1.

Each hospital needs to fill its slots, each student is available until he/she accepted(temporarily) hospital offer. I will prove by analyzing 2 types of stability.

- First type of instability:

In this case, because h prefer s’ to s, so h sent the offer to s’ but s’ denied, after that h asked s and s accepted. That means s’ is matched with another hospital → contradict with “s’ is assigned to no hospital”.

- Second typed of instability:

In this case, because h prefer s’ to s, so h sent the offer to s’ but s’ denied to join h’ → contradict with “ s’ prefer h to h’ ”, s’ should accept h than h’.

=> The matching must be stable.

**Algorithm:**

initially every hospital and student are free.

Loop while there exists a hospital that has not yet offer to all its candidates with free slot.

Choose a hospital h with 1 free slot.

Choose the highest-rank student s who has not given an offer in list of h, send offer to s.

If s is free then

s accepts the offer(temporarily) and slot of h decreases by 1.

else s accepted another offer from hospital h’ then

if s prefers h to h’ then

s rejects offer of h’ and accepts offer of h

slot of h increases by 1 and slot of h’ decreases by 1

end

end

end loop

Return list

2.

I. (n+a)^b = n^b + a1\*n^(b-1) + a2\*n^(b-2) + …. + a^b = θ(n^b)

II.

T(n) = 2T(n^(1/2))+ log n = 2(2T(n^(1/2^2)) + 1/2\*log n) + log n = 2^2\*T(n^(1/2^2)) + 2log n

= 2^2\*(2T(n^(1/2^3)) + 1/2^2\*log n) + 1/2\*log n) + log n = 2^3\*T(n^(1/2^3)) + 3log n = …..

= 2^K\*T(n^(1/2^K)) + Klog n

n^(1/2^K) = 1 ==> K → infinity ==> T(n) → infinity

III.

T(n) = T(n/2) + log n = T(n/4) + 2\*log n = … = T(n/K) + K\*log n

n/K = 1 ==> n=K

=> T(n) = 1 + n\*log n = θ(n\*log n)

IV.

T(n) = 7T(n/3) + 5√n + log3n = 7T(n/3) + θ(n^(1/2))

7 > 3 ^(1/2) => T(n) = θ(n^(log37))

3.

Given a directed graph G with vertex 1,2…,n in a topology sort.

Choose vertex vi , if vi → vi+1 edge is not existed, then there is no path from vi+1 →vi

==> Graph G is not a semi connected.

In opposite, If there exists an vi → vi+1 edge for every i value, then there is a path vi → vi+1 → … → vn

==> All vertices in the graph after vertex i have a path from i ==> Graph G is semi-connected

**Algorithm:**

Do topology sort in the graph G

Loop, Scan all vertex in graph G

choose vertex i

Check whether there exists edge vi → vi+1 , if not

declare G is not semi connected, exit

End loop

Declare G is semi connected

The running-time is O(V+E) with V is vertices number, E is edge number.

Bonus:

The centroid node is the node at equal distance or at distance less than or equal a half of the vertices of the whole graph.

1. Find the centroid node, about O(n)

2. Count all pairs that their path through centroid, about O(nlog n)

3. Delete the centroid, then repeat step 1 for remain sub-trees until find all paths. Total running time is about O(nlog2 n)