Assignment-3

2.

Algorithm :

Let $x = (x_1, x_2, \ldots x_n)$

```
function  SigInxCount (x[1 .... n])
    if  n = 1
        return 0;

Let  L := x [1 .... floor(n/2)]
Let  R := x [ floor (n/2) + 1 .... n];

Let  y , a := SigInxCount (L);
Let  z , b := SigInvCount (R);

Let  j = 1 , j = 1 , k = 0

// to count the no of Significant split
   inversing while (j<=length(y) and j<=length(z)
       if (y[i] > 2 * z[j])
           k += length(y) - i + 1;
           j += 1;

       else
           i += 1;
```

-> At each level,
The counting of Significant split inversions & the normal merge-sort process taken O(n) time.
Further,
At each & every level, we break
~~the~~

the problem into two sub problems. The size of each sub problem is $n/2$.

The Recurrence relation is $T(n) = 2T(n/2) + O(n)$

∴ In total, the time complexity is $O(n \log n)$

2-2

Using a Convolution,

Let $a = (q_1, q_2 \cdots q_n)$

Let $b = (n^{-2}, (n-1)^{-2}, \cdots 1/4, 1, 0, -1, -1/4, \cdots -n^{-2})$

For each j, the convolution of a & b will contain an entry of the form.

$$\sum_{i<j} \frac{q_i}{(j-i)^2} \pm \sum_{i>j} \frac{-q_i}{(j-i)^2}$$

$$\therefore \text{Net force} = (q_j)\left(\sum_{i>j} \frac{q_i}{(j-i)^2} + \sum_{i>j} \frac{-q_i}{(j-i)^2}\right)$$

∴ We can conclude that convolution (a-b) computed in $O(n \log n)$ time.

2-3

In this algorithm, we begin from the root of the tree and see that it is smaller than its two children.

If so, the root is the local min. otherwise, we move to any smaller child.

The algorithm terminate when either
case 1 : we reach a node v that is smaller than both it children.
case 2 : we reach a leaf w;

In case 1, we returns v and in case we return w.

The algorithm performs $O(d) = O(\log n)$ probes of the tree we must now argue that the returned value is a local min.

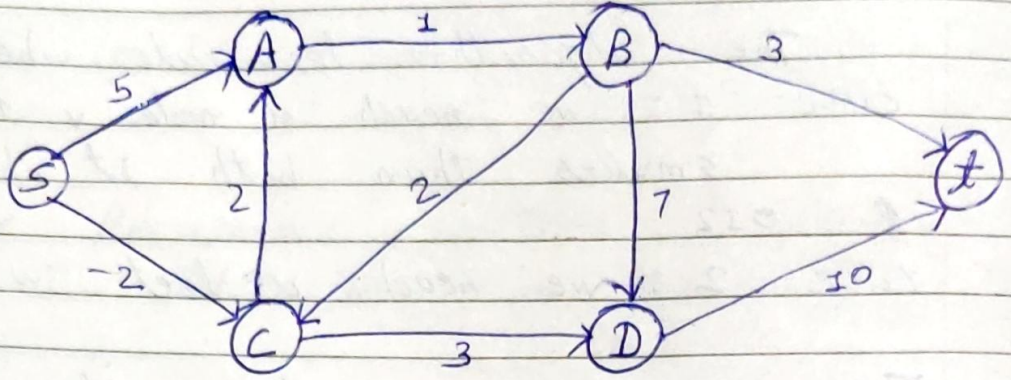
If root is returned, it is a local minimum.

If we terminated in case (1), visa local minimum, because v is smaller than its parent and its two children.

If we terminate in case (2), w is a local minimum, because w is smaller than its parent.

Algorithm local min. in tree T
local min(v)
if $x_v < x_v$ left child and $x_v < x_v$ right child then return v
else if x_v leftchild $< x_v$
return local Min (v, leftchild)
else return local Min (v, right child)
end

Q-4



Using Bellman ford algorithm,

V = 6 nodes

$|V-1| = 5$ iterations (max)

Table

		5	5+1=6	-2	-2+3=1	5+1+3=9
	0	∝	∝	∝	∝	∝
Iteration 1:	S	A	B	C	D	t
		-2+2=0	-2+2+1=1			
Iteration 2:	0	~~5~~	B	-2	1	9
	S	A	B	C	D	t
						-2+2+1+3=4
Iteration 3:	0	0	1	-2	1	~~9~~
	S	A	B	C	D	t
	0	0	1	-2	1	4
	S	A	B	C	D	t

Hence, distances are minimized at 3rd iteration.

To node A : $S \to C \to A$

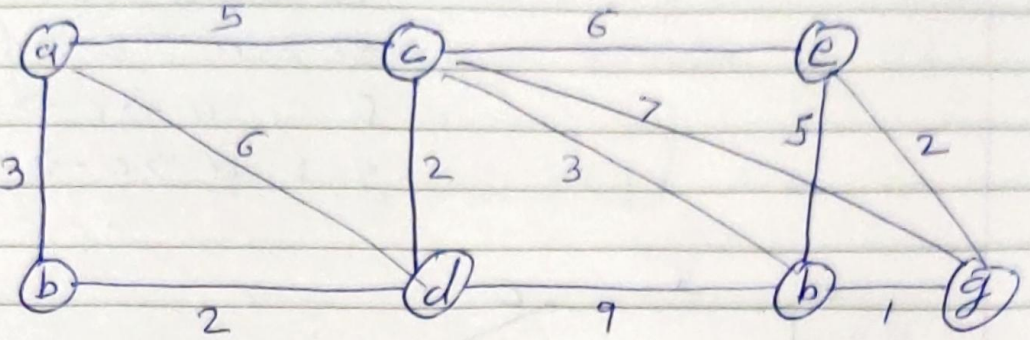
To node B : $S \to C \to A \to B$

To node C : $S \to C$

To node D : $S \to C \to D$

To node t : $S \to C \to A \to B \to t$

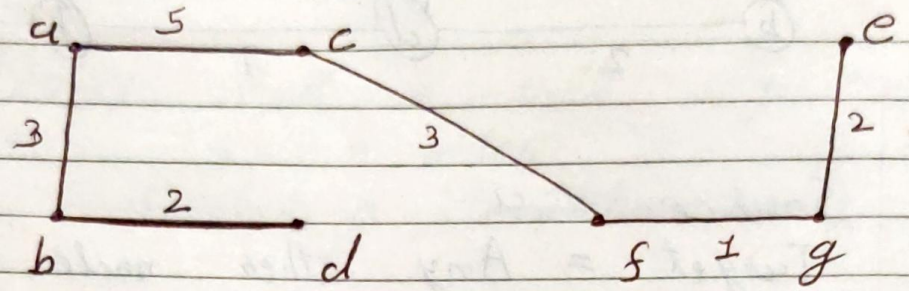
Q-4 Dijkstra's Algorithm,



Source = a
Target = Any other node,

Explored	Unexplored
	a^{∞} b^{∞} c^{∞} d^{∞} e^{∞} f^{∞} g^{∞}
a^{0}	b^{3a} c^{5a} d^{bg} e^{∞} f^{∞} g^{∞}
a^{0} b^{3a}	c^{5a} d^{5b} e^{∞} f^{∞} g^{∞}
a^{0} b^{3a} c^{5a}	d^{5b} e^{11c} f^{8c} g^{12c}
a^{0} b^{3a} c^{5a} d^{5b}	e^{11c} f^{8c} g^{12c}
a^{0} b^{3a} c^{5a} d^{5b} f^{8c} g^{qf}	e^{11c} g^{qf}
a^{0} b^{3a} c^{5a} d^{5b} f^{8c} g^{qf} e^{11g}	e^{11g}

Path from a ⇒ b : a → b = 3
c : a → c = 5
d : a → b → d = 5
e : a → c → f → g → e = 11
f : a → c → f = 8
g : a → c → f → g = 9



2.6

Here a = 1234 and b = 4321

$a = 12 \times 10^2 + 34$ $\quad$ $b = 43 \times 10^2 + 21$

Here n = 4

$a_H = 12$ $\quad$ $b_H = 43$

$a_L = 34$ $\quad$ $b_L = 21$

Now,

$C_0 = a_L * b_L = 34 \times 21 = 714$

$C_1 = a_H * b_H = 12 \times 43 = 516$

$C_2 = (a_H + a_L) * (b_H + b_L) - C_0 - C_1$

$= (12 + 34) * (43 + 21) - 714 - 516$

$= (46 * 64) - 714 - 516$

$= 2944 - 714 - 516$

$= 1714$

Now, $a * b = C_1 * 10^n + C_2 * 10^{n/2} + C_0$

$= 516 \times 10^4 + 1714 \times 10^2 + 714$

$= 5160000 + 171400 + 714$

$\boxed{1234 * 4321 = 5332114}$

2-7

$C(x) = a(x) \cdot b(x)$

$= (1 + 2x + x^2)(1 + 3x + 3x^2 + x^3)$

$= (1 + 3x + 3x^2 + x^3)(2x + 6x^2 + 6x^3 + 2x^4)$
$(x^2 + 3x^3 + 3x^4 + x^5)$

$= 1 + (3x + 2x) + (3x^2 + 6x^2 + x^2) + (x^3 + 6x^3 + 3x^3)$
$+ (2x^4 + 3x^4) + x^5$

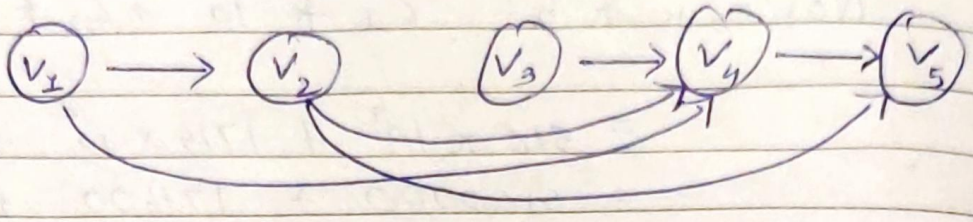
$= 1 + 5x + 10x^2 + 10x^3 + 5x^4 + x^5$

$C = (1, 5, 10, 10, 5, 1)$

2-8

Given Graph $G = (V, E)$
where $V \rightarrow$ no of nodes / vertices
$E \rightarrow$ no. of edges

(a) For the given problem $O(V^2)$ or $O(E)$
good[i] is the length of the longest path v_1 to v_i



good[i] = max(good[j]) + 1 where j < i

v_i , v_j connected $\Rightarrow O(v^2)$

(b) Algorithm :-

```
int path_longest()
{
    for (int i=1; i<=v; ++i)
    {
        good[i] = -1;
        good[i] = 0;
    for (int i=1; i<=v; ++i)
    {
        if (good[i]>=0)
            foreach (v' can be reach from v)
            good[v'] = max(good[v]+1, good[v']);
    }
}
return good[v];
}
```

2-9 (a) The algorithm here do not consider the cost for moving.
It only considers the minimum cost of operation between two cities.

Consider the case when moving cost is greater than the operations cost saved by changing city

Eg : 3 months

NY : 4 2 3

SF : 3 3 2

moving cost is 5.

The algorithm choises SF, NY, SF

But the total cost will be,

$3 + 5 + 2 + 5 + 2 = 17.$

The optimal solution is to just stay in sf for all 3 months.

The total cost will be 8.

(b) Eg : 4 months

NY : 4 2 3 2

SF : 3 3 2 3

Moving cost is 0.

The algo. presented in part (A) works perfectly.

So we have to move every month, which is 3 times

(c) Let NY & SF be arrays of length n, which represent operational cost in n months.

M be the moving cost.

Here, we construct an array cost.

$C[1][1] = NY[1]$

$C[1][2] = SF[1]$

for $j = 2 : n$

$C[j][1] = \min[C[j-1][2] + MC[j-1][1]] + NY[j]$

$C[i][2] = \min(C[i-1][1] + M, C[i-1][2]) + SF[i]$

answer $= \min(C[n][1], C[n][2])$

2-10 ~~The below~~

	Min 1	Min 2	Min 3
A	2	1	1
B	1	10	1

The algo. follows the seq. a_1, a_2, a_3.

i.e. AAA which gives a total value of $2+1+1=4$

whereas the optimal seq. corresponds to BBB giving a total of $1+10+1 = 12$

(b) Let it be denoted by $MAX[A][i-2]$

Similarly, we have the optimal plan till end of minute $(i-1)$ ending with A denoted by $MAX[A][i-1]$.

Similarly, for the optimal plans ending with B, $MAX[B][i-1]$ & $MAX[B][i-2]$ respectively

Based on this, we can find $MAX[A][i]$ & $MAX[B][i]$

If the i^{th} minute involved a MOVE from A to B. or vice versa,
that minute's value will not be added.

If there is no move.
the minute will be added to the total.

$Max[A][i] = max(Max(A)[i-1] + a_i, MAX[B][i-2])$

Similarly, $MAX[B][i] = max(Max[B][i-1] + b_i, MAX[A][i-2])$

The optimal value for i minutes will be the max. of $MAX[A][i]$ & $MAX[B][i]$.

=> Algorithm :-

$MAX[A][0] := 0$, $MAX[B][0] := 0$
$MAX[A][1] := a_i$, $MAX[B][1] := b_i$
Set $i := 2$
while $i <= n$
if $MAX[A][i-1] + a_i < MAX[B][i-2]$
$MAX[A][i] := MAX[B][i-2]$
else
$MAX[A][i] := MAX[A][i-1] + a_i$
end if
Repeat the above,
if conditionds sub. A by B & vice-versa & replacing a_i By b_i.
Increment i
End while

Returns $Max[MAX[A][n], MAX[B][n])$
End Algorithm

-> The above alg. MAX[A] & MAX[B] sums up to two arrays.

-> A fixed no. of operations are performed in each loop iteration
Hence, the alg. has linear time complexity $O(n)$.