1.

(a). What order does Kruskal's algorithm add edges to the minimum spanning tree?

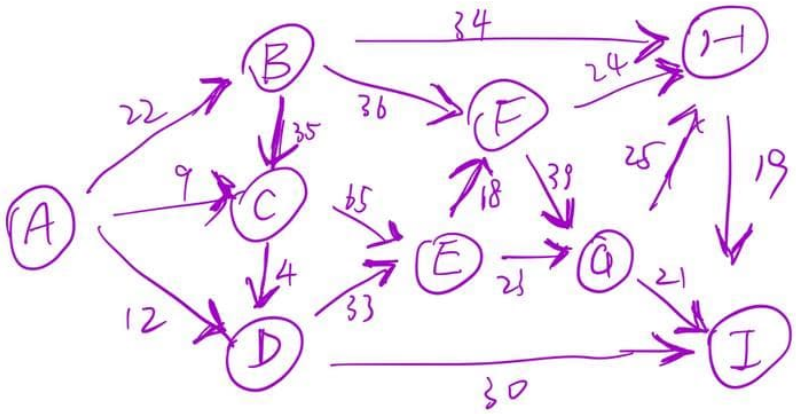
Ans: CD -> AC -> EF -> HI -> GI -> AB -> EG -> DI

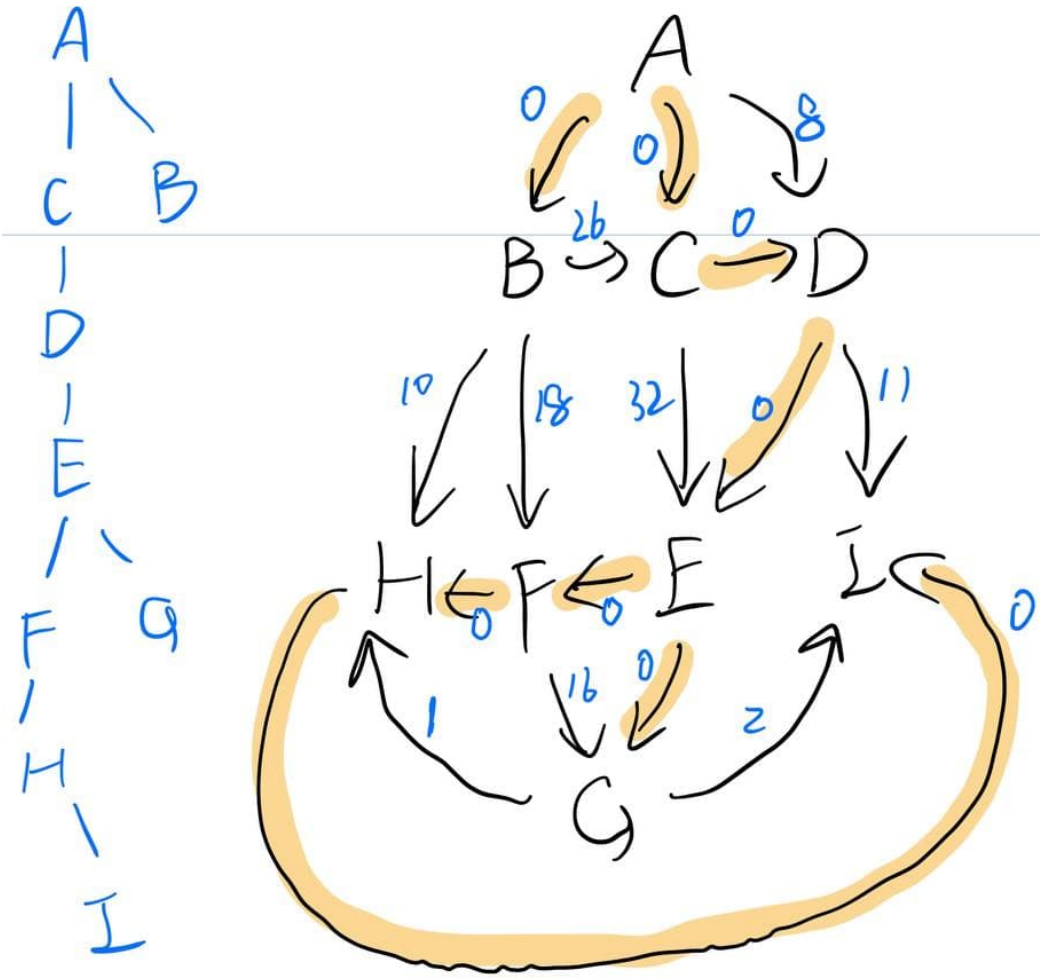
(b). What order does Prim's algorithm add edges to the minimum spanning tree

Ans: AC -> CD -> AB -> DI -> HI -> GI -> EG-> EF

(c). Use the minimum cost arborescence algorithm to find the minimum cost arborescence rooted at A.

Ans: minCost = AC + AB + CD +DE +EG + EF + FH + HI = 9 + 22 + 4 + 33 + 23 + 18 + 24 + 19 = 152.





2.

(a). $\Theta(n^{\log_2 5}) \approx \Theta(n^{2.321})$
(b). $\Theta(n)$
(c). $\Theta(n^3 logn)$

3.

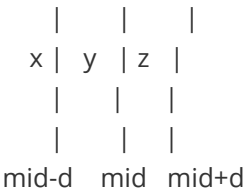
Ans:
General Idea: divide points in the plane into two parts, find the minimum distance in each part and use the smaller one to find possible minimum distance which one point came from part A and another came from part B.

Let me assume my divide & conquer functions to be Find(X, Y), which X is used for dividing part and Y is used for combing part. The result of function find should be current minimal distance in points array X.

- Firstly,
  - I pass two arrays X and Y into my function Find which X represents given points array sorted by x coordinate and Y represents given points array sorted by y coordinate.
- Secondly,
  - I am going to do the divide part which needs at least four points.
  - If I got only 2 or 3 points, I will calculate their minimal distance directly.
  - Else, I will split X into left and right parts according to X's middle point X_{left}, X_{right}. X_{left} means all points in it are on the left side of the middle point. X_{right} is the opposite.
  - then I will traverse Y array and put every point in Y into appropriate side Y_{left}, Y_{right}. Y_{left} means all points in it have smaller or equal x coordinate than X's mid point. Y_{right} is the opposite.
  - Hence, I will call find the Find function twice to get minimal distance of both two sides
    - $d1 = Find(X_{left}, Y_{left})$
    - $d2 = Find(X_{right}, Y_{right})$
- Thirdly,
  - I am going to do the combing part which based on the previous result.
  - current minimal distance $d = min(d1, d2)$
  - By now, I only cover the situation that my closest distance came from either side, but not the situation that one point comes from left side and another comes from right side.
  - Tricky part: If I want to get a smaller distance than d, than my candidate points must be in the range (mid - d, mid + d) and I will call the range as "MiddleBufferedArea". Then I will need to sort everything points in the MiddleBufferedAreaby its y coordinate since my Y array is already sorted when passed inside the function.
  - I am going to traverse every point in Y starting from the bottom point. Then I will check current points and its next 6 points (if have) to see if they have smaller distance than d.
- Lastly,
  - return the minimal distance which is optimal for all three parts: Left, Right, MiddleBufferedArea.

Prove: (combining part):

1. why in range (mid - range, mid + range)?



Contradiction:

Assume one point x is outside the range and other point y is in range (mid - x, mid). If distance(x, y) < d, then my previous minimal d will be distance(x, y) instead of d. which contradicts to our previous suggest d is minimal distances of either side;

Assume one point x is outside the range and other point z is in range (mid - x, mid). No matter where X is, distance(X,Z) is always smaller than d because X,Z min distance = horizontal line > d. Not statisfied.

2. why next 6 points?
*	* .
*	*
*	*

Pf. By contradiction,
If there exist another 7th points, according to pigeonhole principle, there must exist two points that in the same side has a distance smaller than d which contradicts our suggestion that d is minimal distance in either side.

4.

(a) $OPT(i) = max\{OPT(i-1) + S_i, OPT(i-2) + l_{i-1}\}$

(b)
Ans:

- I will construct a one dimensional array called dp with a length of (n + 1)
- dp[i] means maximum profit by the end of day i.
- s_i and l_i means the profit of taking a short job or a long job on day i
- Firstly, intialize my dp array with dp[0] = 0;
- then use the formula above to calculate the current max profit from i = 1 to n which only requires
- when i = 1, i - 2 < 0 so we can set that part to 0 because on day 1 if we take long job we won't gain profit until its next day which is day 2.

(c)
Ans:

- Time Complexity => $\Theta(n)$ since we only need 2n time. (Getting two values and comparing only take constant time for each iteration)
- Space Complexity => $O(n)$ since we only use a array with length of n+1

5.

(a)

- $OPT(i) = max\{OPT(i), OPT(j) + OPT(i-j)\}$ i <- 1 to n/2, j <- 0, i - 1
- $if \quad j == 0, \quad OPT(i-j) = OPT(i) = p_i$

(b)

- Firstly, I construct a one dimension array called dp with a length of n + 1;
- inintialize dp array: dp[0] = 0;
- iterate i from 1 to n, use formula in (a) to get maximun profit.
- in which j <- 1 to n/2 because dp[1] + dp[3] <=> dp[3] + dp[1], we avoid duplicate situation.

(c)
Ans:

- Time Complexity: $\Theta(n^2)$
- Space Complexity: $O(n)$