Arthur Zhou

Dis 1A, 10am-11:50
Parshan

1)

network

A	B
a1 ← ratings	b1 ← ratings
a2	b2
a3	b3
⋮	⋮
an	bn

n shows per network

Stable: if both networks are locked into equilibrium

Disprove by counterexample: Let's say both A & B have 3 shows:

Let's say this was initial schedule:

$A = \{10, 30, 50\}$ ← ratings (a1, a2, a3)

$B = \{20, 40, 60\}$ (b1, b2, b3)

∴ by proof of counterexample with n=3, a stable match doesn't necessarily exist

* If the initial schedule stands, B wins all 3 slots while A has none since all corresponding shows result in B's favor
* If A tries to swap its order into something like this, $A: \{30, 50, 10\}$, it'll win 2 slots but B would want to swap its order to win back more time slots

2)

S (students)	M (med schools/hospitals)
s1 (ranking) ← includes all med schools!	m1 (ranking) ← includes all students!
s2 "	m2 "
s3 "	m3 "
⋮	⋮
s_n "	m_m "

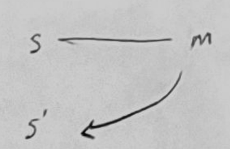
n students

m schools/hospitals

ASSUME: $n < m$

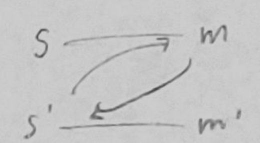
Prove these 2 cases by contradiction to prove stability:

CASE 1:



* s assigned to m
* s' has no school
* m prefers s' to s

CASE 2:



* s matches w/ m
* s' matches w/ m'
* s' prefers m and m' prefers s

Because the instability cases are both contradicted, ∴ my algorithm always creates stable matching!

Algorithm to Ensure Stability:

* Iterate through the list of all n students
  * Let's say s2 has m4 as its top priority AND m4 has space within their med school, m4 takes s2
  * If m4 doesn't have space & another student, say s1, also wants to commit to m4, m4 will check its list
    * If s1 is higher in preference than s2, m4 kicks s2 and takes s1
    * If s2 is higher, s1 keeps searching down its list
  * If s2 gets kicked it goes down its list to find the next highest preference

Stopping criteria: EVERY student runs their list dry OR all open positions get filled (implying there are school-less students)

Disprove CASE 1 by contradiction:

* In case 1, s' had to propose to m at some point

s' proposes to m

YES / NO

YES: s' matches if m has an open role OR if it kicks a lower-ranked student (for students)

CONTRADICTED!

NO: s' gets rejected b/c m has a better choice, but s' is ranked higher than s...

∴ Case 1 is always CONTRADICTED

Disprove CASE 2 by contradiction: ∴ Case 2 is always contradicted!

* We must ask if s' proposed to m at some point

NO ⟶ then s' must've preferred m', but this is NOT TRUE! (m' > m) CONTRADICTED!

YES ⟶ then at some point m rejected s' for s'', then rejected s'' for s, so s'' > s' and s > s'', meaning s > s', but that's a CONTRADICTION since m prefers s' more!

Both are contradicted!

3)

ships — ports

n ships & n ports

$s1$, $s2$, $s3$, ⋮, s_n — $p1$, $p2$, $p3$, ⋮, p_n

s_n → (schedule) — m days

$m > n$

* NO 2 ships in same port in 1 day!

GOAL: Find a truncation so that (*) still holds

FOR EACH SHIP

<u>Possible solution:</u>
- Have each ship rank ports by chronological order of visits
- Have each port rank in opposite chronological order that ships visit them

ships (order)	ports (reversed order)
s1	p1
s2 "	p2 "
s3 "	p3 "
⋮	⋮
s_n "	p_n "

Can think of this as variation of stable match

Use same algorithm:

- Iterate through all n ships
  - Let's say p4 is s2's earliest port. if p4 has no one matches yet it'll take s2
  - If p4 is already matched and let's say s1 requests p4, it'll check
    - If s1 is higher (later in chronological order), p4 will kick s2 and match s1
    - If vice versa, s1 keeps searching
  - If s2 got kicked, it'll go down its list

∴ a stable match and therefore, a truncation for EVERY SHIP should exist!

Let's define a STABLE MATCH as ships having an acceptable arrangement of stop ports.

<u>Prove by Contradiction:</u>

- If arrangement doesn't work, it violates the truncating principle that another ship gets to a certain port after a ship has already stopped there

CONTRADICTED

Let's say:

$s_1 \leftrightarrows p_1 \leftarrow s_2$

- ship 2 stops at p_1
- BUT both s_1 and p_1 "prefer" each other over s_2

s_1 had to ask p_1 to stop there at one point

Both get CONTRADICTED

No ⟶ Then p_1 must've preferred s_2 but that's false!

Yes ⟶ Then p_1 must've dumped s_1 for s_3, and then dumped s_3 for s_2

So $\Rightarrow$ $s_3 > s_1$ and $s_2 > s_3$ meaning $s_2 > s_1$, but this isn't true!

4)

$g_1(n) = 2^{\sqrt{\log n}}$

$g_2(n) = 2^n$

$g_3(n) = n(\log n)^3$

$g_4(n) = n^{\frac{4}{3}}$

$g_5(n) = n^{\log n}$

$g_6(n) = 2^{2^n}$

$g_7(n) = 2^{n^2}$

$g_7 > g_2$ because if you $\log_2()$ both functions n^2 grows faster than n

$g_5 > g_3$ because polynomials grow slower than exponential growth

$g_4 > g_3$ because divide both sides by n, get $(\log n)^3$ and $n^{\frac{1}{3}}$ => $\log n$ and $n^{\frac{1}{4}}$, exponentials grow faster

$g_2 > g_4$ bc exponentials grow faster than polynomials

$g_6 > g_7$ bc exponentials grow faster than polynomials (if you $\log_2()$ both g_6 and g_7)

$g_5 > g_1$: take $\log_2()$ of both sides, get g_1 is $\sqrt{\log n}$ and g_5 is $\log_2(n^{\log n}) = \log n \cdot \log n$, substitute in x for $\log n$ => g_1 is $x^{\frac{1}{2}}$ and g_5 is x^2, so g_5 grows faster

We can establish: $g_6 > \underbrace{g_7 > g_2 > g_4}_{g_5 \text{ is here somewhere}} > g_3$

We have to compare g_3 and g_1 and g_5 overall

$g_3 > g_1$: take $\log_2()$ of both sides and get $g_1 : \sqrt{\log n}$ and $g_3 : \log(n(\log n)^3) = \log n + \log((\log n)^3)$
$\log n$ will be larger than $\sqrt{\log n}$ in the long run and we aren't even accounting for the 2nd term

$g_7 > g_5$: 2^{n^2} grows far too fast for even a super-polynomial like $n^{\log n}$ to catch up

$g_2 > g_5$: 2^n grows faster than superpolynomial growth

$g_5 > g_4$: superpolynomial growth grows faster than polynomial, divide by n, $g_4 : n^{\frac{1}{3}}$ and $g_5 : n^{\log n - 1}$, take $\log_n()$ of each side and get $g_4 : \frac{1}{3}$ and $g_5 : \log n - 1$, so g_5 is bigger

$\boxed{\text{FINAL ANSWER: } g_6 > g_7 > g_2 > g_5 > g_4 > g_3 > g_1}$

$\boxed{\text{Ascending order: } g_1 < g_3 < g_4 < g_5 < g_2 < g_7 < g_6}$

5)

a) BASE CASE: $p(1)$ is the base, plug into

$\frac{n(n+1)}{2}$. $p(1) = \frac{1(1+1)}{2} = 1$ ✓

ASSUME: $p(n) = \frac{n(n+1)}{2}$

PROVE: $p(n+1) = \frac{(n+1)((n+1)+1)}{2}$

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

$1+2+3+\dots+n+(n+1)$

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

$= (n+1)(\frac{n+2}{2}) \boxed{= \frac{(n+1)(n+2)}{2}}$ ✓

$p(n) = 1^2 + 2^2 + \dots + n^2$

b)

$p(n)$:	1^2	$1+2^2$	$1^2+2^2+3^2$	$1^2+2^2+3^2+4^2$	$1^2+2^2+3^2+4^2+5^2$	$\dots+6^2$	$\dots+7^2$
Sum:	1	5	14	30	55	91	140
Δ Sum:	+0 1	+2 3	+8 6	+20 10	+40 15	+70 21	+182 28

2   6   12   20   30   42

$2+6=8$ $2+6+12=20$ ...

Assumed Formula: $p(n) = \frac{n(n+1)}{2} + \frac{n(n+1)(n-1)}{3} = \frac{n(n+1)(2n+1)}{6}$

BASE CASE: $p(1) = \frac{1(1+1)(2 \cdot 1+1)}{6} = 1$ ✓

ASSUME: $p(n) = \frac{n(n+1)(2n+1)}{6}$

PROVE: $p(n+1) = \frac{(n+1)(n+1+1)(2n+2+1)}{6} = \frac{(n+1)(n+2)(2n+3)}{6}$

$1^2+2^2+3^2+\dots+n^2+(n+1)^2 = \frac{n(n+1)(2n+1)}{6} + (n+1)^2$

$= \frac{6(n+1)^2 + n(n+1)(2n+1)}{6} = \frac{6n^2+12n+6+n(n+1)(2n+1)}{6} = \frac{6n^2+12n+6+2n^3+3n^2+n}{6}$

$= \frac{2n^3+9n^2+13n+6}{6} \boxed{= \frac{(n+1)(n+2)(2n+3)}{6}}$ ✓

6) Ex:

$A = [0 \ 0 \ 5 \ 50 \ 1]$ goal: find smallest element w/ min. frequency

$\underbrace{\quad}_{N \text{ elements}}$

(5 in this example)

- If $N=0$, return nothing. If $N=1$, return $A[0]$ → $O(n)$ time
- Iterate through the entirety of A, while populating hashmap H with the key being the element number and value being how many times it shows up in array A
- Define 2 variables: minElement and minFrequency as INT_MAX
- Iterate through H → $O(n)$ worst possible time
  - Compare current value to minFrequency
    - If current value smaller, set minElement to current key and minFrequency to current value
    - If current value = minFrequency, set minElement to the smaller of minElement and current key
    - Else, continue
- Return minElement

$$\boxed{T(n) = O(n)}$$

You iterate at max. twice through all N elements. 2N is $O(n)$ time