, with these properties:

- The *neat list* is always in sorted order. (The messy list may or may not be sorted.)
- The *messy list* has a size of at most $\sqrt{n}$, where n is the total number of values in the entire Bitdiddle Bin.

The LOOKUP algorithm for a Bitdiddle Bin is as follows:

1. Use binary search to look for the value in the neat list.
2. If it wasn't in the neat list, iterate over the entire messy list to look for the value.

The INSERT algorithm is as follows:

1. Append the value to the messy list.
2. If the messy list is now too big, CLEANUP.

This CLEANUP subroutine is run whenever the messy list grows beyond $\sqrt{n}$ items:

1. Sort the messy list.
2. Merge the messy list with the neat list.
3. The merge result is the new neat list. The new messy list is empty.

(a) [4 points] What is the worst-case asymptotic runtime of LOOKUP on a Bitdiddle Bin?

Solution: Binary search on the neat list takes $\Theta(\log n)$, and iterating over the messy list takes $\Theta(\sqrt{n})$, for a total time of $\Theta(\sqrt{n})$.

(b) [8 points] What is the worst-case asymptotic runtime of INSERT on a Bitdiddle Bin? Explain.

Solution: Appending is $\Theta(1)$. Sorting the messy list is $\Theta(\sqrt{n}\log\sqrt{n})$. Merging is $\Theta(n)$. The $\Theta(n)$ term dominates, so the asymptotic runtime is $\Theta(n)$.

(c) [12 points] What is the *amortized* asymptotic runtime of each INSERT operation, when inserting n values into an empty Bitdiddle Bin? Explain.

Solution: Use the aggregate method. The cost is $\Theta(1)$ to append to the messy list, plus $\Theta(n)$ per CLEANUP. Because CLEANUP is called $\Theta(\sqrt{n})$ times, the total cost for the n insertions is $O(n\sqrt{n})$, so the amortized cost per operation is $O(n\sqrt{n}/n) = O(\sqrt{n})$.

Problem 5. Local Minimum [24 points]

Consider an array A containing n distinct integers. We define a *local minimum* of A to be an x such that $x = A[i]$, for some $0 \leq i < n$, with $A[i-1] > A[i]$ and $A[i] < A[i+1]$. In other words, a local minimum x is less than its neighbors in A (for boundary elements, there is only one neighbor). Note that A might have multiple local minima.

As an example, suppose $A = [10, 6, 4, 3, 12, 19, 18]$. Then A has two local minima: 3 and 18.

Of course, the absolute minimum of A is always a local minimum, but it requires $\Omega(n)$ time to compute.

Propose an efficient algorithm to find some local minimum of A, and analyze the running time of your algorithm.

Solution: Use a divide-and-conquer algorithm. Let $m = n/2$, and examine the value $A[m]$ (that is, the element in the middle of the array).

Case 1: $A[m-1] < A[m]$. Then the left half of the array must contain a local minimum, so recurse on the left half. We can show this by contradiction: assume that $A[i]$ is not a local minimum for each $0 \leq i < m$. Then $A[m-1]$ is not a local minimum, which implies that $A[m-2] < A[m-1]$. Similarly, $A[m-3] < A[m-2]$. Continuing in this fashion, we obtain $A[0] < A[1]$. But then $A[0]$ is a local minimum, contrary to our initial assumption.

Case 2: $A[m+1] > A[m]$. Then the right half of the array must contain a local minimum, so recurse on the right half. This is symmetrical to Case 1.

Case 3: $A[m-1] > A[m]$ and $A[m+1] < A[m]$. Then $A[m]$ is a local minimum, so return it.

The running time recurrence is $T(n) = T(n/2) + \Theta(1)$, which yields $T(n) = \Theta(\log n)$.

Problem 6. Straits [24 points]

Let G be a connected, weighted, undirected graph. All the edge weights are positive. An edge e is a *strait* between vertices u and v if it has weight ℓ, is on a path from u to v whose other edges weigh ℓ or more, and every path between u and v contains an edge of weight ℓ or less. In other words, to go from u to v you must cross an edge of weight ℓ, but you do not need to cross edges lighter than ℓ.

Describe an efficient algorithm for finding the weight of the strait between every pair of vertices in G. Analyze the running time of the algorithm.

Hint: Use a $|V|$-by-$|V|$ table to keep track of the weights of all the straits. Initialize it so that it is correct for a graph with no edges. Then add the edges one by one.

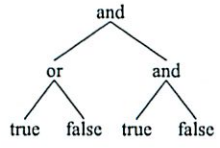
Solution: The initial table has infinity on the diagonal and zero everywhere else. Suppose the table T is correct for $E \setminus \{u, v\}$. We define the table T' for E using the rule

$$T'[x, y] = \max\left[\, T(x, y),\ \min\left(T(x, u), w(u, v), T(v, y)\right),\ \min\left(T(x, v), w(v, u), T(u, y)\right) \,\right].$$

The cost of this update is constant. We perform $V(V-1)$ such updates for every edge, so the total cost is $\Theta(EV^2)$.

Problem 7. The Cake Is a Lie! [24 points]

At Aperture Bakeries, every cake comes with a binary boolean-valued tree indicating whether or not it is available. Each leaf in the tree has either a *true* or a *false* value. Each of the remaining nodes has exactly two children and is labeled either *and* or *or*; the value is the result of recursively applying the operator to the values of the children. One example is the following tree:



If the root of a tree evaluates to *false*, like the one above, the cake is a lie and you cannot have it. Any *true* cake is free for the taking. You may modify a tree to make it *true*; the only thing you can do to change a tree is to turn a *false* leaf into a *true* leaf, or vice versa. This costs \$1 for each leaf you change. You can't alter the operators or the structure of the tree.

Cake is good. Cheap cake is even better. Describe an efficient algorithm to determine the minimum cost of a cake whose tree has n nodes, and analyze its running time.

Solution: One can compute the truth value of each node in $O(n)$ time by starting with the leaves and going up. The precise order can be computed in $O(n)$ time using either a topological sort or BFS. If the root is *true*, the cake is free, so return \$0.

Otherwise, we can use dynamic programming to determine this cost. Augment each node with a field C, the minimum cost to make the node *true*. Each node corresponds to the subproblem of determining C at that node.

The base case consists of the leaves. For each $true$ leaf, set C to \$0. For each $false$ leaf, set C to \$1.

Recursive cases:

For all *and* nodes: Set C to the sum of the children's C values.

For all *or* nodes: Set C to the smaller of the children's C values.

After computing C for all nodes, in the same order in which the truth values were computed initially, return the C value of the root.

Since there are $O(n)$ subproblems, and the cost of each subproblem is constant, the total running time is $O(n)$.

Problem 8. I Am Locutus of Borg, You Will Respond To My Questions [24 points]

Upon arrival at the planet Vertex T, you and Ensign Treaps are captured by the Borg. They promptly throw the ensign out of the airlock. You had better solve their problem, lest you share his fate.

The Vertex T parking lot is an $n \times n$ matrix A. There are already a number of spaceships parked in the lot. For $0 \le i, j < n$, let $A[i][j] = 0$ if there is a ship occupying position (i, j), and 1 otherwise.

	0	1	2	3	4
0	0	1	1	1	0
1	1	1	1	1	1
2	1	1	1	1	1
3	1	1	0	0	0
4	1	1	1	0	1

The Borg want to find the ***largest square parking space*** in which to park the Borg Cube. That is, find the largest k such that there exists a $k \times k$ square in A containing all ones and no zeros. In the example figure, the solution is 3, as illustrated by the 3×3 highlighted box.

Describe an efficient algorithm that finds the size of the largest square parking space in A. Analyze the running time of your algorithm.

Hint: Call $A[0][0]$ be the *top-left* of the parking lot, and call $A[n-1][n-1]$ the *bottom-right*. Use dynamic programming, with the subproblem $S[i, j]$ being the side length of the largest square parking space whose bottom-right corner is at (i, j).

Solution: The base cases are the positions along the top and left sides of the parking lot. In each of these positions, you can fit a 1×1 square parking space if and only if the space is unoccupied. Thus, for the base cases ($i = 0$ or $j = 0$), we have $S[i, j] = A[i][j]$.

Now for the general case. Again, if $A[i][j] = 0$, then $S[i, j] = 0$, so we'll only consider the case where $A[i][j] = 1$. There is a parking space of size x with bottom-right corner at (i, j) if and only if there are three (overlapping) spaces of size $x - 1$ at each of the locations $(i-1, j)$, $(i, j-1)$, and $(i-1, j-1)$. Thus, the largest possible parking space is

$$S[i, j] = 1 + \min(\, S[i-1, j]\,,\ S[i, j-1]\,,\ S[i-1, j-1]\,)$$

The desired answer is $\max_{i,j} S[i, j]$, which requires $O(n^2)$ to compute.

There are n^2 subproblems, and each takes $O(1)$ time to solve, for a time of $O(n^2)$. This, added to the $O(n^2)$ to extract the desired answer, gives a total $O(n^2)$ running time, which is clearly optimal because all the data must be examined.

Exam Thoughts

5/23

not as bad as I thought

time goes fast once exams start

I think I did well on this
Can I make a B?

Break after every problem

Don't put on back too fast!

You just need concent.

Thought more deeply about the problems than any time in semester

---

I feel really confident about - even though prob got any H wrong
But did really deliberate

Afterwards

I don't know why I felt so confident
or feel

felt like I knew the stuff

but I was not able to figure 2 of 3 of
the long problems
and 1 short

but felt like I got everything else

Perhaps I just liked my focus + concentration

#5

Topo ordering
k edges inserted in
May add cycles
Arbitrary weights

"So what matters about the previous topo ordering"

$O(k \cdot (V + E + k))$ algo for shortest path

dij is $O((V+E)\log V)$ "don't fully remember

So have topo best in V+E
So check every path through each of the k possibilities if shorter
└ I'm going off the O() - thats usually right.

This seems likely

"Visit each
- can see if in path
Find shortest path from k
- is better b/c cycles
(Not sure - but time looks correct)

---

check indiv
└ no if 2 nodes.

What does checking mean?
Hand waving it now...

(5.7)

Checking is exploing out from it in Vy Extrive
in ~~DFS~~ ~~BFS~~ DFS

Oh weights!

D_{ij}

- but what is that?
- the queue is no goal

I have a feeling this fails...

└ Leave it...
No longer think about te 0
Now go to 6b and do that!

#6 V_i and p_i

So Score = $\prod p_i \cdot \sum v_i$ chart only adds score

Each score multiplies prob

1) $p_1 \cdot v_1$

2) $p_1 \cdot p_2 \cdot (v_1 + v_2)$

$p_1(v_1 + v_2) \cdot p_2(v_1 + v_2)$

3. $p_1 \cdot p_2 \cdot p_3 \cdot (v_1 + v_2 + v_3)$

$p_1(v_1 + v_2 + v_3) \cdot p_2(v_1 + v_2 + v_3) \cdot p_3(v_1 + v_2 + v_3)$

Easy problem $V_i = 1$

So $p_1(2) \cdot p_2(2) = 2(p_1 \cdot p_2)$

So we add

$p_1 \cdot 1$

$p_1 \underline{\cdot p_1 \cdot p_2 \cdot p_2}$ or $3(p_1 \cdot p_2 \cdot p_3)$

Want optimal subset

DP - include or not!

(2)

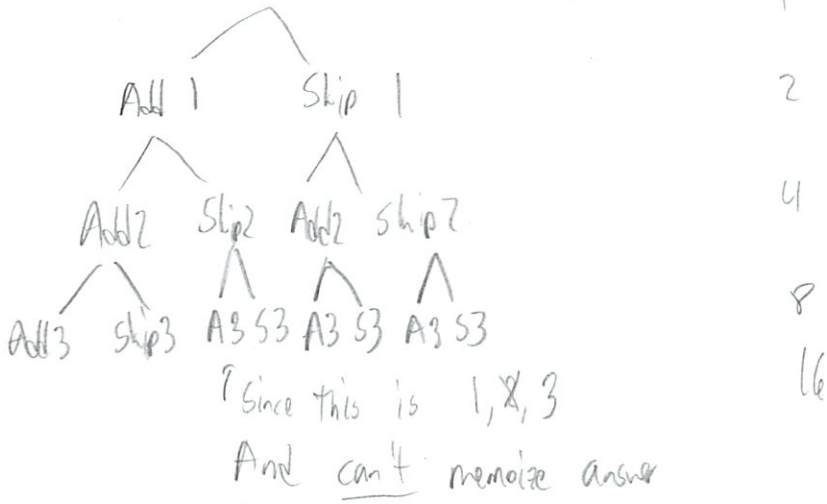
but doesn't need to be contiguous

$DP(i) = \max(p_i \text{ added}, p_i \text{ not added})$

↑ best score

This is O(1) to compute
O(n) to check
w/ memo

No its not just a line



1
2
4
8
16

↑ since this is 1, 2, 3

And <u>can't</u> memoize answer

Recursion $2T(n) + O(1)$

	2^n	n^2
	↑	↑
1	2	1
2	4	4
3	8	9
4	16	16
5	32	25

↑ its exponential

Only choose best col from each

↳ greedy
but thats not best here

DFS ↳ not nlgn

(3) I guess the key is - is one always better?

I think so

Either the EV of the question is good or its not

Plus use the fact its 1 here

keep p sep

just multiply on p

and know whats included, multiply score by that

but still branching exponential

how to limit branching

by revising

Can we say 1 branch is better than the others?

Yeah I think that is the advantage - pick the highest so far at each level - continue that

log n is usually binary search

Or! order the ps from lowest to highest!

Then include or not

I like this

(often I think of the past hw sol lol)

Remember explain, analysis, Correctness

(9)

b) Now v_i can be an int b/w 1 and C

- So limited copper value
- So can bullet etc

- Want a $O(n^2)$
- What I had before was exponential
- So C means we can try each value
- EV each individually
  - time won't allow
- What's n^2
  - each problem w/ every other problem
  - that gets you best of 2
- But at each cost level?

At each cost level, compare each problem w/ every other problem. See what is larger and use that?

? Start w/ local best + work down?

But how compare b/w V_i levels

? Start from high points to low points or back to local EV

(5)

I have some ideas - but don't see how they fit in Cn^2

And I can't see how to get something in Cn^2 to work

Wiese -> no credit since not in time

Lperhaps write

At every possible cost level... ?

Or does that come from the calc of it

Which is n

? Start w/ max points

- try each include or not
- O(n) to calc

Go through each pts level

Skip it, out of time