Studying for Final

5/11

Preposition → true or false

- not always easy
- can try ∀
- but unreliable

∀ for all

ℕ all non neg int

(Think I have gotten more experience w/ proofs)

- Knew nothing at start of semester

Predicate - depends on value of variable

axiom - basic assumptions

theorms - important propositions

lemma - prelim proposition

~~coll~~ corollary - prop a few steps after theorm

ZFC - too complex

Modus ponens

$$\frac{P, \quad P \rightarrow Q}{Q}$$

← antecendent

← Conclusion / consequent

②

Anything on top must be true
- any input

Often proofs follow template

If P, then Q is implication

Proof 1. Assume P
2. Show that Q logically follows
3. So ~~th~~ if P, then Q

Theorm 1.5.1 If $0 < x < 2$ then $-x^3 + 4x + 1 > 0$

x, $2-x$, $2+x$ are all non neg

add 1 gives pos

So $-x^3 + 4x + 1 > 0$ □

But how to get to first step?

Factor!

$-x^3 + 4x = x(2-x)(2+x)$

↑(I think my problem has been this 1st step - or I think that is what I think my problem is)

(How to think how to do that!)

(Need to be more confident in general math abilitey)

(3)

(I think I am getting better)

Contrapositive

Not(Q) → Not(P) iff P → Q

So can prove contrapositive

iff → logical equivilant

holds if and only if other statement does

1. Prove P → Q
2. Prove Q → P

Or 1. Prove P equivilant to something, equiv to something, equiv to Q

"means" is equivilant (I just don't think explictly about it)

<u>Proof by Cases</u>

- Say all the cases
- Prove by one

<u>Proof by Contridiction</u> Show that if prop were false then

Some false fact appears true

which is a contriction! So prop must be true

(4)

Proofs Should be short + simple + well ritten

1. State gameplan
2. keep a linear flow
3. Essay not calculation
4. Avoid symbols
5. define notation
6. Stucture long proofs
7. Avoid "obvious"
8. Conclude

---

2. Well Ordering Principle

Every nonempty set of nonneg int has a smallest element
(I always found this weird)
Can always write a fraction in lowest common terms
If not, (if common factor) could divide by that
Set will be nonempty, will always be a lowest item
So fractions can always be written in lowest possible terms

(5)

Can use to prove that some property P(n) holds for every non-neg int n

1. Define a set C of counterexamples to P being true

$C ::= \{n \in N \mid P(n) \text{ is false}\}$

2. Assume for proof that this is non empty
3. By WOP must be smallest element n in C
4. Reach a contridiction (somehow)
   - Often by showing how to use n to find another member of C that is smaller than n
   - this is the open ended part
5. Conclude that C must be empty, no contridictions, QED

Watch out for special cases 1, 0

Don't get tripped up by notation

Their example seems to be more induction

Say c is smallest counter example

So $n < c$

true $n = 0$, so $c > 0$

So $c - 1$ is non neg (can be 0)

⑥ c-1 < c, so can plug in

Then plug in (seems like ∞ series question)

$$= \frac{(c-1)c}{2}$$

Add c

Show that equivalent

So contradiction

---

3 Logical Formulas

Importance of precise language

NOT()

T	F
F	T

AND()

TT	T
TF	F
FT	F
FF	F

OR()

TT	T
TF	T
FT	T
FF	F

XOR()

TT	F
TF	T
FT	T
FF	F

Implies

P Q	P → Q
T T	T
T F	F
F T	T
F F	T

if conclusion is true its always true

either "or" can be true

↑ if the if is false statement is still true

⑦

P	Q	P iff Q
T	T	T
T	F	F
F	T	F
F	F	T

← equivalent

Can rewrite condition multiple ways

Do part at a time w/ the parenthesis

Can often simplify them

∧ = and = ∩

∨ = or = ∪

iff ↔

⊕ XOR

Contrapositive works Not(Q) → Not(P)

Converse does not! Q → P

<u>Valid</u> = always true

<u>Satisfiable</u> = sometimes true

⑧

Every prop formula is equivilant to a "sum of products"
or disjunctive formula

- means only an or of and terms
  ↑ DNF (disjunctive normal form) ↑ only letter or not letter terms

Conjunctive formulas AND of OR terms

Each formula has both

A and B ↔ B and A Commutivity

(A and B) and C ↔ A and (B AND C) associativity

T and A ↔ A identity

F and A ↔ F Zero ? (stupid!)

A and A ↔ A idempotence

A and $\bar{A}$ ↔ F Contridiction

Not($\bar{A}$) ↔ A double negation

A or $\bar{A}$ ↔ T validity

A and (B or C) ↔ (A and B) or (A and C) distributive

(9)

↳ can both be true

NOT (A and B) ⟺ $\bar{A}$ or $\bar{B}$ De Morgans

NOT (A or B) ⟺ $\bar{A}$ and $\bar{B}$ De Morgans

SATisfiable is problem

Telling if can be satisfied

Can do whole true table - look if ever true

But very hard to tell - exponentially

P vs NP - can SAT have a polynomical solution

$\forall$ for all

$\exists$ there exists

always true = <u>universal</u>

Sometimes true = <u>existential</u>

↳ $\exists$ some value $x \in \mathbb{R}$ where $5x^2 - 7 = 0$

↳ this is true

Order of quantifiers matters

Comes from same set if you don't specify

Not($\forall x$ P(x)) iff $\exists x$ Not(P(x))

4 Math Data Types

Sets

bunch of objects / elements

Order does not matter

Can have set of sets

objects can't be more than once

$\mathbb{N}$ = non neg int

$\mathbb{Z}$ = int

$\mathbb{Q}$ = rational numbers

$\mathbb{R}$ = real numbers

$\mathbb{C}$ = complex numbers

$\cup$ union

$\cap$ intersection

$\bar{A}$ complement $\bar{A} = D - A$

↑ domain

$P(A)$ is power set

for $\{1, 2\}$

$\emptyset$ $\{1\}$ $\{2\}$ $\{1, 2\}$

Distributive Law

$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$

- Can prove by truth table
- still have to Convert then the $\cap$ / $\cup$ to AND/ORs
- Use equivalance
- convert both

Seq

Order matters

Product is

$\mathbb{N} \times \{a, b\} = \{\{0, a\} \{0, b\} \{1, a\} \{1, b\}, \dots\}$

Function

Assigns el from 1 set (domain) to el of other (codomain)

$f: A \to B$

$f(a) = b$

$b = f(a)$

f assigns element $b \in B$ to a

Value of f at argument a

(12)

Composition

$(g \circ f)(x) = h(x) = g(f(x))$

(All this is when I really paid attention)

Binary Relations

$$\overset{R}{A \times B} = \text{graph of } R$$

A ↑ domain, B ↑ Codomain

T = in charge of

Can define <u>image</u>

$R(X) ::= \{ b \in B \mid x R b \text{ for some } x \in X \}$

Basically set of endpoints of arrows that start in X

So T(Meyer) is all subjects Meyer is in charge of

T(F) is all subjects being taught

$T^{-1}(N)$ is all people in charge of subject

So $T^{-1}(\text{G, UAT}) = \text{Eng}$

$b R^{-1} a$ iff $a R b$

(13)

(I think this is where I stopped really paying attention)

$T(T^{-1}(D))$ is set of subjects that have instructors that are also in charge of intro subjects

(So how does this help a PC?
- what do you do in a db?
? use logical tables to simplify queries)

---

5 Infinity

<u>function</u> - at most one arrow out
<u>total</u> - at least one arrow out
- so really exactly 1 arrow out

<u>surjective</u> - every has at least 1 arrow in
<u>injective</u> - every has at most 1 arrow in
<u>bijective</u> - every A exactly 1 arrow out
<u>AND</u> every B exactly 1 arrow in

(can never memorize this stuff!)

(14)

Mapping Rule

↓ Size of

A surj B	$\lvert A\rvert \geq \lvert B\rvert$
A inj B	$\lvert A\rvert \leq \lvert B\rvert$
A bij B	$\lvert A\rvert = \lvert B\rvert$
A strict B	$\lvert A\rvert > \lvert B\rvert$

└ A surj B but NOT (B surj A)

Schroder-Bernstein If A surj B AND B surj A Then A bij B

Infinity is different

$\lvert A \cup \{b\}\rvert = \lvert A\rvert + 1$ is not same size as $\lvert A\rvert$

but is same size if A is ∞

A is infinite iff A bij $A \cup \{b\}$

(I finally get this theory!)

Countably ∞ iff N bij C

– are the smallest inf set

Countable if finite or countably ∞

– if elements can be listed in order

Core because can have ∞ possible inputs

<u>Halting Problem</u> - can't perfectly check stuff for all inputs

We can't know if program will halt

Strings recognisible

- but can we recognize it won't halt? No!

(I still don't get proof - but get concept - likely to not be on quiz)

(Read WP)

(After like 45 min I still don't get it!)

Left some pages out - Cantor

Power set [illegible] of ∞ set, bigger than ∞ set with some type of surjection

(I find all this stuff crazy)

6 First-Order Logic

Russell's Paradox

Let s be variables over all sets

$W := \{s \mid s \notin s\}$

So by def

$s \in W$ iff $s \notin s$

for every set s.

Let s be W

$W \in W$ iff $W \notin W$

- But can claim cat assume W is a set
- Just say no set can ever be a member of itself
- W must contain every set
- And W can't be a set - or would be member of itself

Power sets are strictly bigger

So ∞ sets of different sizes

(I think this set stuff is black VooDoo)

(17)

6.b Induction

Students at begining get candy bar

If prev. student gets candy bar, then next does

(But what if only 20 candy bars)

?emailed at

this is what confuses me -

I think I am thinking too much about this)

$P(0)$ is true

$P(n) \rightarrow P(n+1)$ for all non neg int n

So $P(m)$ true for all non neg int m

Invarience

Reachable - start state is reachaple

- if p is a reachable state of M

and $p \rightarrow q$ is a transition of M

then q is also a reachable state of M

(18)

Preserved invariant — if $P(q)$ is true of state q
and $q \to r$ for some state r
then $P(r)$ holds

↓

* If true for start state, then true for all reachable states *

The Die-Hard jug problem

- State machine with the joint value
- Write all the legal changes/transition out of states
- multiple so nondeterministic
- is a state reachable?
  - Since by Invariant Principal every reachable state preserves
  - This is partial correctness
  - But does not prove termination
  - Need WOP

Fast Exponentiation

Set x, y, z to $a, 1, b$
If $z = 0$ return y and terminate
$r = rem(z, 2)$
$z = quot(z, 2)$
if $r = 1$, then $y = xy$
$x = x^2$

19

Has a preserved invarient

$z \in N$ and $yx^z = a^b$

So show all transitions still have this

Termination

$1 \cdot a^b = a^b$ is start

Only stops when $z = 0$, so if $(x, y, 0)$ is reachable

$y = yx^0 = a^b$

(¿ How do we know y does not change)

- y is never storzed ?

Strong Induction

Hold from not just P(n) but also P(0), P(1), etc

Can reformat WOP into

---

2 Recursive Data Types

- Construct new data types from previous ones
- Base case - what you start with
- Constructor

Structural Induction

P is a predicate on recursivly defined data type R

1. P(b) is true for each base element $b \in R$
2. For all 2 argument constructors

$$[P(r) \text{ AND } P(s)] \rightarrow P(c(r,s))$$

for $\forall r,s \in R$

and likewise for all constructors taking a number of other arguments

P(r) is true for all $r \in R$

Example $\#_c(s)$ = # of occurances of "c" is s

Base $\#_c(\lambda) = 0$
↑empty string

Constructor $\#_c(\langle a, s\rangle) = \begin{cases} \#_c(s) & \text{if } a \neq c \\ 1 + \#_c(s) & \text{if } a = c \end{cases}$

7.6 How to prove? seems so obvious
(This is my concern!)

8 Number Theory

- Study of integers

(my favorite chapter)

Divisability

$a \mid b = ak = b$ for some k

b is divisable by a

Prime

> 1 that $p \mid 1$ and $p \mid p$ and nothing else

Linear Combo

$n = s_0 b_0 + s_1 b_1 + \dots + s_n b_n$

for some int $s_0, \dots, s_n$

Remainder

$n = q \cdot d + r$ AND $0 \leq r < d$

$qcnt(n,d) \rightarrow \frac{n}{d}$
$rem(n,d) \rightarrow$

(22)

Die Hard was actually a linear combo of previous results

- prove inductivly

Greatest common divisor (GCD)

- the largest possible # that divides them both

Euclid's Algorthm

$\gcd(a,b) = \gcd(b, \text{rem}(a,b))$

repeat

Pulviser

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

for some integer s, t

An int is a linear combo of a AND b
iff mutiple of $\gcd(a,b)$

Can keep track of remainder

$x \quad y \quad \text{rem}(x,y) = x - q \cdot y$

? keep plugging for
to maintain linear combo

(23) Jugs

3 can be written as a linear combo of 21, 26

Since 3 is multiple of GCD(21, 26) = 1

$3 = s \cdot 21 + t \cdot 26$

One will be negitive

Keep circling

Fund Theorm of Arithmetic

- every (+) int is a product of unique weakly ↓ seq of primes
  - ↑ only 1 way

Alan Turing

- skipping code 1,0

Modular Arithmetic

a is congruent to b modulo n if $n \mid (a-b)$

$a \equiv b \mod (n)$

↑ $rem(a,n) = rem(b,n)$ ← They have same remainder when /n

(I had a hard time getting this at first)

29

Multiplicative Inverse

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

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

Only 1, -1 have integer inverses

* Lemma 8.6.1 If p is prime and k is not a multiple of p, we must have $\gcd(p, k) = 1$

So

$$sp + tk = 1$$

$$sp = 1 - tk$$

$$p \mid (1 - tk)$$

$$tk \equiv 1 \pmod{p}$$

$$m \cdot k^{-1} = rem(mk, p)\, k^{-1}$$

$$\equiv (mk)k^{-1} \pmod{p}$$

$$\equiv m \pmod{p}$$

$$m = rem(m \cdot k^{-1}, p)$$

(I will never be good at crypto!)

Cancellation - does not work

But this does

$$ak \equiv bk \pmod{p} \rightarrow a \equiv b \pmod{p}$$

↑ if p is prime

and k is not a muliple of p

Fermat's Little Theorm

- alt approach to finding inverse of secret key k

$$k^{p-1} \equiv 1 \pmod{p}$$

(In this section - focus on just leaning material, let alone proofs!)

$$\text{So} \quad k \cdot k^{p-2} \equiv 1 \pmod{p}$$

↑ must be multiplicate inverse of k

Known-Plaintext Attack

$$m^* = m^k \pmod{p}$$

$$m^{p-2} \cdot m^* = m^{p-2} \cdot \text{rem}(m^k, p)$$

$$= m^{p-2} \cdot m^k \pmod{p}$$

$$= m^{p-1} \cdot k \pmod{p}$$

$$= k \pmod{p}$$

(26)

Turing's code didn't work out

But RSA was big

- public key and private key

Arithmetic w/ Arbitrary Modulus (not just prime)

<u>Relatively prime</u> if $\gcd(a, b) = 1$

- think of it as pair
- every int is rel prime to a prime ∄ p

Arithmetic modulo rel. prime still kinda well behaved

$k \cdot k^{-1} = 1 \pmod{n}$

- n is ⊕ int
- k rel prime to n
- k^{-1} exists then

If $ak \equiv bk \pmod{n}$

then $a \equiv b \pmod{n}$

- n is ⊕ int
- k is rel prime to n

(27)

Euler's Theorm

$\phi(n)$ is # of ints $[0, n)$ that are rel. prime to n

If $n =$ prime $\phi(n) = n-1$

But if composite ... complex

$k^{\phi(n)} = 1 \mod n$

(skipping proofs in this section - prob bad choice)

Computing

$\phi(n)$ with $n = pq$: $\phi(pq) = (p-1)(q-1)$ — p, q are primes, $p \neq q$

Need to know the primes that make up

Otherwise hard

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

Can break up

$$\phi(300) = \phi(2^2 \cdot 3 \cdot 5^2)$$
$$= \phi(2^2) \cdot \phi(3) \cdot \phi(5^2)$$
$$= (2^2 - 2^1) \cdot (3^1 - 3^0) \cdot (5^2 - 5^1)$$
$$= 80$$

RSA Cryptosystem

Before hand

1. Generate 2 distinct primes
2. $n = pq$
3. Select e so that
$$\gcd(e, (p-1)(q-1)) = 1$$
4. Public key is (e, n)
5. Compute d
$$de \equiv 1 \mod (p-1)(q-1)$$
ie find the multiplicate inverse of $e \bmod (p-1)(q-1) = d$
6. Secret key is (d, n)

Encoding

Sender checks $\gcd(m, n) = 1$

Then encrypts $m^* = \text{rem}(m^e, n)$

Decoding

$$m = \text{rem}((m^*)^d, n)$$

(29) (Getting into graphs)

9 Directed graphs + Partial Orders

digraphs = directed graphs = with arrows

nodes

directed edges / arrows $e = \langle u \to v \rangle$

indeg $\{ e \in E(G) \mid heads(e) = v \}$

$\sum_{v \in V(G)}$ all indeg $= \sum$ out deg

Simply "binary relation where domain and codomain are same set V

Walk - Sequence

Path = must be unique

Can merge paths

Shortest walk is a path

└ lenght is called the distance (of shortest path)

adjancy matricies - 1 if arrow

Can square to find # of lenght 2 walks b/w those 2 vertices

Path Relation

binary relation G^* called path relation on $V(G)$

$u G^* v$ = path in G from u to v

G^+ = positive path relation

is transitive

always reflexive w/ lenght 0 paths

(but are not ⊕ paths)

Closed walk if begins + ends at same vertex

Cycle - closed walk where vertices distinct except start + end

DAG (Directed Acyclic Graph) - no pos leng sycles

- like the course ordering
- must be a-symmetric
- and irreflexive Not $(a \; R \; a)$

Strict partial order - transive, asymmetric, irreflexive

└ DAGs are strict partial order

31

Weak Partial Orders - are non-diagraphs

transitive
antisymmetric
reflexive

So basically if strict or they are the same thing

$\leq$, $\sqsubseteq$ is ~~str~~ weak partial order
$<$, $\sqsubset$ is strict " "

Total ~~order~~ partial order where every item is comparable
- ie 8.01 6.042 are not comparable
  - no relation specified

Can represent by set containment
Can multiply orders to create new ones

Scheduling is a partial order problem
topological sort of a partial order is total ordering

$a \preceq b \rightarrow a \sqsubseteq b$

↳ just means put 1 item after the other
ie 1 class a semester
every partial order has a topo sort

<u>Parallel schedule</u>

- set of elements scheduled at t_k
- # of items is min # of processors req
  (# of classes taken per semester)

<u>Chain</u> all of the series of events that are comparable

<u>Critical path</u> - the longest chain
↳ lenght of this path = chain
- must have at least that many timesteps called <u>parall time</u>

~~Dilworth's Lemma → antichain~~

<u>antichain</u> - set of elements such that every 2 els uncomparable
if largest chain is of size t, then A can
be partioned into t antichains

<u>Dilworths</u> - for t > 0, every partially ordered set n el
must have chain > t or antichain > $\frac{n}{t}$

Every partially ordered set n els
has chain size > $\sqrt{n}$
or antichain < $\sqrt{n}$

Relation is equivilant if
- reflexive
- symmetric
- transtive

Partition - group disjoint nonempty subsets

---

11 Simple Graphs

- symmetric
- like getting married or having sex
- edge {u,v}
- two vertices <u>adjecent</u> if have an edge b/w them
- an edge is <u>incident</u> to its endpoints
- # of edges = <u>degree</u>
- no simple loops

Sexual Demographics example

$$\sum_{x \in M} deg(x) = \sum_{y \in F} deg(y)$$

divide both side by $|M| \cdot |F|$

$$\left(\frac{\sum_{x \in M} deg(x)}{|M|}\right) \cdot \frac{1}{|F|} = \left(\frac{\sum_{x \in F} deg(y)}{|F|}\right) \cdot \frac{1}{|M|}$$

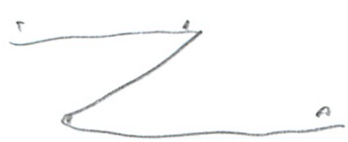
$$\text{Avg deg in } M = \frac{|F|}{|M|} \cdot \text{Avg deg in } F$$

Handshaking Lemma - sum of deg in graph is 2x # edges

Complete Graph - edge b/w each + every node

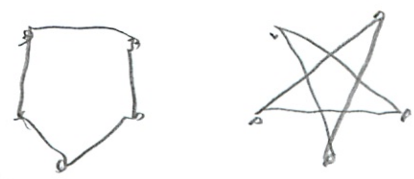
Empty Graph - no edges at all

line graph - one line



Isomorphism - same # points and lines

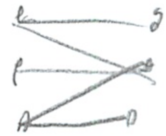
- but diff org of points
- technically is a bij



(35)

Biparte graph

two sides left and right



matching - every thing on the left has a line out and things on right can only be used once

Matching condition every subset on left likes at lest as large a set of women

A matching for set of left L with a set W on right can be found only if Matching condition holds

ie 3 men must like ≥ 3 females

vertex can't be incendent to more then one edge

Covers every vertex is included

Perfect if covers V(G) ∈ the graph

(?, I don't really get this - every line used?)

Hall's Theorm is a matching G that covers L(G) if no subset of L(G) is a bottleneck

bottleneck $|S| > |N(S)|$

^ neighbors

(B.6) (these all seem to be diff ways of saying same thing)

degree-constrained $deg(l) \geq deg(r)$

for every $l \in L(G)$ and $r \in R(G)$

If G is degree-constrained, there is a matching

regular - every node has same degree

- every regular bipate graph has a perfect matching

Couples example

- can show preserved invarients
- no rouge ~~sam~~ couples

Coloring - can't give adjecent vertices same color

goal is to use as few colors as possible

Called the chromatic number $\chi(G)$

good for scheduling

bipate = 2

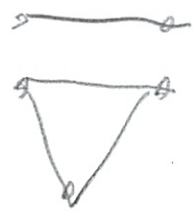
degree k is at worst k+1 colorable

Walk - same as before?

(37)

Subgraphs - can define cycles as
- (don't get it really)

Connected - when every pair of verticy is connected

So  ← not connected

? That is 2 connected components

k-edge connected is how many edges can be removed for it to be connected
↳ the last one is the cut edge
- if it is not on a cycle

Will be atleast V - e connected components

2-colorable -

all are eqivilant
1. has odd lenght cycle
2. not 2 colorable
3. Has an odd length closed walk

Trees

- no cycles
- connected, acyclic, graph

leaves - node w/ degree 1

forest - collection of trees
- that are seperate

1. Every connected subgraph is a tree
2. There is a unique simple path b/w every pair of vertices
3. Adding an edge creates a cycle
4. Removing an edge disconnects graph (cut edge)
5. If tree has at least 2 vertices, has at least 2 leaves
6. The # of vertices in a tree is 1 larger than # of edges

Spanning tree - set of least ~~##~~ lines ~~that~~ needed for graph to still be connected

Minimum weight spanning tree (MST) - if lines have weights
- minimize that

12 Planar Graphs

- drawling lines
- prove can be connections w/o lines overlapping
- important in circuit design

drawing - actually writing stuff out (1 version)

planar - no lines cross

face - the continous regions

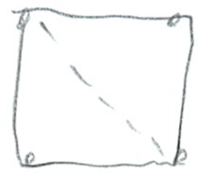
- don't forget the outside face

bridge - not bounded by a cycle

dangle - not really a cycle since parts transversed twice

Can build by

- spliting a face



- adding a bridge

Euler's formula

- for connected graph w/ a planar embedding

$$v - e + f = 2$$

Banding edges

$e \leq 3v - 6$ if $v \geq 3$ vertices, e edges

So can say if something is planar or not

Minor graph that can be obtained by repetedly deleting vertices, deleting edges, merging adj vertices of G

Any subgraph of a planar graph is planar

Merging 2 planar graphs gives a planar graph

(11)

Every planar graph has a vertex of degree at most 5

Every planar graph is 5-colorable

(This should be open-book, like prior semesters)

Polyhedron convex 3D region bounded by a finite # of polygon faces

---

13 State Machines

(I don't remember this chap)

? What is this alternating bit protocol?

- proof

While programs

- Conditional w/ test and branches

runs in an environment

(Yeah I have no clue on this chap)

6.042 OH

5/13

(only one here - what to review)

(Its largly a matter of doing quizes

How you wold do it

Abstract math

Style of content

More time

---

* Care of proof of theorm

Think about how you prove it first

Then read how to get around tricky cases

Thats how you learn defs

Choose on ~~qualit~~ quality

- basics of each subject

Understand problem

②

MQ2-1 — Need learning rules to manipulate

MQ2-2

Counting

When it will appear on the list

Or explicit ordering through this

... - when does the other thing happen

Is something to prove here

Seqence

- Say go in diagonals

How show?

Could also do for real

- but not countable

Ordering in sequence

I didn't realize why proving rationals is Countable is an achievement - whats important here

(3)

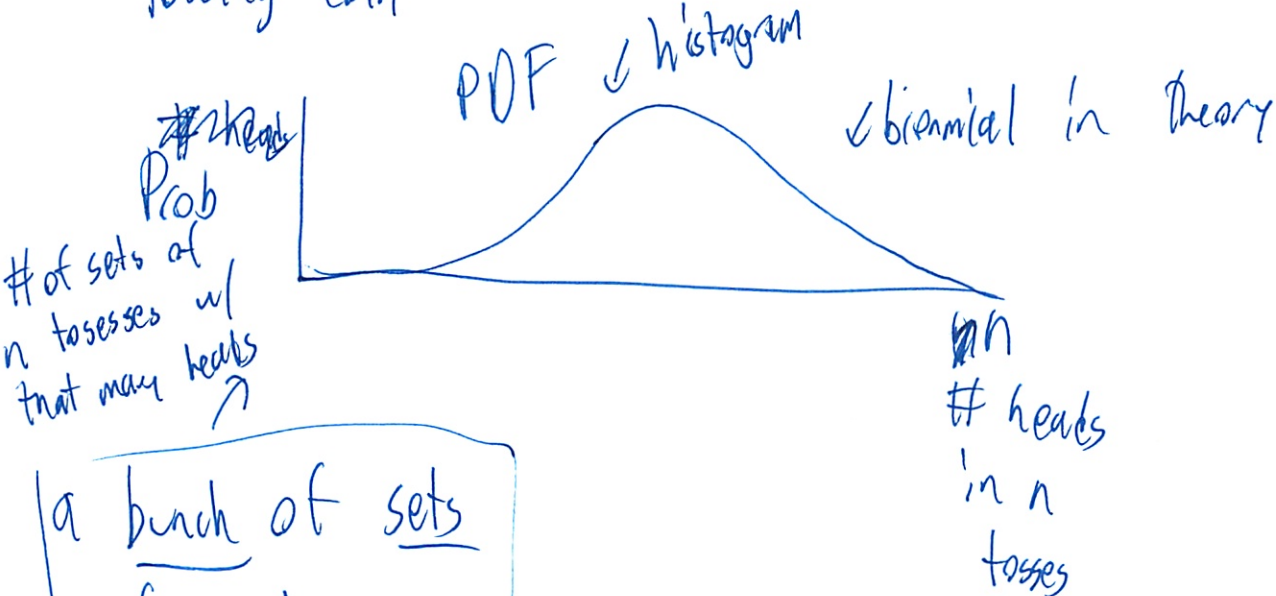
* what is the challenge here

Making progress in each

(I think I just don't have the mindset for this class
- how to change?
- retaking would def help)

Confidence vs prob

Tossing coin

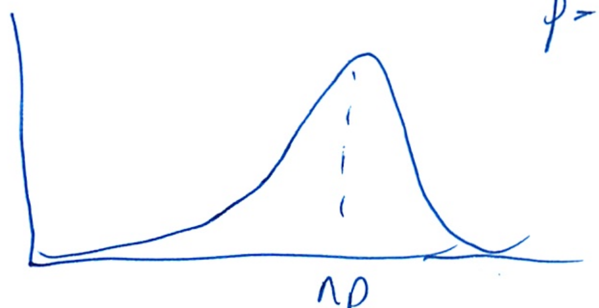


a bunch of sets of n tosses

Out of chance may get something if only do it once

(4)

If had a coin biased to H



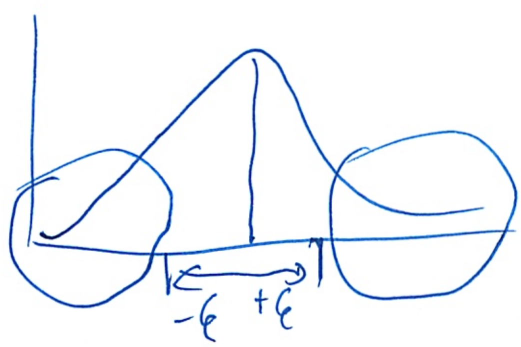
But can't see idea graph

Need to estimate by doing a bunch of sets of n tosses

So $\frac{\# Hs}{n}$ is estimate of p

Could get estimate p wrong - off of real value

So ask what is prob you are real ± error ?



↑ So sum of this

Is where confidence comes in

The Greater ε is the more likely you are in the window

(5)

Estimate is correct only a fraction of time

↳ confidence = fraction of time you are right

~~fractio~~ Could find confidence if knew dist

But often you don't know the exact dist

- need to do approx

Need def of "wrong"

For Pset - cutoff decision rule

Any cutoff will have mistakes though

(I think once I see the sol its obvious, but initally coming up w/ it is hard)

6.00 Notes On Big-O Notation

March 15, 2011

Sarina Canelake

See also http://en.wikipedia.org/wiki/Big_O_notation

- We use big-O notation in the analysis of algorithms to describe an algorithm's usage of computational resources, in a way that is independent of computer architecture or clock rate.
- The worst case running time, or memory usage, of an algorithm is often expressed as a function of the length of its input using big O notation.
  - In 6.00 we generally seek to analyze the worst-case running time. However it is not unusual to see a big-O analysis of memory usage.
  - An expression in big-O notation is expressed as a capital letter "O", followed by a function (generally) in terms of the variable n, which is understood to be the size of the input to the function you are analyzing.
  - This looks like: $O(n)$.
  - If we see a statement such as: $f(x)$ is $O(n)$ it can be read as "f of x is big Oh of n"; it is understood, then, that the number of steps to run $f(x)$ is linear with respect to $|x|$, the size of the input x.
- A description of a function in terms of big O notation only provides an *upper bound* on the growth rate of the function.
  - This means that a function that is $O(n)$ is also, technically, $O(n^2)$, $O(n^3)$, etc
  - However, we generally seek to provide the tightest possible bound. If you say an algorithm is $O(n^3)$, but it is also $O(n^2)$, it is generally best to say $O(n^2)$.
- Why do we use big-O notation? big-O notation allows us to compare different approaches for solving problems, and predict how long it might take to run an algorithm on a very large input.

  With big-O notation we are particularly concerned with the *scalability* of our functions. big-O bounds may not reveal the fastest algorithm for small inputs (for example, remember that for $x < 0.5$, $x^3 < x^2$) but will accurately predict the long-term behavior of the algorithm.
  - This is particularly important in the realm of scientific computing: for example, doing analysis on the human genome or data from Hubble involves input (arrays or lists) of size well into the tens of millions (of base pairs, pixels, etc).

- At this scale it becomes easy to see why big O notation is helpful. Say you're running a program to analyze base pairs and have two different implementations: one is $O(n \lg n)$ and the other is $O(n^3)$. Even without knowing how fast of a computer you're using, it's easy to see that the first algorithm will be $n^3/(n \lg n) = n^2/\lg n$ faster than the second, which is a BIG difference at input that size.

big-O notation is widespread wherever we talk about algorithms. If you take any Course 6 classes in the future, or do anything involving algorithms in the future, you will run into big-O notation again.

- Some common bounds you may see, in order from smallest to largest:
  - $O(1)$: Constant time. $O(1) = O(10) = O(2^{100})$ - why? Even though the constants are huge, they are still *constant.* Thus if you have an algorithm that takes 2^{100} discreet steps, regardless of the size of the input, the algorithm is still $O(1)$ - it runs in constant time; it is *not dependent upon the size of the input.*
  - $O(\lg n)$: Logarithmic time. This is slower than linear time; $O(\log_{10} n) = O(\ln n) = O(\lg n)$ (traditionally in Computer Science we are most concerned with $\lg n$, which is the base-2 logarithm – why is this the case?). The fastest time bound for search.
  - $O(n)$: Linear time. Usually something when you need to examine every single bit of your input.
  - $O(n \lg n)$: This is the fastest time bound we can currently achieve for sorting a list of elements.
  - $O(n^2)$: Quadratic time. Often this is the bound when we have nested loops.
  - $O(2^n)$: Really, REALLY big! A number raised to the power of n is slower than n raised to any power.

- Some questions for you:
  1. Does $O(100n^2) = O(n^2)$?
  2. Does $O(\frac{1}{4}n^3) = O(n^3)$?
  3. Does $O(n) + O(n) = O(n)$?

The answers to all of these are Yes! Why? big-O notation is concerned with the long-term, or *limiting,* behavior of functions. If you're familiar with limits, this will make sense - recall that

$$\lim_{x \to \infty} x^2 = \lim_{x \to \infty} 100x^2 = \infty$$

basically, go out far enough and we can't see a distinction between $100x^2$ and x^2. So, when we talk about big-O notation, we always *drop coefficient multipliers* - because they don't make a difference. Thus, if you're analysing your function and you get that it is $O(n) + O(n)$, that doesn't equal $O(2n)$ - we simply say it is $O(n)$.

One more question for you: Does $O(100n^2 + \frac{1}{4}n^3) = O(n^3)$?

Again, the answer to this is Yes! Because we are only concerned with how our algorithm behaves for very large values of n, when n is big enough, the n^3 term will always dominate the n^2 term, regardless of the coefficient on either of them.

In general, you will always say a function is big-O of its largest factor - for example, if something is $O(n^2 + n \lg n + 100)$ we say it is $O(n^2)$. Constant terms, no matter how huge, are always dropped if a variable term is present - so $O(800 \lg n + 73891) = O(\lg n)$, while $O(73891)$ by itself, with no variable terms present, is $O(1)$.

See the graphs generated by the file `bigO_plots.py` for a more visual explanation of the limiting behavior we're talking about here. Figures 1, 2, and 3 illustrate why we drop coefficients, while figure 4 illustrates how the biggest term will dominate smaller ones.

Now you should understand the What and the Why of big-O notation, as well as How we describe something in big-O terms. But How do we get the bounds in the first place?? Let's go through some examples.

1. We consider all mathematical operations to be constant time ($O(1)$) operations. So the following functions are all considered to be $O(1)$ in complexity:

```
def inc(x):
    return x+1

def mul(x, y):
    return x*y

def foo(x):
    y = x*77.3
    return x/8.2

def bar(x, y):
    z = x + y
    w = x * y
    q = (w**z) % 870
    return 9*q
```

2. Functions containing for loops that go through the whole input are generally O(n). For example, above we defined a function `mul` that was constant-time as it used the built-in Python operator *. If we define our own multiplication function that doesn't use *, it will not be O(1) anymore:

```
def mul2(x, y):
    result = 0
    for i in range(y):
        result += x
    return result
```

   Here, this function is O(y) - the way we've defined it is dependent on the size of the input y, because we execute the `for` loop y times, and each time through the `for` loop we execute a constant-time operation.

3. Consider the following code:

```
def factorial(n):
    result = 1
    for num in range(1, n+1):
        result *= num
    return num
```

   What is the big-O bound on `factorial`?

4. Consider the following code:

```
def factorial2(n):
    result = 1
    count = 0
    for num in range(1, n+1):
        result *= num
        count += 1
    return num
```

   What is the big-O bound on `factorial2`?

5. The complexity of conditionals depends on what the condition is. The complexity of the condition can be constant, linear, or even worse - it all depends on what the condition is.

```
def count_ts(a_str):
    count = 0
    for char in a_str:
        if char == 't':
            count += 1
    return count
```

In this example, we used an `if` statement. The analysis of the runtime of a conditional is highly dependent upon what the conditional's condition actually is; checking if one character is equal to another is a constant-time operation, so this example is linear with respect to the size of `a_str`. So, if we let $n = |\texttt{a\_str}|$, this function is $O(n)$.

Now consider this code:

```
def count_same_ltrs(a_str, b_str):
    count = 0
    for char in a_str:
        if char in b_str:
            count += 1
    return count
```

This code looks very similar to the function `count_ts`, but it is actually very different! The conditional checks if `char in b_str` - this check requires us, in the *worst case*, to check every single character in `b_str`! Why do we care about the worst case? Because big-O notation is an upper bound on the *worst-case running time*. Sometimes analysis becomes easier if you ask yourself, what input could I give this to achieve the maximum number of steps? For the conditional, the worst-case occurs when `char` is **not** in `b_str` - then we have to look at every letter in `b_str` before we can return False.

So, what is the complexity of this function? Let $n = |\texttt{a\_str}|$ and $m = |\texttt{b\_str}|$. Then, the `for` loop is $O(n)$. Each iteration of the `for` loop executes a conditional check that is, in the worst case, $O(m)$. Since we execute an $O(m)$ check $O(n)$ time, we say this function is $O(nm)$.

6. While loops: With while loops you have to combine the analysis of a conditional with one of a for loop.

```
def factorial3(n):
    result = 1
    while n > 0:
        result *= n
        n -= 1
    return result
```

What is the complexity of factorial3?

```
def char_split(a_str):
    result = []
    index = 0
    while len(a_str) != len(result):
        result.append(a_str[index])
        index += 1
    return result
```

In Python, `len` is a constant-time operation. So is string indexing (this is because strings are immutable) and list appending. So, what is the time complexity of `char_split`?

If you are curious, there is a little more information on Python operator complexity here:

`http://wiki.python.org/moin/TimeComplexity` - some notes: (1) CPython just means "Python written in the C language". You are actually using CPython. (2) If you are asked to find the worst-case complexity, you want to use the Worst Case bounds. (3) Note that operations such as slicing and copying aren't $O(1)$ operations.

7. Nested for loops - anytime you're dealing with nested loops, work from the inside out. Figure out the complexity of the innermost loop, then go out a level and multiply (this is similar to the second piece of code in Example 5). So, what is the time complexity of this code fragment, if we let $n = |z|$?

```
result = 0
for i in range(z):
    for j in range(z):
        result += (i*j)
```

8. Recursion. Recursion can be tricky to figure out; think of recursion like a tree. If the tree has lots of branches, it will be more complex than one that has very few branches.

   Consider recursive factorial:

```
def r_factorial(n):
    if n <= 0:
        return 1
    else:
        return n*r_factorial(n-1)
```

   What is the time complexity of this? The time complexity of `r_factorial` will be dependent upon the number of times it is called. If we look at the recursive call, we notice that it is: `r_factorial(n-1)`. This means that, every time we call `r_factorial`, we make a recursive call to a subproblem of size $n-1$. So given an input of size n, we make the recursive call to subproblem of size $n-1$, which makes a call to subproblem of size $n-2$, which makes a call to subproblem of size $n-3$, ... see a pattern? We'll have to do this until we make a call to $n-n=0$ before we hit the base case - or, n calls. So, `r_factorial` is $O(n)$. There is a direct correlation from this recursive call to the iterative loop `for i in range(n, 0, -1)`.

   In general, we can say that any recursive function $g(x)$ whose recursive call is on a subproblem of size $x-1$ will have a linear time bound, assuming that the rest of the recursive call is $O(1)$ in complexity (this was the case here, because the `n*` factor was $O(1)$).

   How about this function?

```
def foo(n):
    if n <= 1:
        return 1
    return foo(n/2) + 1
```

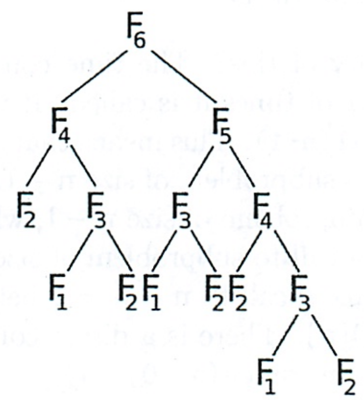
   In this problem, the recursive call is to a subproblem of size $n/2$. How can we visualize this? First we make a call to a problem of size n, which calls a subproblem of size $n/2$, which calls a subproblem of size $n/4$, which calls a subproblem of size $n/(2^3)$, ... See the pattern yet? We can make the intuition that we'll need to make recursive calls until $n = 1$, which will happen when $n/2^x = 1$.

   So, to figure out how many steps this takes, simply solve for x in terms of n:

$$\frac{n}{2^x} = 1$$
$$n = 2^x$$
$$\log_2 n = \log_2(2^x)$$
$$\therefore x = \log_2 n$$

So, it'll take $\log_2 n$ steps to solve this recursive equation. In general, we can say that if a recursive function $g(x)$ makes a recursive call to a subproblem of size x/b, the complexity of the function will be $\log_b n$. Again, this is assuming that the remainder of the recursive function has complexity of $O(1)$.

Finally, how do we deal with the complexity of something like Fibonacci? The recursive call to Fibonacci is $\mathtt{fib}(n) = \mathtt{fib}(n-1) + \mathtt{fib}(n-2)$. This may initially seem linear, but it's not. If you draw this in a tree fashion, you get something like:



The *depth* of this tree (the number of levels it has) is n, and at each level we see a branching factor of two (every call to fib generates two more calls to fib). Thus, a loose bound on fib is $O(2^n)$. In fact, there exists a tighter bound on Fibonacci involving the Golden Ratio; Google for "Fibonacci complexity" to find out more if you're interested in maths : D

I hope you found these notes helpful! Please email me at `sarina@mit.edu` if you find any typos, or if you wish to propose any corrections / better examples / additional information that you think ought to be here.

CRSR

Navigation

- Home
  - Notes
  - Infernal Device
  - Contingency
  - Check
  - SSL
  - Sendmail
  - Emacs
  - Corollaries
  - Sudoku
  - Admissions
  - Interviews
  - O(n log n)
  - Bit Count
  - Santa
  - RightToLeft
  - FrogPad
  - OSGi Logging
  - XMonad
  - Einstein
  - P and NP
  - HTTPS virtual hosts
- ProgLang
- Transportation
- Software
- Misc
- Links
- Networking

O(n log n)

Hypothetically, let us say that you are a Famous Computer Scientist, having graduated from the Famous Computer Scientist School in Smellslikefish, Massachusetts. As a Famous Computer Scientist, people are throwing money at you hand over fist. So, you have a big pile of checks. (You remember checks, little slips of paper with signatures? They represent money.)

You also have a problem: A shiny new Boeing 747 has caught your eye and you need to know whether your pile of checks represents enough money to pay for it. That problem is easy to solve, since all you need to do is to add up the amounts of the checks. But you also have a more pressing problem: Do you have enough time to sum up your checks before the 747 dealer closes for the day?

Being a Famous Computer Scientist, you know something that few others know: that you have to sort the checks, smallest to largest, before you can add them up.

A digression: That is the lowest form of computer science humor, a somewhat obscure reference to a painful fact about computing hardware. In this case the problem is that numbers such as $3141.59, $100000.00, and $2.57 are represented by the computer as *floating point* numbers. Floating point numbers are effectively represented in scientific or exponential notation, such as 3.14159 * 10^3, 1.0000000 * 10^5, or 2.57 * 10^0. The computer, however, only allocates a certain number of digits for each number, as if instead of using one digit to the left of the decimal point in the exponential notation followed by further digits to the right, it only had 3 (or 6 or 8) to the left of the decimal and *no* digits to the right. That would mean that 3141.59 would be represented in the computer as 314*10^1, 100000.00 as 100*10^3, and 2.57 as

257*10-2.

Adding floating point numbers is a multi-step process: the computer first adjusts the numbers so the exponents are the same, then adds the values. But 257^10-2 added to 100*10^3 becomes 0*10^3 or 0.0 when adjusted, so 2.57 + 100000.00 equals 100000.00.

To help mitigate that loss of precision, you need to sort your numbers from smaller to larger before you add them, in hopes that the small numbers will add up to something large enough to avoid completely disappearing when you add the sum to the larger numbers.

There is a discipline dedicated those kind of painful problems, called numerical analysis. Unfortunately, I am supremely unqualified to talk about that discipline, so I shall just quote another Famous Computer Scientist, Edsger W. Dijkstra: End of digression.

The bottom line here is that you need to sort your pile of checks before you can add them together, and you need to know how long it will take to sort them to decide whether you can buy your new bird today or not.

At this point, as a Famous Computer Scientist myself, I am going to use a very snazzy computer science trick to avoid having to continually talk about checks and 747s: abstraction. **[1]** I will throw away all the irrelevant details, including the 747 dealer (who wants to go home now anyway) and in fact both the numbers and the addition operation, and state the problem this way: you have a list of *things* that you need to sort, and you want to know how long it will take. The only operation you have on the *things* is to compare two and ask, "Is this one smaller than that one?"

Algorithm behavior

Sorting is a fundamental computer science problem and is well studied. There are many, many algorithms known for sorting. (The *algorithm* is fundamental to computer science. If you have a program, it is made up of code written by a programmer. The code implements one or, more likely, many algorithms. An algorithm is a high-level, general, abstract description of a process, while the code is the details; the low-level, concrete, specific description of the process plus a huge bundle of other trivia.)

Each of the algorithms has its own behavior, and its own speed. But what does it mean to talk about the *speed* of an algorithm, which is a thing of the mind and does not *do* anything? That is where the "Big-Oh"**[2]** notation comes in.

For every algorithm, any operation used in that algorithm, and any input given to the algorithm, there is a mathematical expression that describes how many times the operation is used on that input by that algorithm. Consider a simple algorithm, going through a list of 12 numbers counting how many there are in the list, one by one. If the operation is looking at a number to determine whether it is the last one in the list, then the operation is used 12 times by that algorithm on that input.

The expression normally depends on the size of the input: if that simple algorithm is given a list of length *n* (the variable traditionally used for such things), then it will use that operation *n* times.

Unfortunately, very often the mathematical expression is complex, not very helpful, and very unenlightening. The big-oh notation is designed to highlight the most important part of the expression while hiding irrelevant details. It describes the *asymptotic* behavior of the expression as the inputs of the expression get larger. In this case, that means that it describes the behavior of the algorithm as the size of the inputs to it get bigger.

Take, for example, the expression 37n^3 + 12n^2 + 19. If n is 1, that equals 37+12+19 = 68. If n is 10, it equals 37*1000 + 12*100 + 19 = 37000 + 1200 + 19. Clearly, as n gets bigger, the final 19 becomes irrelevant. Less clearly, 12n^2 component also becomes irrelevant, because it will be dwarfed by the first. Finally, the 37 is also irrelevant since 37 times a huge number is just another huge number. The asymptotically important part of the expression is the n^3 element. So, 37n^3+12n^2+19 is O(n^3), pronounced "order of n-cubed".

The "order of" an expression describes a kind of upper bound for the expression. It is defined as an expression which, when multiplied by some constant, is greater than or equal to the original expression for all input values larger than some other constant. Because we can choose the new expression, we can pick one that is simpler than the original in the same way that n^3 is simpler than 37n^3+12n^2+19.

(By the way, in case you were wondering, when n = 13, 37n^3+12n^2+19 equals 83336, while 38*n^3 equals 83486. So, n^3 multiplied by 38 is greater than the original expression for all input values greater than n=12. Hence, 37n^3+12n^2+19 is indeed O(n^3).)

The behavior of the counting algorithm is O(n), "order of n". How does that describe the speed of the algorithm? If each operation takes a finite (and by assumption, constant) time, then the time taken by any implementation of the algorithm will depend on the size of the input primarily by n times the time taken by the operation (plus some constant factors here and there). The Big-Oh notation provides a way of comparing two algorithms; for a sufficiently large value of n, n^2 will be greater than 10000n, and thus an O(n^2) algorithm will be slower than an O(n) algorithm for any sufficiently large input. There is a traditional hierarchy of algorithms:

- O(1) is constant-time; such an algorithm does not depend on the size of its inputs.
- O(n) is linear-time; such an algorithm looks at each input element once and is generally pretty good.
- O(n log n) is also pretty decent (that is n times the logarithm base 2 of n).
- O(n^2), O(n^3), etc. These are polynomial-time, and generally starting to look pretty slow, although they are still useful.
- O(2^n) is exponential-time, which is common for artificial intelligence tasks and is really quite bad. Exponential-time algorithms begin to run the risk of having a decent-sized input not finish before the person wanting the result retires.

There are worse; like O(2^2^...(n times)...^2).

Sorting

How does that apply to sorting? Sorting, as a problem, is clearly at least as bad as O(n), since it has to look at each item, but that is just as clearly not a good estimate. Can a better bound be found? What do the well-known sorting algorithms do?

The best sorting algorithms, going by names like "heapsort" and "quicksort", are O(n log n). But that does not necessarily mean those are the absolute best algorithms. There might well be a better one yet to be discovered. **[5]**

So, is there a way to find the performance of *any* sorting algorithm? In this case, the answer is yes.

Going back to our operation, comparison, it takes two items and says either "yes, this one is smaller than that", or "no, this one is not smaller than that". It is a *binary* comparison; it produces one of two possible answers.

A sorted list is a permutation of the original list; it is the same list with the elements rearranged. For a list of n elements, there are n! possible permutations; that is n * (n-1) * (n-2) * ... * 1. The question becomes, how many comparisons do you need to pick out one specific permutation out of the n! possibilities?

Each comparison, because it is binary, reduces the possibilities by half. So, the answer to the question is a number of comparisons, c, such that 2^c >= n!. Solving for c, that is

```
c >= log n! =
     log (n * (n-1) * ... * 1) =
     log n + log (n-1) + ... + log 1
```

The last expression is of the order of

```
log n + log (n-1) + ... + log (n/2)
```

which is of the order of

```
n/2 * log (n/2)
```

which is of the order of n log n. **[3]**

Ultimately, using only comparisons, sorting is O(n log n); any fewer comparisons would not be able to pick out a single permutation from the possibilities. **[4]**

Heapsort is therefore about as good as sorting algorithms are going to get; anything better will be only an incremental improvement. (Quicksort has bad behavior for certain inputs; for example, using quicksort on an already sorted input is O(n^2).)

A creative part of computer science is often that changing the problem, or adding information, allows dramatic improvements. For example, if I can change the problem to adding a single new element to an already sorted list while keeping it sorted, I can easily find an O(n) algorithm; simply taking the new item on the end and re-sorting would be foolish when I could just go down the list and identify the spot where the new item goes.

(That might give you an idea for beating the problem. Taking one item as a list, it as already sorted. Adding another item to the list is O(n). Adding a third is also O(n). But the resulting algorithm, called "insertion sort", for creating a sorted list from a list of n items is quite bad: Using an O(n) algorithm n times is O(n^2).)

Physics often says fundamental things about the universe. Mathematics is "the queen of the sciences" (even though it is not a science) because it frequently makes final, absolute, dramatic statements. But here is one from computer science: sorting using only basic comparisons is a fundamental problem. (Have you ever tried to find a word in an unsorted dictionary?) The shortest possible time it takes to sort n items is of the order of n log n. And, the heapsort algorithm, if it is not itself the best possible algorithm, is the neighbor of the best.

Appendix

I am seeing a lot of traffic to this page, presumably from people interested in the O(*e*) notation. If that describes you, here are some good links to learn more, in a slightly more serious fashion:

- **Big O notation** from Wikipedia is reasonably complete, reasonably correct (as far as I can see), and reasonably complex. Probably the exact opposite of the "Wikipedia is a wasteland of pop culture" stereotype.
- **Plain English Explanation of Big O Notation** is more readable and reasonably complete.
- **Big O notation from CS Animated**. What can I say, it's *animated*. Actually, it does have a very good illustration of the process of reducing the total number of operations identified in a block of code to a single, simplified Big O (or in that case, the related Big Theta) expression, as well as a thorough discussion of the ideas, aimed at budding computer scientists.
- Check out **P and NP** for similar entertainment.

I originally wrote this article not to discuss the Big O, but to highlight the properties of the sort algorithm. Algorithm analysis is complicated and interesting on its own. The most entertaining example that I know about is from an article by Jon Bently, collected in one of the Programming Pearls books if I recall correctly. He showed two algorithms for reversing an array of things with the same algorithmic complexity, and then showed that one was much, *much*, faster than the other because the slower was largely pessimal in regard to page-based virtual memory.

If you are interested in floating point numbers, check out **Anatomy of a floating point number** or the cannonical **What Every Computer Scientist Should Know About Floating-Point Arithmetic** .

Footnotes

[1] To save time, I will skip the joke (physics, I think) and just give you the punchline: "Assume that each cow is a perfect sphere of uniform density...."

[2] No, I am not referring to the manga or anime series on Cartoon

Network. And, no, I do not understand it either.

[3] I have taken this argument from Bruce Mills, *Theoretical Introduction to Programming*.

[4]

Is it possible to do better by *not* using comparisons? Yep. I recently ran across a brilliant example of this: Intelligent Design sort. Say you have a deck of 52 playing cards shuffled into a random order that you wish to sort. The probability that you could pick the three of Diamonds first, followed by the Jack of Clubs, and so on (or *whatever* order the deck is in), is 1/52 * 1/51 * ... * 1, or 1/52!. 52! is

```
80658175170943878571660636856403766975289505440883277824000000000000
```

according to my copy of *bc*. Clearly, the likelihood of this ordering is far too miniscule to happen by chance; a Higher Power *must* have arranged it. How can we, as mere mortals, improve on such? Therefore, the deck is already sorted. Unfortunately, like most things Intelligent Design, this sort algorithm has no useful properties.

For a better example, suppose that you have many, many decks of cards mixed together in a big heap, and that you want to sort them all by suit and value. There are only 52 possible combinations of face and value: Ace of Hearts, four of Spades, and so forth; every Ace of Hearts is equivalent to every other Ace of Hearts. You arrange 52 bins, examine each card once, and toss it into the appropriate bin. This algorithm, bin sort, sorts the heap in O(n), and can do so because it involves no comparisons between any two cards.

[5] For an interesting discussion (with graphs) of the difference between n log n and n algorithms, see **O(N log N) Complexity - Similar to linear?** on stackoverflow. (And yes, I did give up on keeping the notes in order in the text.)

Return to Top | **About this site...**
Last edited Tue Aug 24 16:45:36 2010.

gloria i ad inferni faciamus opus

I looked at quick sort, heap sort

The ln n comes from an array where an item is removed each time

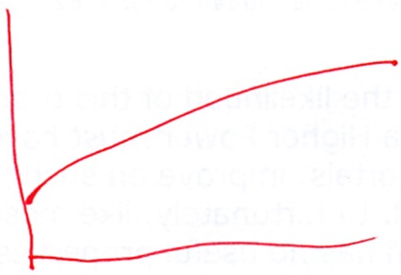
- first time n to look through
- next n-1

So $n + n-1 + n-2 \cdots + 1 + 0$

Like when $n = 10$

$$10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 + 0$$

$$= 55$$



$\ln(10) = 2.30$

$\frac{\ln(10)}{\ln(2)} = 3.32$ ← well of course actual # may not be close — but its the spirit

(12)

14 Sums + Asymptotics

5/15

(This is the furtherst unit back I remember doing recently) (going to try more things out - not just record facts) (On cheat sheet anyway) (or should be)

Can add stuff up in closed form

$1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$

Solve w/ perturbation method

$S = 1 + x + x^2 + x^3 + \cdots + x^n$

multiply by x

$xS = x + x^2 + x^3 + x^4 + \cdots x^{n+1}$

{what is goal
S = some small # of terms

$S - xS = 1 - x^{n+1}$

Solve for S

$S(1-x) = 1 - x^{n+1}$

$S = \frac{1 - x^{n+1}}{1-x}$,

but this is not always possible!

(43)

Annuity value can do too

∞ geometric series

look at proof

$$\lim_{n \to \infty} \sum_{i=0}^{n} x^i$$

$$\lim_{n \to \infty} \frac{1-x^{n+1}}{1-x}$$ by what we did before

that is $1-x^{\infty}$

$$\frac{1-\infty}{1-x}$$

$$\frac{-\infty}{1-x}$$

$-\infty$

No!

$\frac{1}{1-x}$ somehow!

Oh $\lim_{n \to \infty} x^{n+1} = 0$ when $|x|<1$

ohhhh ?

but how did we know that

Oh specified at begining

(14)

Can also differentiate or integrate w/ respect to 1

$$\frac{d}{dx}\left(\sum_{i=0}^{n-1} x^i\right) = \frac{d}{dx}\left(\frac{1-x^n}{1-x}\right)$$

$$\sum_{i=0}^{n-1} \frac{d}{dx}\left(x^i\right) =$$

↓ ? how do we know this step?
guess it just simply follows

$$\sum_{i=0}^{n-1} i x^{i-1} =$$

$$= \frac{-nx^{n-1}(1-x)-(-1)(-x^n)}{(1-x)^2}$$

$$= \frac{-nx^{n-1}+nx^n+1-x^n}{(1-x)^2}$$

$$= \frac{1-nx^{n-1}+(n-1)x^n}{(1-x)^2}$$

messes up exponent
so multiply by x

$$\sum_{i=1}^{n-1} i x^i = \frac{x-nx^n+(n-1)x^{n+1}}{(1-x)^2}$$

(45)

(so much to study here!)

Can also approx. sums

$$S = \sum_{i=1}^{n} \sqrt{i}$$

No closed form expression known

But can estimate

$$I + f(1) \leq S \leq I + f(n)$$

$$I = \int_1^n f(x)\,dx$$

$$S = \sum_{i=1}^{n} f(i)$$

Shaded area proof

Blocks hanging over table

Harmonic #

$$H_n \sum_{i=1}^{n} \frac{1}{i}$$

Can estimate

Close to $\ln(n)$

Can say if something is asy. =

$$f(x) \sim g(x)$$

$$\lim_{x \to \infty} \frac{f(x)}{g(x)} = 1$$

$\prod$ means multiply

$$n! = \prod_{i=1}^{n} i$$

Can convert to sum by taking a log

$$P = \prod_{i=1}^{n} f(i)$$

$$\ln(P) = \sum_{i=1}^{n} \ln(f(i))$$

Then can find, or approx closed form for $\ln(n!)$

$$\ln(n!) = \sum_{i=1}^{n} \ln(i)$$

Can use the approx for bounds

47

Stirling's Formula

$$n! \sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{\epsilon(n)}$$

$$\frac{1}{12n+1} \le \epsilon(n) \le \frac{1}{12n}$$

Just don't ask where it came from!

Sometimes double summation - do one at a time

Asymptotic notation

Little oh

$f(x) = o(g(x))$

iff $\lim_{x \to \infty} \frac{f(x)}{g(x)} = 0$

asy. smaller

$x^a = o(x^b)$ for all nonneg constants $a < b$

(skipping the rest of this)

(48)

<u>Big O</u> upper bound

$f = O(g)$

$\limsup \frac{f(x)}{g(x)} < \infty$

If $f = o(g)$ or $f \sim g$ then $f = O(g)$
little oh

If $f = o(g)$ then $g \neq O(f)$
not true that

<u>Theta</u> if running time is precisely quadratic - upper bound and lower bound

$T(n) = O(n^2)$
$n^2 = O(T(n))$ } $T(n) = \Theta(n^2)$

T is order of n^3

(How does all this really fit in to software?)

<u>Pitfalls</u> Constants are $O(1)$

$\sum O(1) = O(1)$

For lower bound $n^2 = O(T(n))$
a very interesting way of putting it

(4)

Omega for lower bounds

$f = \Omega(g)$ is $g = O(f)$

- that pitfall on the last pg
(I get this now!)

Little Omega grows strictly faster than another function

$f = \omega(g)$ means $g = o(f)$

---

15 Cardinality / Counting Rules

Count one thing by counting another
If bij b/w them + something countable
Like where you put ones in a seq

Product Rule $|P_1 \times P_2 \times \ldots \times P_n| = |P_1| \cdot |P_2| \cdot \ldots \cdot |P_n|$

like size · flavor · ice or not

Sum Rule for disjoint sets
just add it up

$|A_1 \cup A_2 \cup \ldots \cup A_n| = |A_1| + |A_2| + |A_3| + \ldots + |A_n|$

So can have password 6-8 symbols
first must be letter
rest letter or #

$(F \times S^5) \cup (F \times S^6) \cup (F \times S^7)$

$S \times S \times S \times S \times S$

$= 52 \cdot 62^5 + 52 \cdot 62^6 + 52 \cdot 62^7$
$\approx 1.8 \cdot 10^{14}$ diff passwords

But if have unique items

$|S| = n \cdot (n-1) \cdot (n-2)$
↑ # ways to award 3 awards to n people
each person only gets 1 award max

(I think I can understand and do all this)
This is call permutation $n!$

Division Rule k-to-1 function
like 10 fingers-to-1 person (assuming no amputees)

$|A| = k \cdot |B|$
if $A \to B$ is k-to-1

(51)

Like the knights of the round table problem where any cyclic shift is same thing

---

N choose k

$\binom{n}{k}$ = # els of k el subsets in n el sets

I can select 5 books from 100 in $\binom{100}{5}$ ways

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Then bookkeeper rule $\frac{10!}{1!\,2!\,2!\,3!\,1!\,1!}$

B O K E P R

Binomial Theorm

$(a+b)^4 = \ldots$

One term for every seq of a and bs!

$$(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^{n-k} b^k$$

$$= \binom{4}{0} a^4 b^0 + \binom{4}{1} a^3 b^1 + \binom{4}{2} a^2 b^2 + \ldots$$

Can count card hands

52

Inclusion Exclusion

$$|S_1 \cup S_2| = |S_1| + |S_2| - |S_1 \cap S_2|$$

$$|S_1 \cup S_2 \cup S_3| = |S_1| + |S_2| + |S_3| - |S_1 \cap S_2| - |S_2 \cap S_3| - |S_1 \cap S_3| + |S_1 \cap S_2 \cap S_3|$$

Combinatorial Proofs

- 2 diff ways of counting something
- provide a story for each way
- like selecting employees

Pascal's Identity

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

Pidgeon Hole Principle

If more pigeons than holes, must be at least 2 pigons in at least one hole.

The magic trick ~ was not here for
I forget the exact scheme
But you concoct a scheme

Prob is last section

16 Events + Prob Space

(I should prob focus on cheat sheet)

4 step method

~~1. Flow sim~~

1. Find sample space
2. Define events of interest
3. Determin individual outcome probabilities - multiply branch
4. ~~Def~~ Compute Event Probabilites - add up marked branch

Lots of fun examples

All of the set rules

- on cheat sheet

Conditional Prob

- did already on further at leaves of table

Law of Total Prob

Independence

(skipping rewriting this - should know)

(55)

Things can only be pair-wise ind

Or they can all be ind - mutual ind

---

17 RVs

Assign outcomes to numbers on RV

Like # heads

Indicator / Bernoulli 1 or 0

CDF

Binomial dist

Expectations

- weight avg of possible values

Mean time to failure

∞ Expectations

---

18 Deviations from Mean

This starts the non-quized section, so will do cheat sheet

What was worst case varience?

- Can't find it forget about it

(56)

I've noticed not doing proofs at all for this section!

The ∞ Expectation

$$E[Q] = \sum_{k \in N} E[Q \mid T=k] \cdot P(T=k)$$

$$= \sum_{k \in N} 2^{k+1} \cdot 2^{-(k+1)} = \sum_{k \in N} = \infty$$

Done!

Final Exam

YOUR NAME: ______________________________

- **This is an open-notes exam. However, calculators are not allowed.**
- You may assume all results from lecture, the notes, problem sets, and recitation.
- Write your solutions in the space provided. If you need more space, write on the back of the sheet containing the problem.
- Be neat and write legibly. You will be graded not only on the correctness of your answers, but also on the clarity with which you express them.
- GOOD LUCK!

Problem	Points	Grade	Grader
1	15		
2	10		
3	10		
4	10		
5	10		
6	15		
7	10		
8	10		
9	10		
Total	100		

Problem 1. [15 points] Consider the following sequence of predicates:

$$\begin{aligned}
Q_1(x_1) &= x_1 \\
Q_2(x_1, x_2) &= x_1 \Rightarrow x_2 \\
Q_3(x_1, x_2, x_3) &= (x_1 \Rightarrow x_2) \Rightarrow x_3 \\
Q_4(x_1, x_2, x_3, x_4) &= ((x_1 \Rightarrow x_2) \Rightarrow x_3) \Rightarrow x_4 \\
Q_5(x_1, x_2, x_3, x_4, x_5) &= (((x_1 \Rightarrow x_2) \Rightarrow x_3) \Rightarrow x_4) \Rightarrow x_5 \\
\ldots \quad & \quad \ldots
\end{aligned}$$

Let T_n be the number of different true/false settings of the variables $x_1, x_2, \ldots, x_n$ for which $Q_n(x_1, x_2, \ldots, x_n)$ is true. For example, $T_2 = 3$ since $Q_2(x_1, x_2)$ is true for 3 different settings of the variables x_1 and x_2:

x_1	x_2	$Q_2(x_1, x_2)$
T	T	T
T	F	F
F	T	T
F	F	T

(a) Express T_{n+1} in terms of T_n, assuming $n \geq 1$.

Solution. We have:

$$Q_{n+1}(x_1, x_2, \ldots, x_{n+1}) = Q_n(x_1, x_2, \ldots, x_n) \Rightarrow x_{n+1}$$

If x_{n+1} is true, then Q_{n+1} is true for all 2^n settings of the variables $x_1, x_2, \ldots, x_n$. If x_{n+1} is false, then Q_{n+1} is true for all settings of $x_1, x_2, \ldots, x_n$ *except* for the T_n settings that make Q_n true. Thus, altogether we have:

$$T_{n+1} = 2^n + 2^n - T_n = 2^{n+1} - T_n$$

(b) Use induction to prove that $T_n = \frac{1}{3}(2^{n+1} + (-1)^n)$ for $n \geq 1$. You may assume your answer to the previous part without proof.

Solution. The proof is by induction. Let $P(n)$ be the proposition that $T_n = (2^{n+1} + (-1)^n)/3$.

Base case: There is a single setting of x_1 that makes $Q_1(x_1) = x_1$ true, and $T_1 = (2^{1+1} + (-1)^1)/3 = 1$. Therefore, $P(0)$ is true.

Inductive step: For $n \geq 0$, we assume $P(n)$ and reason as follows:

$$\begin{aligned}
T_{n+1} &= 2^{n+1} - T_n \\
&= 2^{n+1} - \left(\frac{2^{n+1} + (-1)^n}{3}\right) \\
&= \frac{2^{n+2} + (-1)^{n+1}}{3}
\end{aligned}$$

The first step uses the result from the previous problem part, the second uses the induction hypothesis $P(n)$, and the third is simplification. This implies that $P(n+1)$ is true. By the principle of induction, $P(n)$ is true for all $n \geq 1$.

Problem 2. [10 points] There is no clock in my kitchen. However:

- The faucet drips every 54 seconds after I shut off the water.
- The toaster pops out toast every 87 seconds after I plug it in.

I'd like to fry an egg for exactly 141 seconds. My plan is to plug in the toaster and shut off the faucet at the same instant. I'll start frying when the faucet drips for the D-th time and stop frying when the toaster pops for the P-th time. What values of D and P make this plan work?

$D =$ ☐ $P =$ ☐

Reminder: Calculators are not allowed.

Solution. The Pulverizer gives $5 \cdot 87 - 8 \cdot 54 = 3$. Multiplying by 47 gives:

$$\begin{aligned} 235 \cdot 87 - 376 \cdot 54 &= 141 \\ \Rightarrow \qquad 235 \cdot 87 &= 141 + 376 \cdot 54 \end{aligned}$$

Thus, I'll start frying after at drip $D = 376$ and stop 141 seconds later at pop $P = 87$.

Problem 3. [10 points] Circle either **true** or **false** next to each statement below. Assume graphs are undirected without self-loops or multi-edges.

a 1. For all $n \geq 3$, the complete graph on n vertices has an Euler tour. **false**

b 2. If a graph contains no odd-length cycle, then it is bipartite. **true**

c 3. Every non-bipartite graph contains a 3-cycle as a subgraph. **false**

d 4. Every graph with a Hamiltonian cycle also has an Euler tour. **false**

e 5. There exists a graph with 20 vertices, 10 edges, and 5 connected components. **false**

f 6. Every connected graph has a tree as a subgraph. **true**

g 7. In every planar embedding of a connected planar graph, the number of vertices plus the number of faces is greater than the number of edges. **true**

h 8. If every girl likes at least 2 boys, then every girl can be matched with a boy she likes. **false**

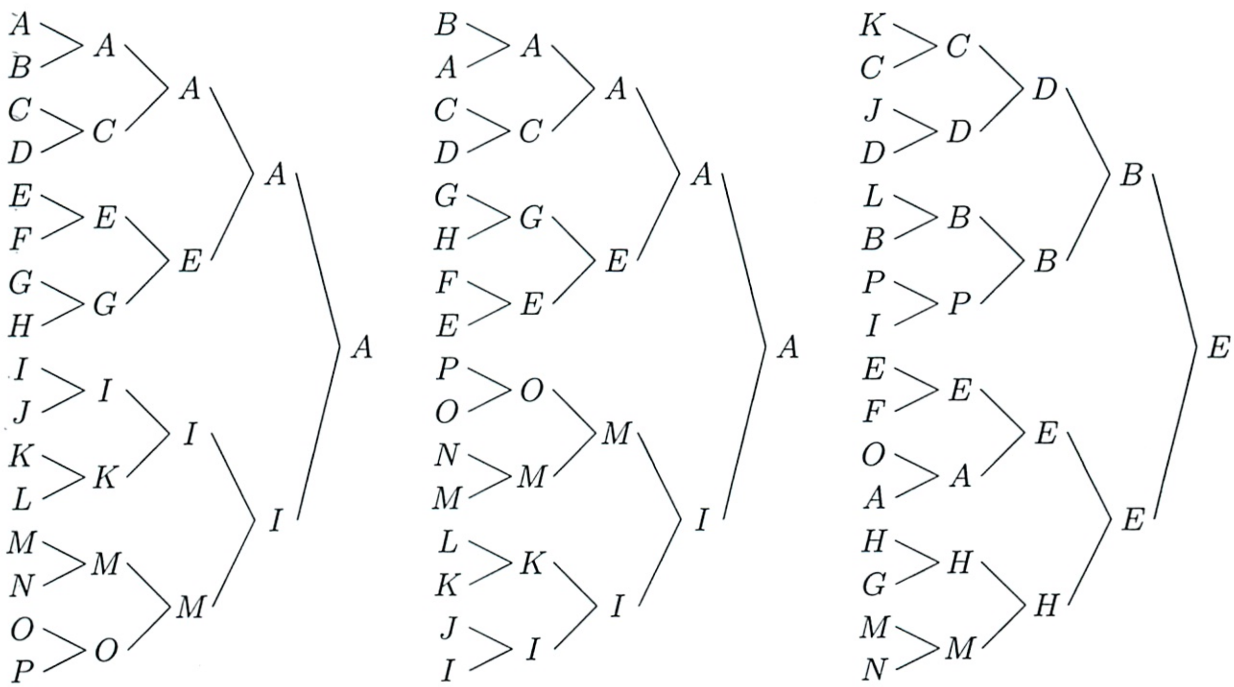
i 9. If every vertex in a graph has degree 3, then the graph is 4-colorable. **true**

j 10. There exists a six-vertex graph with vertex degrees 0, 1, 2, 3, 4, and 5. **false**

Problem 4. [10 points] In the final round of the World Cup, 16 teams play a *single-elimination tournament*.

- The teams are called $A, B, C, \ldots, P$.
- The tournament consists of a sequence of rounds.
  - In each round, the teams are paired up and play matches.
  - The losers are eliminated, and the winners advance to the next round.
  - The tournament ends when there is only one team left.

For example, three possible single-elimination tournaments are depicted below:



Two tournaments are *the same* if the same matches are played and won by the same teams. For example, the first and second tournaments shown above are the same, but the third is different. How many *different* tournaments are possible?

Solution. Suppose that we draw the tournament so that the winning team in each game is listed *above* the losing team. Then the ordering of teams on the left completely determines all matches and winners. Therefore, there are 16! single-elimination tournaments.

Another approach is to use a result from earlier in the course: the number of ways to pair up $2n$ people is $(2n)!/n!\ 2^n$. In a single-elimination tournament, we must pair up 16 teams, determine who wins the 8 matches between them, then pair up the 8 winning teams, detrmine who wins the 4 matches, and so forth. The number of ways in which this can be done is:

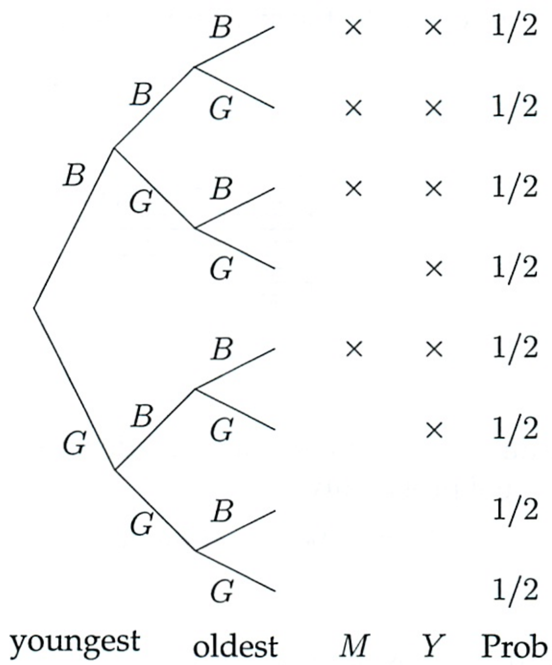
$$\frac{16!}{8!\ 2^8} \cdot 2^8 \cdot \frac{8!}{4!\ 2^4} \cdot 2^4 \cdot \frac{4!}{2!\ 2^2} \cdot 2^2 \cdot \frac{2!}{1!\ 2^1} \cdot 2^1 = 16!$$

A final alternative is to use the General Product Rule. The champions can be chosen in 16 ways, the other finalists in 15 ways, the semi-finalist that played the champions in 14 ways, the other semi-finalist in 13 ways, and so forth. In all, this gives 16! tournaments again.

Problem 5. [10 points] There are 3 children of different ages. What is the probability that at least two are boys, given that at least one of the two youngest children is a boy?

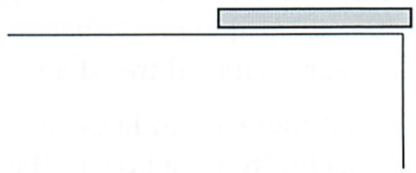
Assume that each child is equally likely to be a boy or a girl and that their genders are mutually independent. A correct answer alone is sufficient. However, to be eligible for partial credit, you must include a clearly-labeled tree diagram.

Solution. Let M be the event that there are at least two boys, and let Y be the event that at least one of the two youngest children is a boy. In the tree diagram below, all edge probabilities are 1/2.

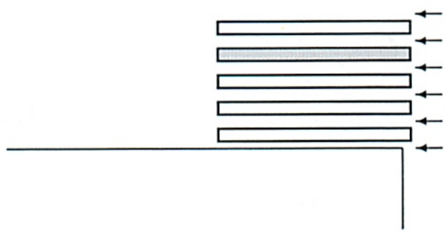


$$\Pr(M \mid Y) = \frac{\Pr(M \cap Y)}{\Pr(Y)}$$
$$= \frac{1/2}{3/4}$$
$$= 2/3$$

Problem 6. [15 points] On the morning of day 1, I put a gray document on my desk. This creates a stack of height 1:



On each subsequent morning, I insert a white document into the stack at a position selected uniformly at random. For example, the stack might look like this on the evening of day 5:



Then, on the following morning, I would insert a white document at one of the six positions indicated above with equal probability.

Let the random variable B_n be the number of white documents below the gray document on day n.

(a) Express $\Pr(B_{n+1} = 0)$ in terms of $\Pr(B_n = 0)$.

$\Pr(B_{n+1} = 0) =$

Solution.

$$\Pr(B_{n+1} = 0) = \frac{n}{n+1} \Pr(B_n = 0)$$

(b) Express $\Pr(B_{n+1} = n)$ in terms of $\Pr(B_n = n - 1)$.

$\Pr(B_{n+1} = n) =$

Solution.

$$\Pr(B_{n+1} = n) = \frac{n}{n+1} \Pr(B_n = n - 1)$$

(d) Use induction to prove that B_n is uniformly distributed on $\{0, 1, 2, \ldots, n-1\}$. You may assume your answers to the preceding problem parts without justification.

Solution. We use induction. Let $P(n)$ be the proposition that B_n is uniformly distributed on the set $\{0, 1, 2, \ldots, n-1\}$.

Base case. The random variable B_1 is always equal to 0, so it is uniformly distributed on $\{0\}$.

Inductive step. Assume that the random variable B_n is uniformly distributed on the set $\{0, 1, 2, \ldots, n-1\}$ and consider the random variable B_{n+1}. There are three cases:

$$\begin{aligned}
\Pr(B_{n+1} = 0) &= \frac{n}{n+1} \Pr(B_n = 0) \\
&= \frac{n}{n+1} \frac{1}{n} && (*) \\
&= \frac{1}{n+1} \\
\Pr(B_{n+1} = n) &= \frac{n}{n+1} \Pr(B_n = n-1) \\
&= \frac{n}{n+1} \frac{1}{n} && (*) \\
&= \frac{1}{n+1} \\
\Pr(B_{n+1} = k) &= \frac{n-k}{n+1} \Pr(B_{n+1} = k) + \frac{k}{n+1} \Pr(B_{n+1} = k-1) \\
&= \frac{n-k}{n+1} \frac{1}{n} + \frac{k}{n+1} \frac{1}{n} && (*) \\
&= \frac{1}{n+1}
\end{aligned}$$

In each case, the first equation comes from the preceding problem parts. We use the induction hypotheses on the starred lines. The remaining steps are simplfications. This shows that B_{n+1} is uniformly distributed, and the claim follows from the principle of induction.

(c) Express $\Pr(B_{n+1} = k)$ in terms of $\Pr(B_n = k)$ and $\Pr(B_n = k-1)$ assuming that $0 < k < n$.

$$\Pr(B_{n+1} = k) = \boxed{}$$

Solution.

$$\Pr(B_{n+1} = k) = \frac{n-k}{n+1} \Pr(B_{n+1} = k) + \frac{k}{n+1} \Pr(B_{n+1} = k-1)$$

Problem 7. [10 points] Bubba and Stu are shooting at a road sign. They take shots in this order:

Bubba, Stu, Stu, Bubba, Bubba, Stu, Stu, Bubba, Bubba, Stu, Stu, etc.

With each shot:

- Bubba hits the sign with probability $2/5$.
- Stu hits the sign with probability $1/4$.

What is the probability that Bubba hits the sign before Stu? Assume that hits occur mutually independently. You must give a *closed-form* answer to receive full credit.

Solution.

$$
\begin{aligned}
\Pr(\text{Bubba hits first}) &= \frac{2}{5} + \\
&\frac{3}{5}\left(\frac{3}{4}\right)^2\frac{2}{5} + \frac{3}{5}\left(\frac{3}{4}\right)^2\frac{3}{5}\frac{2}{5} + \\
&\frac{3}{5}\left(\frac{3}{4}\right)^2\left(\frac{3}{5}\right)^2\left(\frac{3}{4}\right)^2\frac{2}{5} + \frac{3}{5}\left(\frac{3}{4}\right)^2\left(\frac{3}{5}\right)^2\left(\frac{3}{4}\right)^2\frac{3}{5}\frac{2}{5} + \\
&\cdots \\
&= \frac{2}{5} + \frac{3}{5}\frac{2}{5}\sum_{i=1}^{\infty}\left[\left(\frac{3}{4}\right)^{2i}\left(\frac{3}{5}\right)^{2i-2} + \left(\frac{3}{4}\right)^{2i}\left(\frac{3}{5}\right)^{2i-1}\right] \\
&= \frac{2}{5} + \frac{6}{25}\sum_{i=1}^{\infty}\left(\frac{3}{4}\right)^{2i}\left(\frac{3}{5}\right)^{2i-2}\left(1+\frac{3}{5}\right) \\
&= \frac{2}{5} + \frac{6}{25}\frac{8}{5}\sum_{i=1}^{\infty}\left(\frac{9}{16}\right)^{i}\left(\frac{9}{25}\right)^{i-1} \\
&= \frac{2}{5} + \frac{6}{25}\frac{8}{5}\frac{25}{9}\sum_{i=1}^{\infty}\left(\frac{9}{16}\frac{9}{25}\right)^{i} \\
&= \frac{2}{5} + \frac{16}{15}\sum_{i=1}^{\infty}\left(\frac{81}{400}\right)^{i} \\
&= \frac{2}{5} + \frac{16}{15}\left(\frac{1}{1-81/400} - 1\right) \\
&= \frac{2}{5} + \frac{16}{15}\frac{81}{319} \\
&= \frac{214}{319}
\end{aligned}
$$

Problem 8. [10 points] There are three types of men (A, B, and C), and three types of women (D, E, and F). Some couples are *compatible* and others are not, as indicated below:

	A	B	C
D	no	yes	yes
E	no	no	yes
F	yes	no	no

Men and women with the personality types shown below attend a dance.

Men:	A	A	B	B	B	C	C	C	C
Women:	D	D	D	E	F	F	F	F	F

Suppose a pairing of the women and men is selected uniformly at random.

(a) What is the probability that a particular man of type A is paired with a compatible woman?

Solution. $5/9$

(b) What is the expected number of compatible couples?

Solution. Let I_k be an indicator for the event that the k-th man is paired with a compatible woman. Then the total number of compatible couples is:

$$\begin{aligned}
\text{Ex}(I_1 + \ldots + I_9) &= \text{Ex}(I_1) + \ldots + \text{Ex}(I_9) \\
&= \frac{5}{9} + \frac{5}{9} + \frac{3}{9} + \frac{3}{9} + \frac{3}{9} + \frac{4}{9} + \frac{4}{9} + \frac{4}{9} + \frac{4}{9} \\
&= \frac{35}{9}
\end{aligned}$$

Problem 9. [10 points] Every Skywalker serves either the *light side* or the *dark side*.

- The first Skywalker serves the dark side.
- For $n \geq 2$, the n-th Skywalker serves the same side as the $(n-1)$-st Skywalker with probability $1/4$, and the opposite side with probability $3/4$.

Let d_n be the probability that the n-th Skywalker serves the dark side.

(a) Express d_n with a recurrence equation and sufficient base cases.

Solution.

$$\begin{aligned} d_1 &= 1 \\ d_{n+1} &= \frac{1}{4} \cdot d_n + \frac{3}{4} \cdot (1 - d_n) \\ &= \frac{3}{4} - \frac{1}{2} d_n \end{aligned}$$

(b) Give a closed-form expression for d_n.

Solution. The characteristic equation is $x - 1/2 = 0$. The only root is $x = -1/2$. Therefore, the homogenous solution has the form $d_n = A \cdot (-1/2)^n$. For a particular solution, we first guess $d_n = c$. This is indeed a solution for $c = 1/2$. Therefore, the complete solution has the form $d_n = 1/2 + A \cdot (-1/2)^n$. Since $d_1 = 1$, we must have $A = -1/2$. Therefore:

$$d_n = \frac{1}{2} + \left(-\frac{1}{2}\right)^{n+1}$$

6.042 Pratice Final

5/15

Spring 05

1. Series of predicates
   - ~~??~~ guess its def

   T_n = # true false settings for which its true

   Implies - true if false or then true

T_1	T	✓ 1	
	F		
T_2	T T	✓	
	T F		3
	F T	✓	
	F F	✓	
T_3	(above) T T	✓	
	F F		
	T T	✓	
	T F	✓	

So never more than 4

T_4	T T	✓	always 3
	F F		
	T T	✓	
	T F	✓	

(2)

So form $n > 2$

$$T_{n+1} = T_n = 3$$

Not what they were thinking at all!

$$Q_{n+1} = Q_n \to x_{n+1}$$ (↓ Implies)

If x_{n+1} is true, then Q_{n+1} is true for all 2^n settings of x_n. If x_{n+1} is false Q_{n+1} is true for all x_n except for T_n that makes Q_n true

$$T_{n+1} = 2^n + 2^n - T_n$$
$$= 2^{n+1} - T_n$$

I had an inkling of that but when I testing things out - that's not what I got!
(But at least made an attempt - thought knew how to solve)
How to fix that ?? - I Dk!

3)

b) Use induction to prove $T_n = \frac{1}{3}(2^{n+1} + (-1)^n)$ for $n \geq 1$

; Here they are giving you the answer + asking to show how you got it

Oh here is where they quantative want ans

n = # of items

Does it mach what I fond

No - I had $T_n = 3$

Try 1, 2, 3 etc

$n=1$ $\frac{1}{3}(2^2 + (-1)^1) = 1$

$n=2$ $\frac{1}{3}(2^3 + (-1)^2) = 3$

$n=3$ $\frac{1}{3}(2^4 + (-1)^3) = 5$

$n=4$ $\frac{1}{3}(2^5 + (-1)^4) = 11$

But how to prove using fundementals of problem

- This is the ~~section~~ type of proof I always have the most trouble on

(4)

Proof by induction ← I did not know could use

Base

$T = \frac{1}{3}(2^{1+1} + (-1)^1) = 1$

which I did

Inductive

$T_{n+1} = 2^{n+1} - T_n$

$= 2^{n+1} - \left(\frac{2^{n+1} + (-1)^n}{3}\right)$

$= \frac{2^{n+2} + (-1)^{n+1}}{3}$

? So same thing except n+1

I remember being able to use this technique beginning of year

It makes sense now - but how would I have thought to do that

- guess just have to think of doing it
- plug in for

(5)

2. No clock

Faucet drips 54 sec

Toster 87 sec

Fry egg 141 sec

(This is # theory)

D, P makes it work ?

Pulvaizer

- g+v would prob also do it

- but can't have - time which pulverizer gives

Start both at same time

Oh I see when is $P \cdot 87 - D \cdot 54 = 141$

So pulvaizer can work!

$\gcd(ab) = sa + tb \quad \exists\ s, t$

? is 141 the gcd of 54, 87 ? but here we

or multiple of ?

cheated ($\gcd = 3$

$141/3 = 47$

(6)

gcd(54, 87)

		rem	
87	54	33	$= 87 - 1 \cdot 54$
54	33	21	$= 54 - 33$
			$= 54 - (87-54)$
			$= 2 \cdot 54 - 87$
33	21	12	$= 33 - 21$
			$= (87-54) - (2 \cdot 54 - 87)$
			$= 2 \cdot 87 - 3 \cdot 54$
21	12	9	$= 21 - 12$
			$= (2 \cdot 54 - 87) - (2 \cdot 87 - 3 \cdot 54)$
			$= 5 \cdot 54 - 3 \cdot 87$
12	9	3	$= 12 - 9$
			$= (2 \cdot 87 - 3 \cdot 54) - (5 \cdot 54 - 3 \cdot 87)$
			$= 5 \cdot 87 - 8 \cdot 54$
9	3	0	so 3

and $\frac{141}{3} = 47$ so i multiply by 47

7 Try it out ~~5·8~~ ~~28~~

Oh would be 3

~~4·28~~ ~~5·80~~

← they had that!

5·47 = ~~235~~ = P = 87

8·47 = 376 = D ✓

& there

235·87 − 376·54 = 141

235·87 = 141 + 376·54

I think they screwed up in the ans

I think I got this!

---

3. Undirected graph

(this stuff I am bad at!)

a) star − did not cover F

b) Not true T

c) No F

d) Halmton − not did F

e) where's my formula − oh $v - e + f = 2$

20 − 10 + faces

(8)

just try  · 5

15 vertices
10 edges
5 components

but then 5 more vertices

False ✓

6) T ✓

7. $v+f > e$?

well $v-e+f=2$

$v+f=2-e$

? what next?

cases if $e=2$

$v+f=0$ or < 2

if $e=5$

$v+f=-3 << 2$

if $e=1$

$v+f=1$ =

(No) True

I don't get why not

(9)

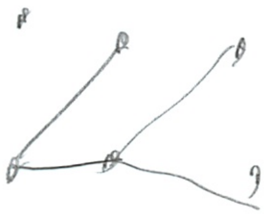
h) True - if girl picks 1s?
- depends how many boys
- can't conclude

5 girls
2 boys

F ✓

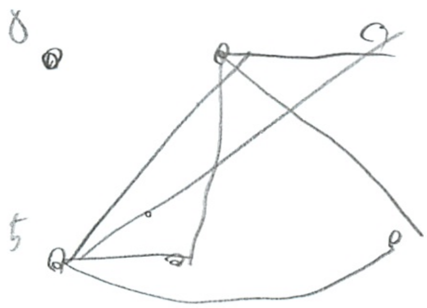
i) True ✓

j)



No!

?, sum of vertices must = something



No ✓

(10)

4. Single elimination torney

Teams A, B, ... P

Seq of cards

2 torns same if   same team - diff position

- this is isomorphism

(make sure to have formulas on sheet)

- did not write down

But think about it

← 2 choices

2^n ← be able to recognize

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

$2^2 = 4$

$2^4 = 16$

16!

(11)

Draw so winning team above losing team

16!

Or # ways to pair up 2n people = $\frac{(2n)!}{n!\,2^n}$

So a bunch of pair ups

$$\frac{16!}{8!\,2^8}\cdot 2^8 + \frac{8!}{4!\,2^4}\cdot 2^4 + \cdots \text{etc}$$

Oh I answered the wrong thing. I answered how many isomorphisms there were. They wanted # different tourneys

I could have done divison rule

$\frac{16!}{16}$ ← no — also who wins 2nd round diff!

I don't know how I would have done it

Oh I see how they did it now top one always wins

— then order matters

So don't have to remove order later!

(12)

Or General Product Rule chap ~~[illegible]~~ 16 ways
Finalist 15 ways
semi 14
other 13
etc

I don't see this either - oh each indv spot

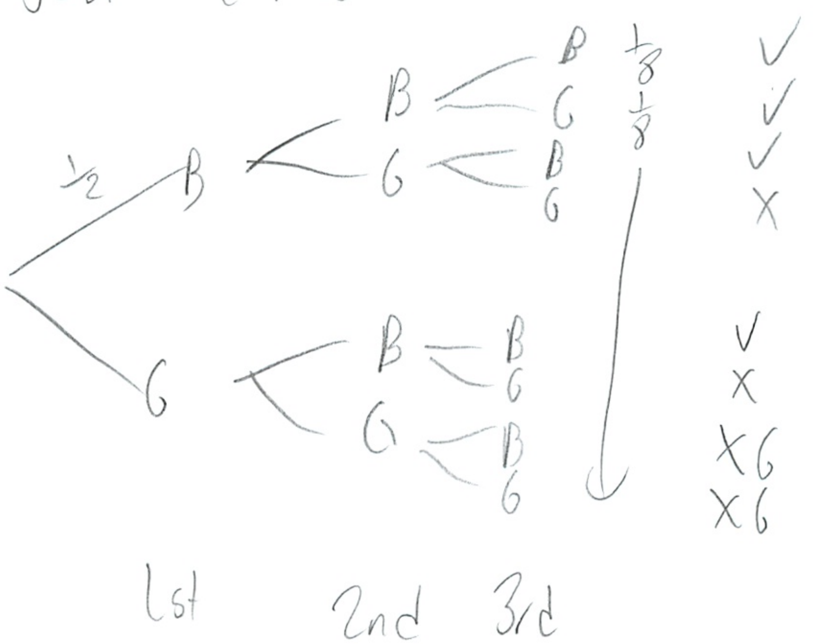
---

5. 3 children

$P(2 \text{ boys} \mid \text{one of 2 youngest is b})$

Mutually ind, =ly likly

Just a tree



So $\frac{4}{6} = \frac{2}{3}$ ✓

(13)

G. Gray Document

Put white doc in stack at random

- in one of the six pos

B_n = # white docs below gray doc @ day n

a) $P(B_{n+1} = 0)$ terms $P(B_n = 0)$

; induction

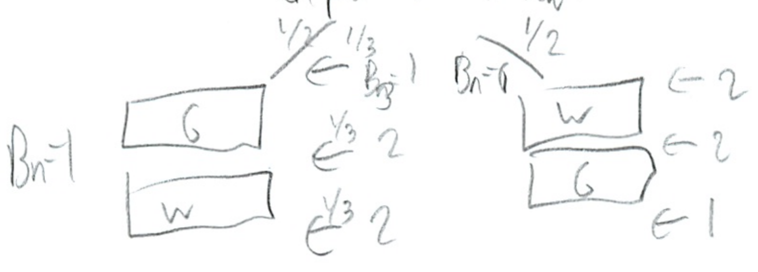
So day 2

can do top or bottom

$E[B_2] = \frac{1}{2}$

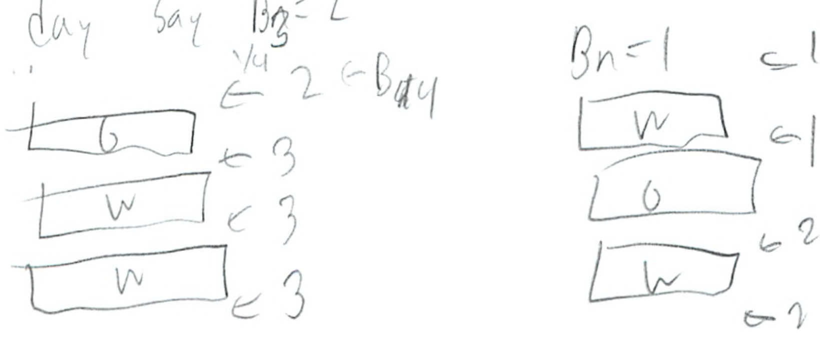
day 3

depends what I did day before



} whats the transition rule here?
- qu is asking!

k=4 Next day say $B_3 = 2$



(4)

So $B_{n+1} = B_n + \frac{B_n}{n}$

? what is this E[ ] ? not what asking!

$P(B_{n+1} = 0)$ if $P(B_n = 0)$

So ~~then~~ very simple

~~B_{n+1}~~ not what asking

~~$P(B_{n+1} = 0 | B_n = 0)$~~ $= \frac{1}{n}$

$P(B_{n+1} = 0) = \frac{n}{n+1} P(B_n = 0)$

? That is the more General ?

? isn't that ? that it will gain 1 ?

b) Try next one ~~B~~ $P(B_{n+1} = n)$ in terms $P(B_n = n-1)$

? one has been added

?

So $A=4$ if $B_n = n-1$

$B_4 = 4 - 1 = 3$

~~pre~~

And next one $A=5$ is ? impossible

— depends how it draws

(15)

~~$t=2$ $B_2 = 0$ or 1~~

~~$t=3$~~ B_3

I did lable dates wrong

~~$B_4 = 3$~~ $B_3 = 2$

then $B_4 = 3$ is $\frac{3}{4}$

? But how does that scale

That white paper is added under

$$\frac{B_n}{n}$$

? but about probs

- give up

$$P(B_{n+1} = n) = \frac{n}{n+1} P(B_n = n-1)$$

?: is this just same as above but w/ inductive as opposed to the base case

(16)

c) $P(B_{n+1}=k)$

Oh general case

$$P(B_{n+1}=k) = \frac{n-k}{n+1} P(B_{n+1}=k) + \frac{k}{n+1} P(B_{n+1}=k-1)$$

I have to get better at these general case problems

- But how?

Could I have figured it out w/ more time?

(most of these are prob!)

7. Shooting at sign

$P(B \text{ hits}) = \frac{2}{3}$

$P(S \text{ hits}) = \frac{1}{4}$

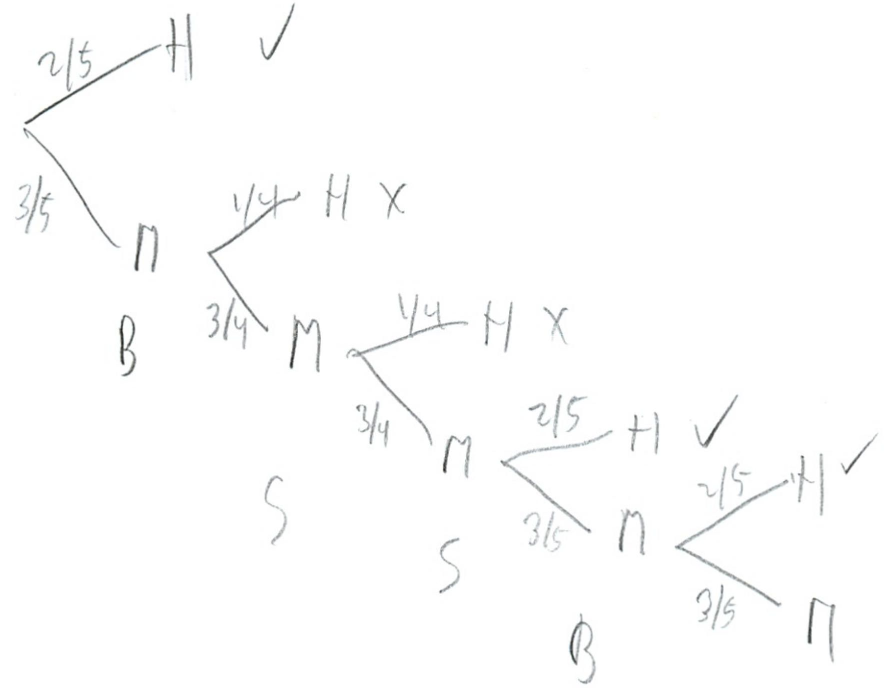
$P(B \text{ hits before } St)$

Oh one of those asy answers

? tree

↓

(17)



? How to make readable - since weird pattern above

$$E[\text{Bubba wins}] = \frac{2}{5} + \left(\frac{3}{4}\right)^2 \frac{2}{5} + \left(\frac{3}{4}\right)^2 \left(\frac{3}{5}\right)\left(\frac{2}{5}\right) + \cdots$$

? but where to go from here?

So they give the long ~~shorter~~ laid out version

← oh there is a pattern

$$= \frac{2}{5} + \frac{3}{5}\cdot\frac{2}{5} \sum_{i=1}^{\infty} \left(\left(\frac{3}{4}\right)^{2i} \left(\frac{3}{5}\right)^{2i-2} + \left(3/4\right)^{2i} \left(\frac{3}{5}\right)^{2i-1} \right)$$

You need to assume pattern in who shoots too

then fancy math to simplify ⌐ How can you do this?

$$= \frac{2}{5} + \frac{6}{25} \sum_{i=1}^{\infty} \left(\frac{3}{4}\right)^{2i} \left(\frac{3}{5}\right)^{2i-2} \left(1 + \frac{3}{5}\right)$$

(18)

$$= \frac{2}{5} + \frac{6}{25}\frac{8}{5}\sum_{i=1}^{\infty}\left(\frac{9}{16}\right)^{i}\left(\frac{9}{25}\right)^{i-1}$$

$$= \frac{2}{5} + \frac{6}{25}\cdot\frac{8}{5}\cdot\frac{25}{9}\sum_{i=1}^{\infty}\left(\frac{9}{16}\frac{9}{25}\right)^{i}$$

$$= \frac{2}{5} + \frac{16}{15}\sum_{i=1}^{\infty}\left(\frac{81}{400}\right)^{i}$$

$$= \frac{2}{5} + \frac{16}{15}\left(\frac{1}{1-81/400} - 1\right)$$

$$= \frac{2}{5} + \frac{16}{15}\frac{81}{319}$$

$$= \frac{214}{319}$$

Wow!

Let me try the convert to closed form for Σ

$$\sum_{i=1}^{\infty}\left(\frac{81}{400}\right)^{i}$$

$$= \cancel{\left(\frac{81}{400}\right)^{0}} + \left(\frac{81}{400}\right)^{1} + \left(\frac{81}{400}\right)^{2} + \left(\frac{81}{400}\right) + \text{etc}$$

to ∞! how to deal w/

= X

how did they get that?

the ∞ geometric seq rule! ← put on cheat sheet

(19)

Was on cheat sheet - need to recognize + use!

", but then why -1?

Oh it starts at 0!

(I have a feeling test will be broader - not just counting)

Will do old miniquizes tomorrow night

3 men A B C

3 women D E F

Some comptable

Some attend dance

2 As 3 Bs 4 Cs 3 Ds 1 Es 5 Fs

A man + woman are picked at random

a) P(man of type a paired well) ← setting it up!

well A can only be w/ F = $\frac{5}{9}$ ✓

(or will final tilt towards prob?)

(20)

b) E[# compatable couples]

I like how they set last one up - gave a clue!

$A \quad \frac{3}{9} \cdot \frac{5}{9} + \frac{3}{9} \cdot \frac{3}{9} + \frac{4}{9} \cdot \frac{4}{9}$

↑ P(A)

↑ A will find match

$= \frac{15}{81} + \frac{1}{9} + \frac{16}{81} = \frac{35}{81}$

↑ $\frac{9}{81}$

$\frac{35}{9}$

They did

$\frac{5}{9}$ ~~A~~ $\frac{5}{9} + \frac{5}{9} + \frac{3}{9} + \frac{3}{9} + \frac{3}{9} + \frac{4}{9} + \frac{4}{9} + \frac{4}{9} + \frac{4}{9}$

Oh since E[ ] I should have 3 # A's

not P(A)

Grrrr

(21)

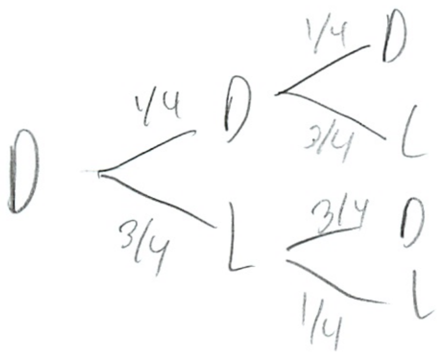
(Last problem)

9.' Skywalker either light or dark

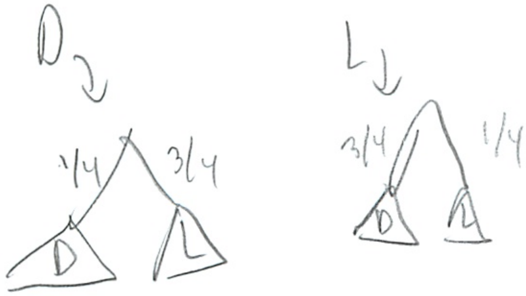
First skywalker - dark side

$n \geq 2$ nth Skywalker serves same side as Skywalker $n-1$ w/ $p = \frac{1}{4}$

d_n = prob nth serves dark side



(So that is triangle?
not as neat, again, as what I did in class
or two triangles



(but how "control" n?

$E[D_n] = 1 + \frac{1}{4}(D+1) + \frac{3}{4}(L+1)$ $N = D + L$

B(22)

$$D = \frac{1}{4}(D+1) + \frac{3}{4}L$$

$$L = \frac{1}{4}(L+1) + \frac{3}{4}D$$

? Am I confusing things?

induction!

$d_n = 1$

✓ Yeah $L = 1-D$

$$d_{n+1} = \frac{1}{4}d_n + \frac{3}{4}(1-d_n)$$

$$= \frac{3}{4} - \frac{1}{2}d_n$$

That was not that hard! What did I do wrong?

What is $1-d_n$?

d_{n+1} means new one - but $\frac{1}{4}d_n$

not $\frac{1}{4}(d_n+1)$???

Ohhh d_n is <u>Prob</u> that serves same side

Not E[ ] ← why do I keep messing that up!

$\frac{1}{4}$ · previous or $\frac{3}{4}$ its not

try $d_1 = 1$

$$d_2 = \frac{1}{4}\cdot 1 + \frac{3}{4}\cdot 0$$

$$= \frac{1}{4}$$

(23)

$d_3 = \frac{1}{4} \cdot \frac{1}{4} + \frac{3}{4} \cdot \frac{3}{4}$

$= \frac{1}{16} + \frac{9}{16}$

$= \frac{10}{16}$

?, I guess it works

But dont really see how you put it together

I guess on tree track p( ) next will be dark

b) Give closed form

$d[n+1] = \frac{3}{4} - \frac{1}{2} d[n]$

?, what do here? - consider the ds the same?

$d = \frac{3}{4} - \frac{1}{2}d$

$d + \frac{1}{2}d = \frac{3}{4}$

$d(1 + \frac{1}{2}) = \frac{3}{4}$

$d = \frac{\frac{3}{4}}{\frac{3}{2}} = \frac{3}{4} \cdot \frac{2}{3} = \frac{1}{2}$

(21)

They say

The characteristic equation is $x - \frac{1}{2} = 0$

whats this?

The only root is $x = -\frac{1}{2}$

So the homogeneous solution has form

$$d_n = A \cdot \left(-\frac{1}{2}\right)^n$$

For a particular solution, we first guess $d_n = c$

This is indeed a sol for $c = \frac{1}{2}$

Therefore the complete sol has form

$$d_n = \frac{1}{2} + A\left(-\frac{1}{2}\right)^n$$

Since $d_1 = 1$, we must have $A = -\frac{1}{2}$

So

$$d_n = \frac{1}{2} + \left(-\frac{1}{2}\right)^{n+1}$$

I didn't follow that at all! Did we do something like that in this year

---

Also test seemed weighted towards # - no proofs That might have been easier - though I would still have screwed it up - Can I do the proofs?