John Jea

1.) Chere are 2n men. In how many ways can they be paired up?

Diagram:

for $n=3$: $\overset{\curvearrowright}{\_\ \_}\ \overset{\curvearrowright}{\_\ \_}\ \overset{\curvearrowright}{\_\ \_}$

$$(2n-1)\cdot(2n-3)\cdot(2n-5)$$

$$(5)\cdot(3)\cdot(1)=15$$

for $n=4$: $\overset{\curvearrowright}{\_\ \_}\ \_\ \_\ \_\ \_\ \_\ \_$

$$(2n-1)\cdot(2n-3)\cdot(2n-5)\cdot(2n-7)$$

$$7\cdot5\cdot3\cdot1=105$$

Code:
```
ans = 1
  for i in [0, n-1]:
      ans *= (2n - (2i + 1))
```

Explanation: Che code above is based on the mathematical diagram which works for all $n \geq 0$. Che explanation behind the expression is:

1. We can line $2 \cdot n$ blank spaces for possible "slots".
2. For the first space, it can be paired up with $(2n-1)$ spaces.
3. For the second space, it can be paired up with $(2n-3)$ spaces.
4. Che series goes on until all spaces are filled.
5. Hence, $(2n-1)\cdot(2n-3)\ldots$

Closed mathematical Expression: $\boxed{\prod_{i=0}^{n-1}(2n-(2i+1))}$

product of terms

② False.
Consider the pair <M, W> in a set of {M, W, W2}

M ranking/preference list:
1. W2
2. W

W ranking/preference list:
1. M

W2 ranking/preference list!
1. W

* In this example, there is a stable pair which is <M, W>. However, W is not the first choice of M which proves that the statement is false.

3. It's not always possible.

b.) given Network A = $\{a_1 : -1, a_2 : 0, a_3 : 2, a_4 : 4\}$
and Network B = $\{b_1 : -5, b_2 : -4, b_3 : 1, b_4 : 3\}$
where $a_1 \dots a_5$ and $b_1 \dots b_5$ are the TV shows.

Consider the time slots:

$$\begin{array}{ll} A: & \underline{a_1}\ \underline{a_2}\ \underline{a_3}\ \underline{a_4} \\ B: & \underline{b_1}\ \underline{b_2}\ \underline{b_3}\ \underline{b_4} \end{array} \Rightarrow \frac{-1}{-5}\ \frac{0}{-4}\ \frac{2}{1}\ \frac{4}{3}$$

A wins all the slots in the schedule.

then B changes its own schedule:

$$\begin{array}{ll} A: & \underline{a_1}\ \underline{a_2}\ \underline{a_3}\ \underline{a_4} \\ B: & \underline{b_2}\ \underline{b_3}\ \underline{b_4}\ \underline{b_1} \end{array} \Rightarrow \frac{-1}{-4}\ \frac{0}{1}\ \frac{2}{3}\ \frac{4}{-5}$$

B now wins 2 slots.

* By proving that a network can change its own schedule and win more time slots, then there isn't always a stable pair of schedules.

④ We can use Gale-Shapley's algorithm (modified) to show that there's always a stable assignment.

* The algorithm starts by going through each hospital's preference list in a sorted order, from most preferable to least.
For every candidate, if they are not already ascigned or their current ascignment is less preferred, then accept that student. Otherwise, reject.

* This will ensure that there is no instability. Some pairs will be broken, so to fill remaining seats, continue the algorithm until all the total seats has been filled.

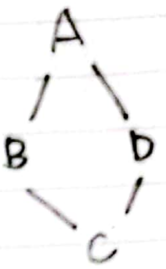
Proof of correctness:

1. By going through a hospital's preference list in order, we ensure that the first type of instability will never happen, as the only way to break the <h, s'> pair is if student s' is paired up with another hospital which contradicts the first type of Instability.

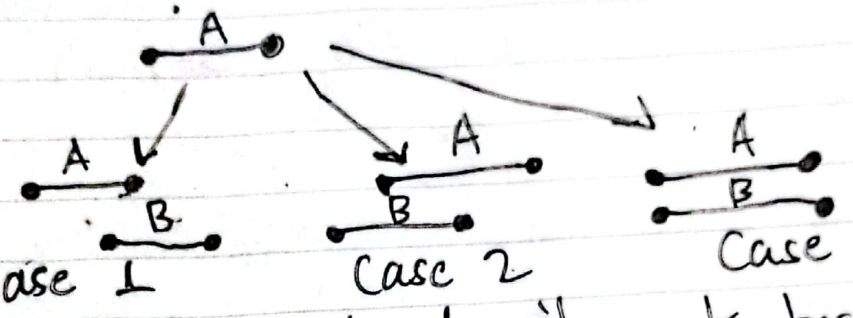
2. The second type of instability is handled when the current assignment for student is evaluated by the preference number.

* thus, the algorithm never produces an unstable result.

59a.) Consider intervals A, B, C, D, we want to show that the conflict graph is impossible.



Let's solve it by cases.



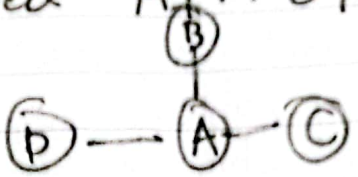
Case 1: the conflict graph doesn't work because interval D would have to conflict with B for it to match the conflict with A, which is impossible in this case.

Case 2: The reasoning in case 1 applies as they're symmetric.

Case 3: It's impossible because for C to conflict with B, it would also have to conflict with A.

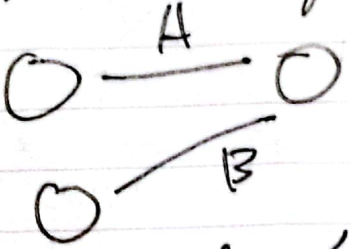
Hence, in all cases, the conflict graph is unachievable.

(5b.) Let's have 4 nodes called A, B, C, D. We'd want to show that

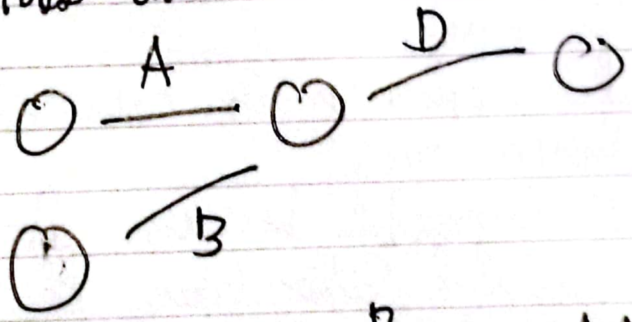


is an invalid conflict graph.

Solve it by case, the diagram starts with



+ this connects an edge <A,B>. Now we want to form an edge either <A,D> or <A,C>, this gives us:



+ this is impossible because B would then result to a connection with D, which is not in the desired conflict graph. Thus, it's impossible

6. $\sqrt{2n}$, $n+10$, $n^2\log n$, $n^{2.5}$, 10^n, 100^n

f_2, f_3, f_6, f_1, f_4, f_5

1. $\sqrt{2n}$ is faster than $n+10$ because $\sqrt{n}$ is faster than linear time.
2. Quadratic time * log is slower than linear time.
3. $n^2 \cdot n^{0.5}$ is slower than $n^2 \cdot \log n$ because $\sqrt{n}$ is slower than $\log(n)$.
4. 10^n is slower than $n^{2.5}$ because it's exponential.
5. $100^n > 10^n$

7. given: $f_1(n) = 2^{\sqrt{\log n}}$

$f_2(n) = n\,(\log n)^3$

$f_3(n) = 2^n$

$f_4(n) = n^{5/3}$

$f_5(n) = n^{\log n}$

$f_6(n) = 2^{2^n}$

$f_7(n) = 2^{n^2}$

Exponential times: $2^{\sqrt{\log n}}$, 2^n, 2^{2^n}, 2^{n^2}

↑ bounded almost a constant, the order is $2^{\sqrt{\log n}}, 2^n, 2^{n^2}, 2^{2^n}$

Other times are: $n(\log n)^3, n^{5/3}, n^{\log n}$

$\Rightarrow n(\log n)^3, n^{5/3}, n^{\log n}$

Final order $\Rightarrow 2^{\sqrt{\log n}}, n(\log n)^3, n^{5/3}, n^{\log n}, 2^n, 2^{n^2}, 2^{2^n}$

or $f_1, f_2, f_4, f_5, f_3, f_7, f_6$

8a .. We can use quick sort to sort it from ascending order. This algorithm will be O(n.logn). We can use a two-pointer approach as follows in O(n) time complexity:

```
Given an array A of size n, and integer T:
    // Store the original index in O(n)
num_index = {n:i for i,n in enumerate(A)]

    // Sort the array in O(nlogn) using quick sort
A.sort()

    // Use two pointers in O(n)
left=0
right=n-1
while left < right:
    s= A[left]+A[right]
    if s < T:
        left +=1
    elif s > T:
        right -=1
    else:
        return [num_index[A[left]],
                num_index[A[right]] ]
return [-1,-1]
```

This algorithm will return the indices of the two numbers that add up to T in O(nlogn) time.

(8b) The bulk portion of the algorithm is the two-painter approach. We will try to prove why this works.

Because of the array's sorted nature, the two pointer approach works because of how we do it, let's prove its by cases:

Case 1: the current sum is equal;
The algorithm returns the two indices that correspond to left and right.

Case 2: the current sum is less than T:
* We move left by 1 which will increase the current sum.

Case 3: the current sum is greater than T:
* We move right by -1 which will decrease the current sum.

* The algorithm works because of two things:
1. left initially contains the smallest number and right initially contains the largest number.
2. By comparing the current sum to target, we're able to make the smallest possible adjustment ensuring that we don't miss any more pairs.

* By narrowing down the search space and making the best decision every state, the algorithm will work properly.