Fulati Aizihaer **Discussion:** 1F

Time: 2:00pm - 3:50pm, Friday

TA name: Parshan

Exercise 3, Page 22

- Rule:
  - Each show has a fixed rating
  - No two TV shows have exactly the same rating
- Stability Rules:
  1. (S,T) is stable if there isn't any other pair where either network can win more time slots by changing time slots for TV shows.
  2. There is no (S', T) pair such that A can win more time slots for either network.
  3. There is no (S, T') pair such that B can win more time slots for either network.

SOLUTION: ---

(b) Proof by contradiction: For every set of TV shows and ratings, there is not always a stable pair of schedules. Give an example of a set of TV shows and associated ratings for which there is no stable pair of schedules.

- Suppose each network has n=2 shows, totalling 4 TV shows.
- A1 with rating of 1, A2 with rating of 3, B1 with rating of 2, B2 with rating of 4
- If we initially have the pairs (A1, B1) and (A2, B2) with ratings of (1, 2) and (3, 4)
  - Network B gets both the showtime slots and Network A gets none
  - Network A will have the ability to flip around A1 and A2 making the pair (A2, B1) and (A1, B2), to win 1 time slot, which goes against our rule making it unstable.
  - Additionally, after we flip A1 and A2, Network B will have the ability to flip B1 and B2 to make the original pair to win back the slot it lost, which goes against our rule making it unstable.
- Therefore, (A1, B1) and (A2, B2) with ratings of (1, 2) and (3, 4) will never have a stable pair of schedules since either network will have the ability to unilaterally change its own schedule to win more time slots.

Fulati Aizihaer **Discussion:** 1F

Time: 2:00pm - 3:50pm, Friday

TA name: Parshan

Exercise 4, on Page 22

Question: Show that there is always a stable assignment of students to hospitals, and give an algorithm to find one.

SOLUTION: ---

Proof by contradiction: There is not always a stable assignment of students to hospitals:

First Type:

- Suppose there are students **s** and **s'**, and a hospital **h**, so that **s** is assigned to **h**, and **s'** is assigned to **no hospital**, and **h** prefers **s'** to **s**.
- When the algorithm terminates, it means' applied to all the **hospitals** in its preference.
- If **h** was empty and accepted **s'** first, it means that **s** cannot replace **s'** since **h** prefers **s'** to **s**.
- And if **h** already had accepted **s** first, it means that **s'** would have replaced **s** if **s'** ranked higher on **h**'s preference list compared to **s.**
- Since this is not the case, there will always be a stable assignment of students to hospitals

Second Type:

- Suppose there are students **s** and **s'**, and hospitals **h** and **h'**, so that **s** is assigned to **h**, and **s'** is assigned to **h'**, and **h** prefers **s'** to **s** and **s'** prefers **h** to **h'**.
- Which means **s'** applied to **h** before applying to **h'** since it prefers **h** to **h'**.
- If **h** was empty and accepted **s'** first, it means that **s** cannot replace **s'** since **h** prefers **s'** to **s**.
- And if **h** already had accepted **s** first, it means that **s'** would have replaced **s** if **h** really preferred **s'** to **s**.
- Since at the end **s** is assigned to **h** and **s'** is assigned to **h'**, we can see that **h** prefers **s'** to **s** and **s'** prefers **h** to **h'** statement is false.
- This means that there will always be a stable assignment of students to hospitals

Fulati Aizihaer **Discussion:** 1F

Time: 2:00pm - 3:50pm, Friday

TA name: Parshan

Algorithm:

- While there is a **s** that hasn't been assigned to a hospital and hasn't asked all the hospitals on its preference list:
- **s** will ask to get assigned to the **h** that is on the top of their preference list.
- If the **h** is not full, then the **h** will accept **s**, and **s** will crossout **h** from its preference list.
- If the **h** is full, check if **s** is ranked higher on **h**'s preference list compared to **s'**,who is already assigned at **h** and is the lowest ranked assigned student.
    - If **s** is ranked higher than **s'** in **h**'s preference list, then remove **s'** and assign **s** into **h**.
        - Then sort all the students assigned in **h** based on its preference
    - If **s** is ranked lower than **s'**, then **s** will be rejected and cross **h** out from its preference list

TIME COMPLEXITY:

- In the while loop, worst case is that it runs for O(n * m) where n is the number of students and m is the total preference of each student. Best case is O(n) if all students get their top preference in one go
- Assignment of students to the hospitals, it can be done in O(1) if there is available space in the hospital that the student applies to. However, if there is no space in the hospital, then it can take O(n) time.
- If the hospital is full, then comparing the two student's ranking in the hospital's preference list will be in O(n) time, where n is the number of students.
- Storing the student in the hospital's assignment list can take O(nlogn) based on the algorithm used for sorting

Fulati Aizihaer **Discussion:** 1F

Time: 2:00pm - 3:50pm, Friday

TA name: Parshan

Exercise 6 on page 25

Question: Given the schedule for each ship, find a truncation of each so that condition continues to hold: no two ships are ever in the same port on the same day.

Show that such a set of truncations can always be found, and give an algorithm to find them.

SOLUTION: --

Proof: Show that such a set of truncations can always be found

- If a ship is docked at a port and is the port's top preference, then it gets truncated for the rest of the month. Which makes sure that each ship will get truncated at a port since there are **n** ships and **n** ports.
- If a ship is docked at a port and is not the port's top preference, then it doesn't get truncated at that port. Algorithm below also makes sure that there is only one ship allowed at each port and that if there is a second ship going to a port that already has a ship docked, the second ship will be put on hold at the sea for the day so that the first ship can leave the next day.
- Once all ships are docked on a port, the algorithm will truncate all the ships, making sure that all the ships are truncated at some point during the month.

Fulati Aizihaer **Discussion:** 1F

Time: 2:00pm - 3:50pm, Friday

TA name: Parshan

Algorithm:

- Create a hashtable for the ships and its schedule based on the ship's schedule (which port its going to first)
- Create a hashtable for the ports and its preference based on reverse order of which ships docks at the port (top choice is the last ship that docks)
- Initialize m as the days in a month, and set m = 1 to represent the first day of the month
- While **m** is less than or equal to the last day of the month:
    - If there is a ship **s** that is not currently at a port:
    - **s** will be sent to the port **p** that is first on its schedule:
        - If **p** is empty then **s** will dock at **p**, and **p** will be crossed out from **s**'s schedule
            - If **s** is the first preference of **p**, then ship **s** gets truncated for the rest of the month there since **s** is the last ship that will be docking at **p**
        - If **p** is not empty and has **s'** docked already at **p**, we keep **s** at the sea for the day
    - If all the ships are docked at a port and there are no ships that are not currently at a port, then truncate all the ships and terminate the algorithm.
    - Go to next day on the schedule
    - If any ship that got assigned to a port was not the first preference of that port, then remove those ships from those ports

TIME COMPLEXITY:

- In the while loop, it can run the best case of O(n) time (n = total ships) where all the ships get assigned to a port on the first day while fulfilling the condition. And the worst case of O(n * m) time to go through the full schedule of each ship.
- Assignment of ships to the ports, it can be done in O(1) if the port is empty for the first preference of the ship that goes there. However, if the port is not empty, then it can take O(n) time.
- If any ship that got assigned to a port wasn't the first preference of that port, then removing those ships can take O(n) time.

Fulati Aizihaer **Discussion:** 1F

Time: 2:00pm - 3:50pm, Friday

TA name: Parshan

4. Exercise 4 on page 67

$g_1(n) = 2^{\sqrt{logn}}$ **<** $g_3(n) = n(logn)^3$ **<** $g_4(n) = n^{4/3}$ **<** $g_5(n) = n^{logn}$ **<** $g_2(n) = 2^n$ **<** $g_7(n) = 2^{n^2}$ **<** $g_6(n) = 2^{2^n}$

Analyze time complexity:

- $2^{\sqrt{logn}}$ grows much slower compared to n. Additionally, exponent $2^{\sqrt{logn}}$ grows slower than the exponent $(logn)^3$.
- $(logn)^3$ grows slower than $n^{1/3}$, so $n(logn)^3$ grows slower than $n^{4/3}$
- 4/3 is a constant, so O(4/3) is faster than O(log n), so $n^{4/3}$ grows slower than n^{logn}
- O(n^2) (Quadratic) is faster than O(2^n) (Exponential) , so n^{logn} grows slower than 2^n
- O(n) is faster than O(n^2), so 2^n grows slower than 2^{n^2}
- O(n^2) (Quadratic) is faster than O(2^n) (Exponential), so 2^{n^2} grows slower than 2^{2^n}

Fulati Aizihaer **Discussion:** 1F

Time: 2:00pm - 3:50pm, Friday

TA name: Parshan

5. a). Prove (by induction) that sum of the first n integers (1+2+....+n) is n(n+1)/2

- Assume n = 1; $1 = \frac{1(1+1)}{2}$ → TRUE
- Assume n = k; 1+2+...+k = $\frac{k(k+1)}{2}$ → Assume its TRUE
- Prove n = k+1 is also true
- 1 + 2 + … + k + (k+1) = $\frac{k(k+1)}{2} + (k + 1)$
- 1 + 2 + … + k + (k+1) = $\frac{k(k+1)}{2} + \frac{2(k+1)}{2}$
- 1 + 2 + … + k + (k+1) = $\frac{k(k+1) + 2(k+1)}{2}$
- 1 + 2 + … + k + (k+1) = $\frac{k^2+k + 2k+2}{2}$
- 1 + 2 + … + k + (k+1) = $\frac{k^2+3k+2}{2}$
- 1 + 2 + … + k + (k+1) = $\frac{(k+1)(k+2)}{2}$ → n = k+1 is TRUE
- Therefore, the sum of the first n integers (1+2+....+n) is n(n+1)/2

b). What is 1^3 + 2^3 + 3^3 + ... + n^3 = ? Prove your answer by induction.

- Assume $1^3 + 2^3 + 3^3 +\ldots + n^3 = (\frac{n(n+1)}{2})^2$
- Assume n = 1; $1^3 = (\frac{1(1+1)}{2})^2$ → 1 = 1 → TRUE
- Assume n = k; $1^3 + 2^3 + 3^3 +\ldots + k^3 = (\frac{k(k+1)}{2})^2$ → Assume its TRUE
- Prove n = k+1 is also true
- $1^3 + 2^3 + 3^3 +\ldots + k^3 + (k + 1)^3 = (\frac{k(k+1)}{2})^2 + (k + 1)^3$
- $1^3 + 2^3 + 3^3 +\ldots + k^3 + (k + 1)^3 = \frac{k^2(k+1)^2}{4} + \frac{4(k+1)^3}{4}$
- $1^3 + 2^3 + 3^3 +\ldots + k^3 + (k + 1)^3 = \frac{k^2(k+1)^2+4(k+1)^3}{4}$
- $1^3 + 2^3 + 3^3 +\ldots + k^3 + (k + 1)^3 = \frac{(k+1)^2(k^2+4(k+1))}{4}$
- $1^3 + 2^3 + 3^3 +\ldots + k^3 + (k + 1)^3 = \frac{(k+1)^2(k^2+4k+4)}{4}$
- $1^3 + 2^3 + 3^3 +\ldots + k^3 + (k + 1)^3 = \frac{(k+1)^2(k+2)^2}{4}$
- $1^3 + 2^3 + 3^3 +\ldots + k^3 + (k + 1)^3 = (\frac{(k+1)(k+1+1)}{2})^2$ → n = k+1 is TRUE
- Therefore, the sum of the first n integers $1^3 + 2^3 + 3^3 +\ldots + n^3$ is $(\frac{n(n+1)}{2})^2$

Fulati Aizihaer **Discussion:** 1F

Time: 2:00pm - 3:50pm, Friday

TA name: Parshan

6. Given an array A of size N. The elements of the array consist of positive integers. You have to find the largest element with minimum frequency.

- Store the array values into a hashtable such that **[key, count]**
- Initialize **largestKey** and make it equal to the key value of the first element in the hashtable
- Initialize **minCount** and make it equal to the count value of the first element in the hashtable
- Loop through the hashtable starting from the second element till the last element:
- If the value of current count is less than **minCount**, then replace the value of the **minCount** with current count, and replace the value of **largestKey** with the current key.
- If the value of current count is equal to the minCount, and the value of current key is larger than the value of largestKey, then replace the value of largestKey with value of current key.
- After the iteration ends, return the **largestKey**

TIME COMPLEXITY:

- Looping through the hashtable has the worst case of O(n) time to loop through the n elements.
- Comparing the current count with min count and updating the values take O(1) time
- Comparing the value of current key with largestKey and updating the values take O(1) time