Design and Analysis of Algorithm

PYQ-2023 & 2024 & suggestions

1) Describe the characteristics of an algorithm-

⇒ (a) **Finite:-** It should terminate after a finite number of steps.

(b) **unambiguous:-** Every step should clearly specify next step.

(c) **input & output:-** Algorithm take input & Produce output.

(d) **Effectiveness:-** Algorithm should solve the Problem correctly for all possible inputs within a reasonable amount of time.

2) what is the TC of searching an element in a BST in the best & worst case ?

⇒ O(1) → Best case (The element is found at the root node).

⇒ O(n) → The BST is skewed. (height of BST = n)

3)

BST	Best	worst
Insertion:-	O(1)	O(n)
deletion:-	O(1)	O(n)
Height:-	O(logn)	O(n)

Search in Balanced BST or AVL Tree
O(1) & O(logn)
Mostly - O(1)
O(n)

3) what is Recursion & how does it work?

⇒ Recursion is a Programming technique where a function calls itself to solve a Problem by breaking it into smaller subProblems.

How it work :-

Recursion works using two essential Part.

i) Base case :-
- The condition where the function stop calling itself.
- Prevents infinite recursion.

ii) Recursive case :-
- The Part where the function calls itself with a smaller input.

e.g.

```
int fact(int n) {
    if(n==0) return 1;
    else return n * fact(n-1);
}
```

4) How can you compute the max no. of nodes at height 'h', provided there are 'n' nodes in a BT.

⇒ In BT the max no. of nodes at height 'h' is 2^h, if the total no. of nodes is n, then the max possible nodes at height 'h' is $\min(2^h, n)$.

5) what are the conditions required for applying the greedy method?

⇒ A Problem can be solved using greedy method if it satisfies the following two conditions -

a) Greedy choice property :- A globally optimal solution is obtained by making a locally optimal choice at each step.

b) optimal substructure :- An optimal solution of the Problem contains optimal solutions to its subproblems.

) How are the subsequences different from substrings?

⇒

substring	subsequence
a) characters must be contiguous	a) characters need not to be contiguous
b) example: abcde	b) subsequence:
substring: bcd ✓	ace ✓
ace ✗	bd ✓

*) what are the advantages of using Bellman ford algo for shortest path?

⇒ a) detects negative weight cycles (can determine if a graph contain negative cycle).

b) Applicable to both directed & undirected graphs.

---

8) A machine needs a min of 100 sec to sort 1000 names by quick sort. what will be the approximate min time needed (in sec) to sort 100 names.

⇒ $T(n) \propto (n \log n)$

⇒ $\frac{T_1(n)}{T_2(n)} = \frac{n_1(\log n_1)}{n_2(\log n_2)} \Rightarrow \frac{100}{T_2} = \frac{1000 \log 10^3}{100 \log 10^2}$

$\Rightarrow T_2 = \frac{10 \times 2}{3} = 6.67 \text{ sec}$ (Ans)

9) char of algo. (done)

10) state the recurrence relations for the binary search & quick sort algo.

⇒ BS: $T(n) = T(n/2) + O(1)$

Quick sort:

~~best/avg case~~ worst case: $T(n) = T(n-1) + O(n)$

Best case: $T(n) = 2T(n/2) + O(n)$

11) A complete binary tree with 'n' non-leaf nodes contains ___ nodes.

⇒ Total nodes = non-leaf + leaf $= n + (n+1) = 2n+1$.

12) which among the following take least time in worst case:—

		Best	worst
a)	Insertion sort	$O(n)$	$O(n^2)$
b)	quick sort	$O(n \log n)$	$O(n^2)$
c)	Merge sort	$O(n \log n)$	$O(n \log n)$
d)	Heap sort	$O(n \log n)$	$O(n \log n)$

13) The way a card game player arranges his cards as he ~~plays~~ picks them up one by one, is an example of...—

⇒ Insertion sort

14) what is the best & worst case sc of quick sort?

⇒ Best case - $O(\log n)$ ⇒ Partition is balanced, recursion depth ($\log n$)

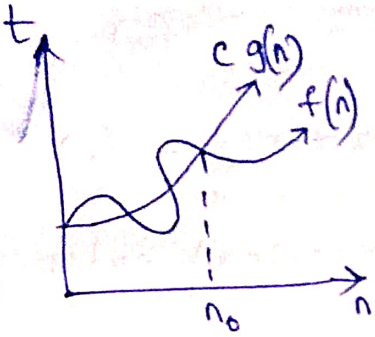
worst " - $O(n)$ ⇒ Partition is highly unbalanced (already sorted array)

Recursion depth → n

Section-B

1) State the Big-O, Big-oh, Big-Ω notations. Explain their graphical interpretation.

⇒ **Big-oh:-** f(n) and g(n) are two TC or SC,



$f(n) = O(g(n))$ if ∃ a positive constant c & n_0 such that

$$f(n) \le c(g(n)) \quad \forall\ n > n_0$$

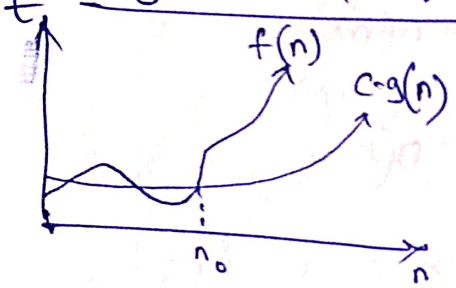
$c, n_0 > 0$.

⇒ e.g. $f(n) = n^r$, $g(n) = n^r \log n$

here $f(n) \le g(n)$

so, $f(n) = O(g(n))$.

⇒ **Big-omega (Ω):-** $f(n) = \Omega(g(n))$



if ∃ a positive constant c & n_0 such that

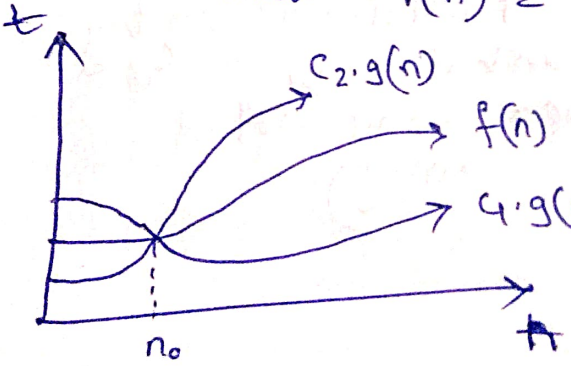
$$f(n) \ge c \cdot g(n)$$ where

$n \ge n_0$ & $c, n_0 > 0$

⇒ **Big-theta (Θ):-** if $f(n) = \Theta(g(n))$ then g(n) is asymptotically tight bound for f(n).

$$c_1(g(n)) \le f(n) \le c_2(g(n))$$ ∀ c_1, c_2 are positive constant & $n_0 > n$.

⇒ if $f(n) = O(g(n))$ & $f(n) = \Omega(g(n))$ then $f(n) = \Theta(g(n))$



2) Explain the Greedy knapsack algorithm.

⇒ Greedy knapsack algorithm
(Fractional knapsack)

The Greedy knapsack algo. is used to solve the Fractional knapsack problem, where fractions of items are allowed to be taken. The objective is to maximize total Profit without exceeding the knapsack's capacity.

Problem definition:-

- Given n-items, each with
  - Profit P_i
  - weight w_i
- knapsack capacity w
- Items can be taken partially.

Greedy:- the greedy approach selects items based on the highest profit-to-weight ratio. $\left(\text{ratio} = \frac{P_i}{w_i}\right)$

Algorithm:-

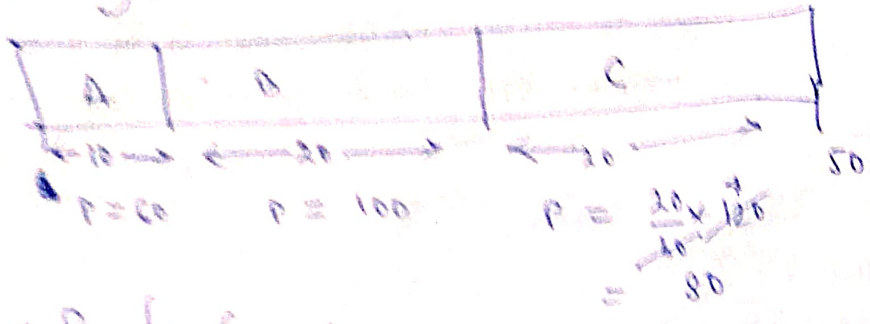
1) compute $\frac{P_i}{w_i}$ for each item.
2) sort items in descending order of P_i/w_i.
3) Initialize remaining capacity = w
   total Profit = 0.
4) for each item:
   - if $w_i \leq$ remaining capacity, take the whole item.
   - Else take the fraction that fits & stop.
5) output: The maximum Profit.

Pseudocode:-

```
sort items by decreasing (Pi/wi)
Profit = 0
capacity = w
for each item i:
    if wi ≤ capacity:
        Profit += Pi
        capacity -= wi
    else:
        Profit += (capacity / wi) * Pi
        break;
```

Item	Profit	weight	P/W
A	60	10	6
B	100	20	5
C	120	30	4

capacity, W = 50



total Profit (max) = 240.

Time complexity = O(n log n)

3) Find the longest common subsequence between
X = {AGiGiTAB} and Y = {GiXTXAYB}

X = "AGiGiTAB"
Y = "GiXTXAYB"

Recurrence Relation:- $L[i][j] = \begin{cases} 0 & ; \text{if } i=0 \text{ or } j=0 \\ L[i-1][j-1]+1 & ; \text{if } X[i]=Y[j] \\ \max(L[i-1][j], L[i][j-1]) & ; \text{if } X[i] \neq Y[j] \end{cases}$

using Dynamic Programming:-

		Y	0 Gi	1 X	2 T	3 X	4 A	5 Y	6 B
	X	0	0	0	0	0	0	0	0
0	A	0	0	0	0	0	1	1	1
1	Gi	0	1	1	1	1	1	1	1
2	Gi	0	1	1	1	1	1	1	1
3	T	0	1	1	2	2	2	2	2
4	A	0	1	1	2	2	3	3	3
5	B	0	1	1	2	2	3	3	4

length of longest common subsequence = 4

& The longest common subsequence is "GTAB"

4) you have an array of size 10. in which 8 numbers are arranged into min-heap. How do you add a new number to the heap? write the pseudo code for the above problem.

⇒ steps to insert a new element in min-heap:-

1. check overflow:-
   if heap size = array size, insertion is not possible.
2. insert the new element at the next free position - index (n+1)
3. Heapify up:-
   - compare the inserted element with its parent.
   - if the element is smaller than its parents, swap them.
   - Repeat until:
     - Heap property is satisfied or
     - The element reaches the root.

⇒ Pseudo code:-

```
INSERT_MIN_HEAP (heap, n, key, maxSize)
if n == maxSize then
    print "Heap overflow"
    return
n = n+1
i = n
heap[i] = key
while i>1 & heap[i] < heap[i/2] do
    swap (heap[i], heap[i/2])
    i = i/2
return n.
```

~~The average case TC of quick sort~~

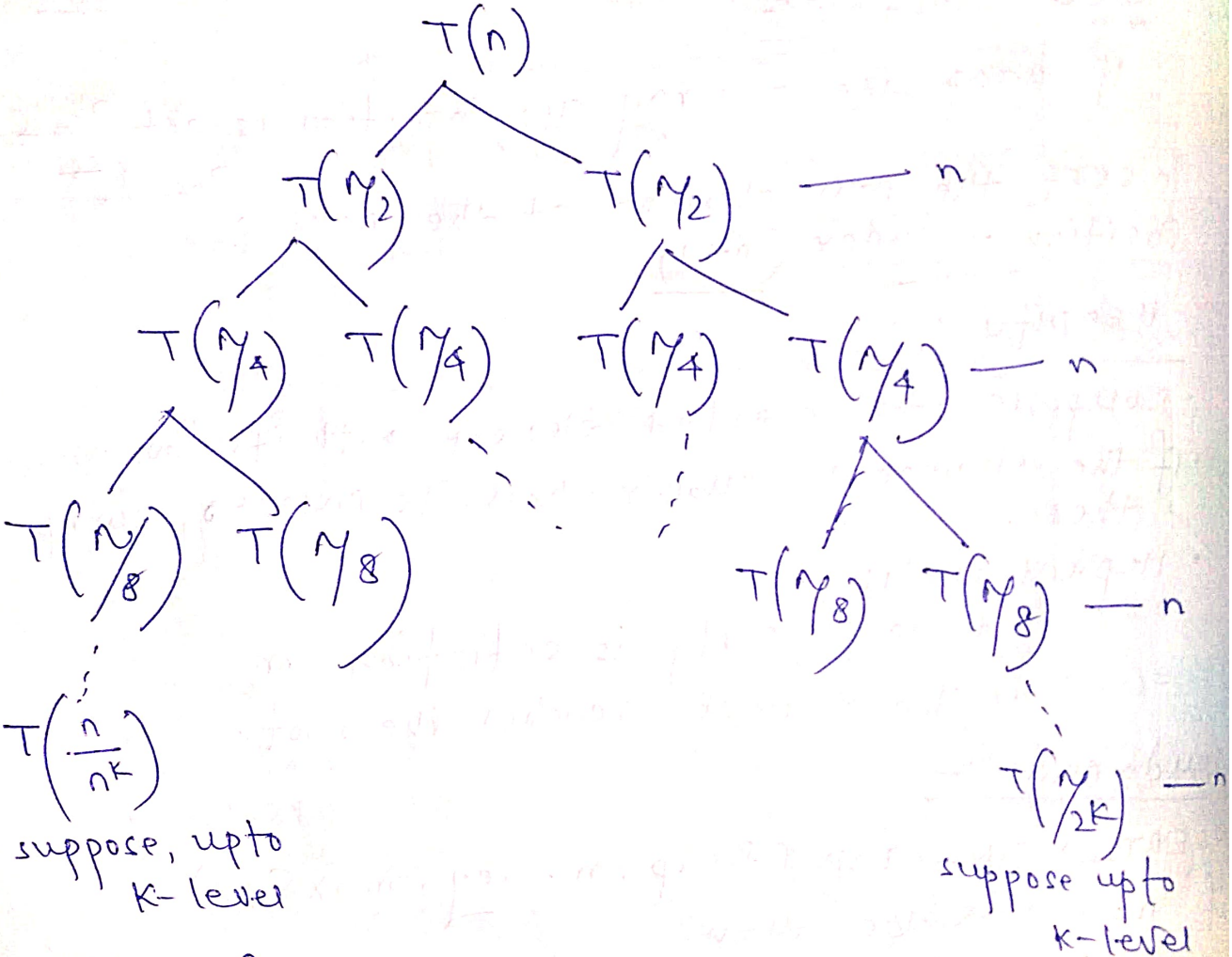
From the recurrence relation, obtain the TC of Quick sort algo through recursion Tree.

⇒ Quick sort average case recurrence relation

$$T(n) = 2T(n/2) + \theta(n), \quad \boxed{T(1) = 1}$$

↓ assuming base case

Recursion Tree :-



so, $\frac{n}{2^k} = 1$

$\Rightarrow 2^k = n$

$\log_2 2^k = \log_2 n$

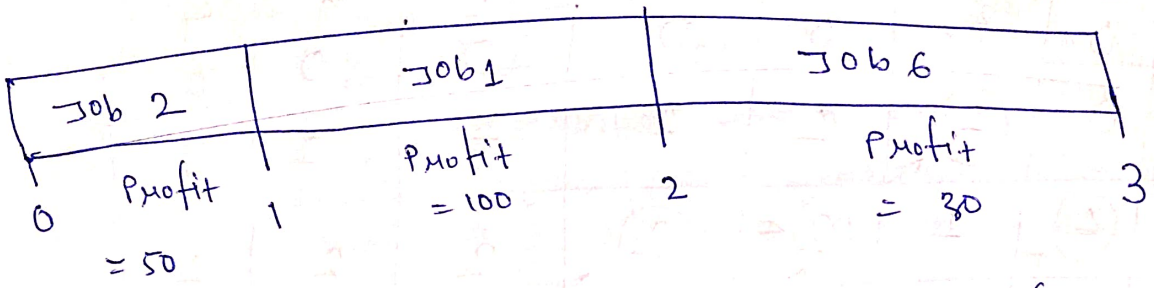
$k = \log_2 n$

so the Time complexity will be $O(n * \log n)$

6)

Job	deadline	Profit
Job 1	2	100
Job 2	1	50
Job 3	3	20
Job 4	2	40
Job 5	1	10
Job 6	3	30

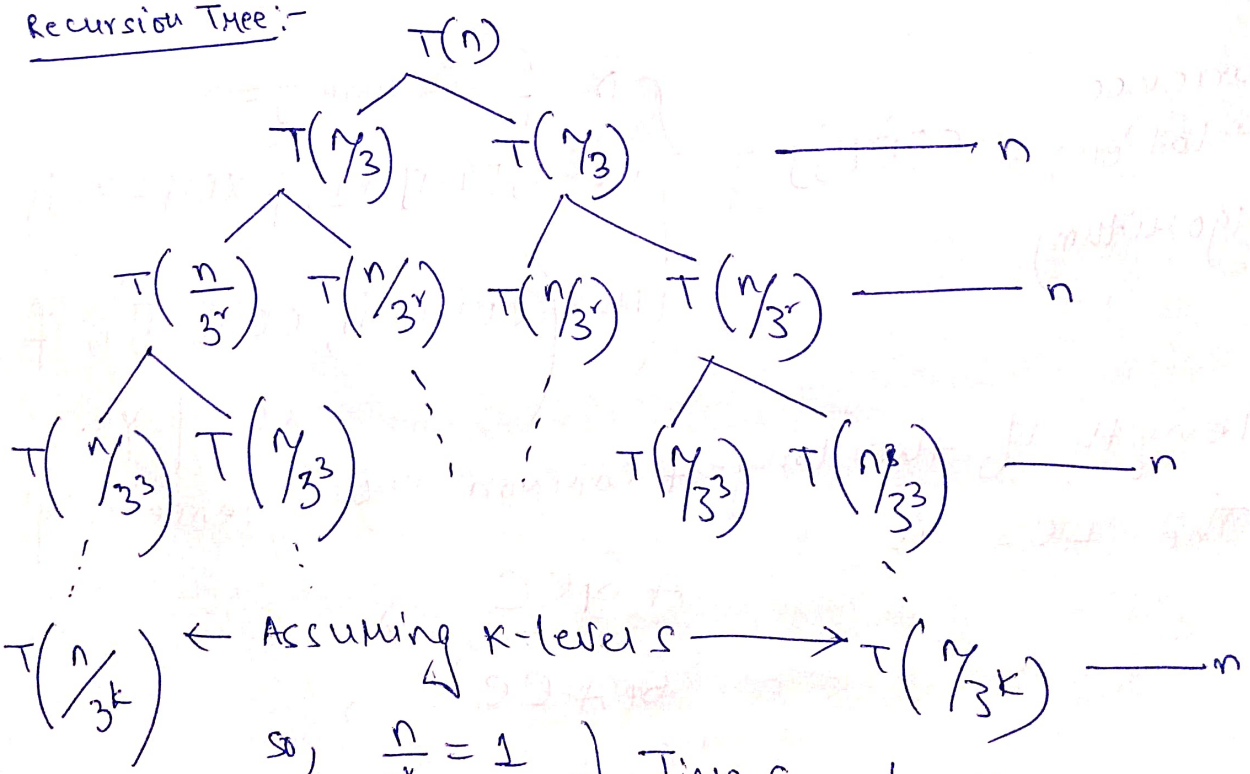
⇒ max deadline = 3



the sequence of job to maximize the Profit (180) is Job 2 → Job 1 → Job 5.

7) $T(n) = 2T(n/3) + n$, assuming base case $T(1) = 1$

Recursion Tree:-



So, $\frac{n}{3^k} = 1$

$\Rightarrow k = \log_3 n$

Time Complexity $= O(n * \log_3 n)$

8) Find the longest common subsequences between $X = \{AGTACGCG\}$ & $Y = \{GTAGTCC\}$ using dp.

⇒ $X =$ "AGTACGCG"

$Y =$ "GTAGTCC"

Dynamic Programming paradigm :-

		X	A	G	A	C	G	C	G
	X	0	0	0	0	0	0	0	0
	G	0	0	1	1	1	1	1	1
	T	0	1	1	2	2	2	2	2
	A	0	1	2	2	2	3	3	3
	G	0	1	2	2	3	3	4	4
	C	0	1	2	2	3	3	4	4
	C	0	1	2	2	3	3	4	4

Recurrence relation: (Algorithm)

$$c[i][j] = \begin{cases} 0 & \text{if } i=0 \text{ or } j=0 \\ c[i-1][j-1]+1, & \text{if } x[i]=y[i] \\ \max(c[i-1][j], c[i][j-1]), & x[i] \neq y[j] \end{cases}$$

⇒ length of the longest common subsequence is 4

⇒ The LCS is 'AGCC',

'GACC',

'GAGC'

9) Explain the Prim's algo implemented through min-heap.

⇒ **Prim's algorithm using min-heap**

Prim's algorithm is greedy algorithm used to find the MST of a connected, weighted, undirected graph. It builds the MST by adding the min weight edge, that connects a vertex in the MST to a vertex outside it.

Greedy strategy:- At each step choose the edge with the min-weight that connects the current MST to a new vertex.

A min-heap is used to efficiently extract the vertex with the min-key value.

Algorithm steps:-

1. Initialize:
   - Key[v] = ∞ for all vertices
   - parent[v] = -1
   - Key[source] = 0
2. Insert all vertices into a min-heap based on key
3. while heap is not empty:
   - Extract vertex 'u' with min-key
   - Mark 'u' as included in MST
   - for every adjacent vertex 'v' of 'u':
     - if 'v' is not in MST & weight(u, v) < Key[v]
       - update Key[v]
       - set parent[v] = u
       - Decrease key in heap
4. The edges (parent[v], v) form the MST.

Pseudocode :-

```
PRIMS (G, source)
    for each vertex 'v' in G:
        Key[v] = ∞
        parent[v] = -1
        inMST = false
    Key[source] = 0
    MinHeap = all vertices with Key
    while MinHeap is not empty:
        u = extractMin (MinHeap)
        inMST[u] = true

        for each 'v' adjacent to u:
            if inMST[v] == false &
                weight(u,v) < key[v]:
                parent[v] = u
                decreaseKey(MinHeap,
                    v, key[v])
```

Time Complexity = $O(E \log V)$

---

10. You have an array of size 15. in which 12 numbers are arranged into a max-heap, if you wanna add a new number into the heap, how to do that? write the pseudo code)

Insertion of a new element

1. **check overflow :-**
   If current heap size = array size, insertion is not possible
2. **Insert at the end :-**
   Place the new element at position 'n+1'
3. **Heapify up :-**
   - compare the inserted element with its parent.
   - if the element > its parent, swap them.
   - continue this process until:
     - max-heap property is satisfied
     - the element reaches the root.

Pseudo code :-

```
INSERT-MAX-HEAP (heap, n, key, max size)
    if (n == max size) then
        print "overflow"
        return

    n = n+1
    i = n
    heap[i] = key
    while (i > 1 && heap[i] > heap[i/2]) do
        swap (heap[i], heap[i/2])
        i = i/2

    return n
```

Time Complexity:

Best case	worst case
O(1)	O(log n)

11) From the recurrence relation, obtain the Time Complexity of the Merge sort algorithm through back substitution Method.

⇒

$$T(n) = 2T(n/2) + \bullet(n) \quad \text{——(i)}$$

$$T(n/2) = 2T(n/2^2) + \bullet(n/2) \quad \text{——(ii)}$$

$$T(n/2^2) = 2T\left(\frac{n}{2^3}\right) + \left(\frac{n}{2^2}\right) \quad \text{——(iii)}$$

Put (ii) in (i)

$$T(n) = 2\left\{2T\left(\frac{n}{2^2}\right) + \frac{n}{2}\right\} + n \quad \text{——(iv)}$$

Put (iii) in (iv)

$$T(n) = 2\left\{2\left\{2T\left(\frac{n}{2^3}\right) + \frac{n}{2^2}\right\} + \frac{n}{2}\right\} + n$$

$\}$ K-times

$$T(n) = 2^k T\left(\frac{n}{2^k}\right) + \underbrace{n + \cdots + n}_{k \cdot n} + n$$

here $\frac{n}{2^k} = 1 \Rightarrow k = \log_2 n$

$(T(1) = 1)$

⇒

$$T(n) = 2^{\log n} + n(\log n)$$

$$= n + n\log n$$

$$T(n) = O(n\log n)$$

12) Explain the algorithm of matrix chain multiplication using dp approach.

⇒ **Matrix chain Multiplication (MCM) using dp :-**

Given a sequence of matrices $A_1, A_2 \dots A_n$ with dimensions, $A_i = P_{i-1} \times P_i$, find the most efficient way to multiply them by deciding the optimal parenthesization so that the total number of scalar multiplication is minimum.

Algorithm :-

1. **Base case:** $m[i][j] = 0 \quad ; \text{ if } i = j$

2. **Recurrence Relation :-**

$$m[i][j] = \min_{i \le k < j} \left(m[i][k] + m[k+1][j] + P_{i-1} * P_k * P_j \right)$$

Algorithm - Bottom up approach :-

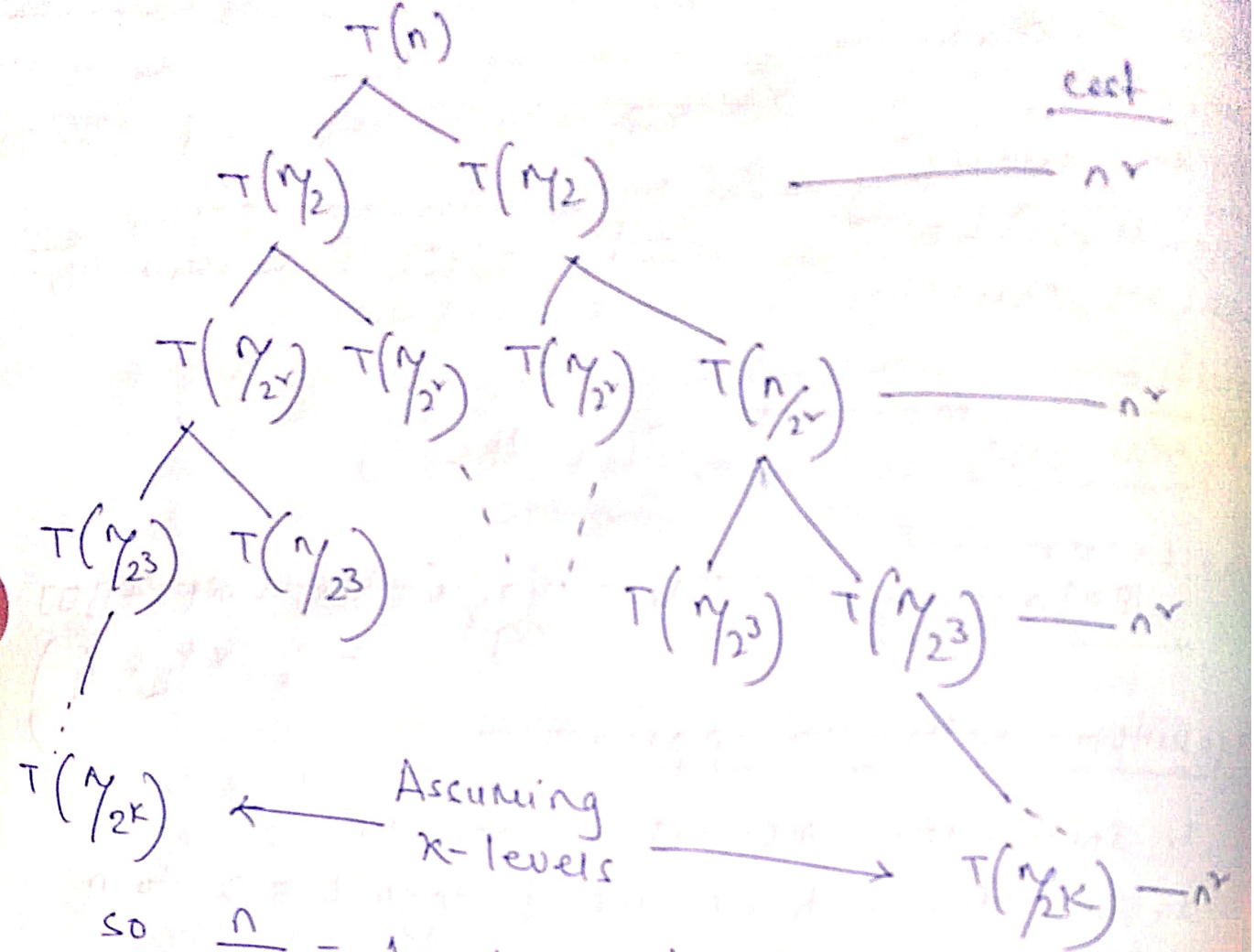
1. Initialize $m[i][j] = 0$ for all i,
2. Consider matrix chains of length $L = 2$ to n
3. For each chain (i, j):
   - Try all possible split positions k
   - compute cost using the recurrence
   - store min-cost in $m[i][j]$
4. Final ans stored in $m[1][n]$

Time Complexity $(TC) = O(n^3)$

Space " $(SC) = O(n^2)$

(3) $T(n) = 2T(n/2) + n^2$

⇒ Assume Base Case $T(1) = 1$



So $\frac{n}{2^k} = 1 \Rightarrow k = \log_2 n$

So Time Complexity $= O(n^2 \log_2 n)$

Section - C

1) a) what is the main idea behind 'divide & conquer' strategy?

⇒ The divide & conquer strategy works by dividing a problem into smaller subproblems of the same type, solving each sub problem independently & then combining their solutions to obtain the final solution.

b) compare & contrast dp & greedy method.

⇒ DP solves a problem by considering all possible sub problems & gurantees an optimal solution.

⇒ Greedy method makes the best local choice at each step, which is faster but does not always gurantee optimality.

- considers all possible choices
- requires extra memory
- e.g. MCM, etc LCS,

- chooses the best immediate choice
- requires little or no extra memory
- e.g. prim's, kruskal etc.

c) Describe the DP algorithm to solve the 0/1 knapsack problem

⇒ Given n-items, each with weight w[i] & profit/value, v[i] & a knapsack with capacity 'w'. Determine the max total value that can be obtained such that the total weight does not exceed 'w', each item can be chosen atmost once.

DP table definition :-

dp[i][j] = max profit using first 'i' items with knapsack capacity j

Base case :-

dp[0][j] = 0

dp[i][0] = 0

Recurrence relation :-

$$dp[i][j] = \begin{cases} dp[i-1][j], & \text{if } w_i > j \\ \max(dp[i-1][j], dp[i-1][j-w_i] + v_i), & \text{if } w_i \le j. \end{cases}$$

exclude the item → $(dp[i-1][j])$

include the item → $(dp[i-1][j-w_i] + v_i)$

Algorithm steps :-

1. create a dp table of size $(n+1) * (w+1)$
2. Initialize 1st row & 1st column as 0
3. Fill the table using recurrence relation.
4. The optimal solution is stored in dp[n][w].

T.C = O(nW)

S.C = O(nW)

2(a) A recursive function computes the nth Fibonacci number, Analyze its space complexity.

⇒ Recursive definition of nth fibonacci

$$F(n) = \begin{cases} 0 ; & n = 1 \\ 1 ; & n = 1 \\ f(n-1) + F(n-2), & n > 1 \end{cases}$$

The max depth of recursion can be

$F(n) \rightarrow F(n-1) \rightarrow F(n-2) \dashrightarrow F(1)$

Hence the max calls can be $O(n)$

so, space complexity $= O(n)$

(b) The matrices M1, M2, M3, M4 have the following dimensions $20*4, 4*15, 15*12, 12*6$. Explain the dp approach to find an optimal parenth-- of the above matrix chain product.

⇒ we can parenthesize 4 matrices ~~using~~ in

$$\frac{(3\times 2)!}{(3+1)!\ 3!} = \frac{6\times 5\times 4!}{4!\times 3\times 2} = 5 \text{ ways.}$$

⇒ Recursive equation for MCM is

$$M[i,j] = \begin{cases} 0 ; & \text{if } i=j \\ \min_{i \le k < j} \{ M[i,k] + M[k+1,j] + p_{i-1} * p_k * p_j \}; & \text{otherwise} \end{cases}$$

$M[14] = M1 * M2 * M3 * M4$

20×4 4×15 15×12 12×6

11 0	22 0	33 0	44 0
12 1200	23 720	34 1080	
13 1680	24 1008		
14 1488			

So, the min scalar multiplications needed to multiply M1 * M2 * M3 * M4 = 1488

1) M[12]
= M1 * M2
20×4 4×15
⇒ 20 × 4 × 15 = 1200

2) M[2 3]
= M2 * M3
4×15 15×12
⇒ 4 × 15 * 12 = 720

3) M[3 4]
= M3 * M4
= 15×12 12×6
⇒ 15 × 12 × 6 = 1080

4) M[1, 3]
= M1 * M2 * M3

M1 * (M2 M3) | (M1 M2) * M3
20×4 4×12 | 20×15 15×12
= 720 + 20 × 4 × 12 | = 1200 + 20 × 15 × 12
= 1680 | = 4800

5) M[2 4]
= M2 * M3 * M4

M2 * (M3 M4) | (M2 M3) * M4
4×15 15×6 | 4×12 12×6
⇒
1080 + 4 * 15 * 6 | 720 + 4 * 12 * 6
= 1440 | = 1008

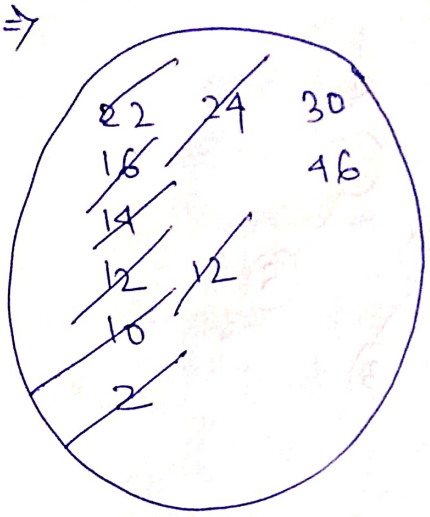
6) M[1 4]
= M1 * M2 * M3 * M4

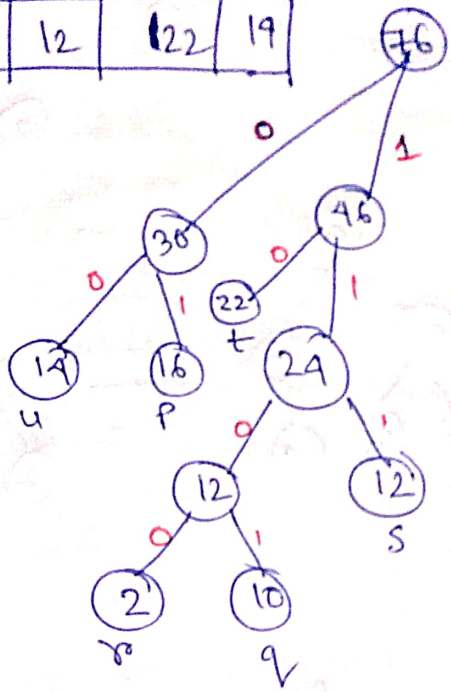
M1 (M2 M3 M4) | (M1 M2) (M3 M4) | (M1 M2 M3) M4
20*4 4*6 | 20×15 15×6 | 20×12
1008 + 20 × 4 × 6 | 1200 + 1080 + ( ) | 1680 + 20 × 12 × 6 ( )
= 1488

3) a) Find the percentage of improvement on variable length code over fixed length coding through Huffman encoding

p	q	r	s	t	u
16	10	2	12	22	14

⇒





suppose the ~~length~~ fixed length is $\lceil \log N \rceil$

where, $N = 6$

so $\lceil \log 6 \rceil = 3$

So, fixed length encoding requires = 228.

Huffman encoding

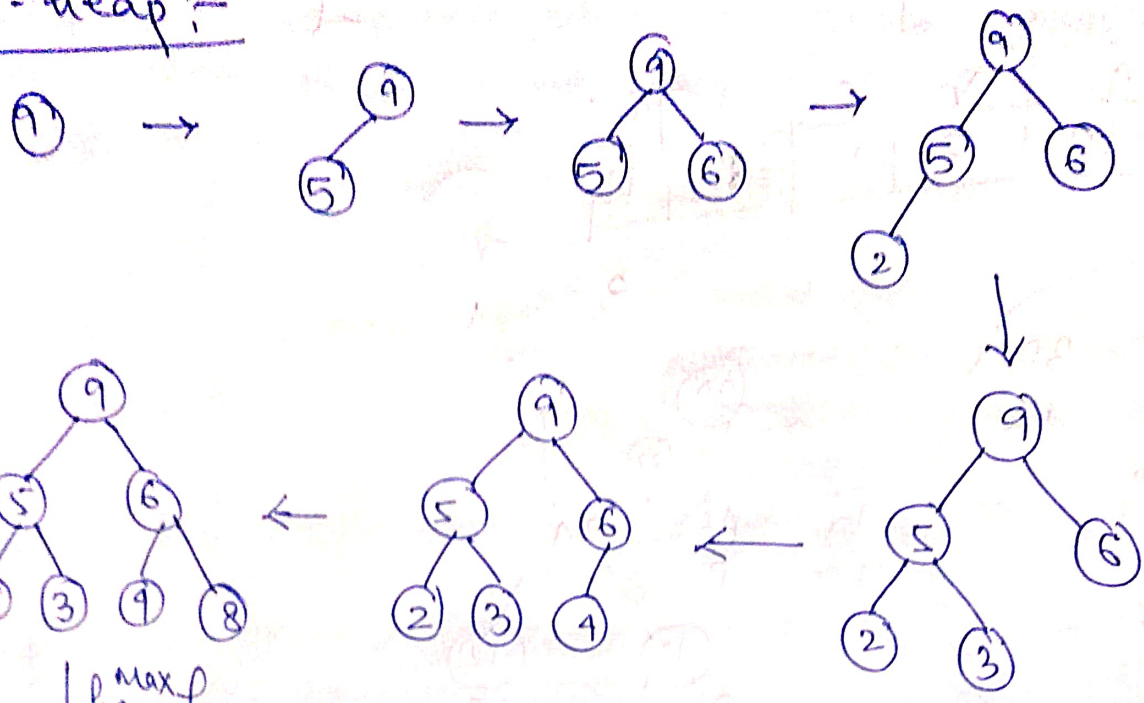
char	bits	total bits
p	01	16 X 2 = 32
q	1101	10 X 4 = 40
r	1100	4 X 2 = 8
s	111	12 X 3 = 36
t	10	22 X 2 = 44
u	00	14 X 2 = 28
		total = 188

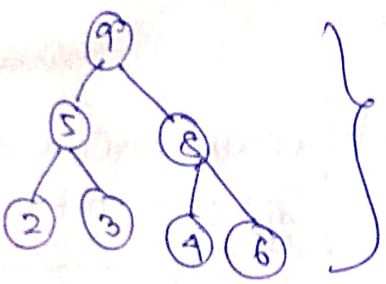
So, Percentage of improvement $= \frac{228 - 188}{228} \times 100 = \frac{40}{228} \times 100$

$= 17.54\ \%$

b) Given the array: [9, 5, 6, 2, 3, 9, 8]

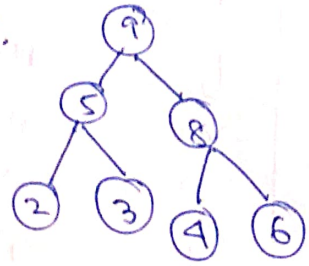
Max-heap :-

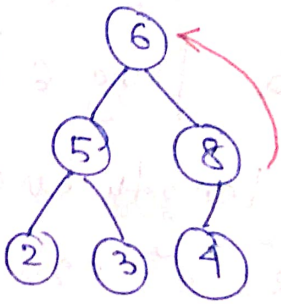


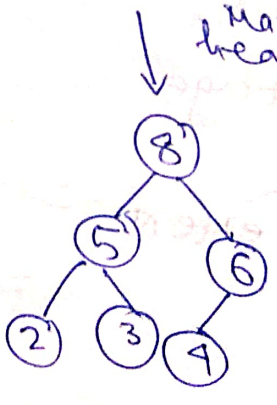


max-heap

Delete max :







resultant heap

4) a) what is spanning tree?

=> A spanning tree of a connected graph is a subgraph that

- includes all the vertices of the graph
- is connected.
- contains no cycles

If a graph has n-vertices, spanning tree has (n-1) edges exactly.

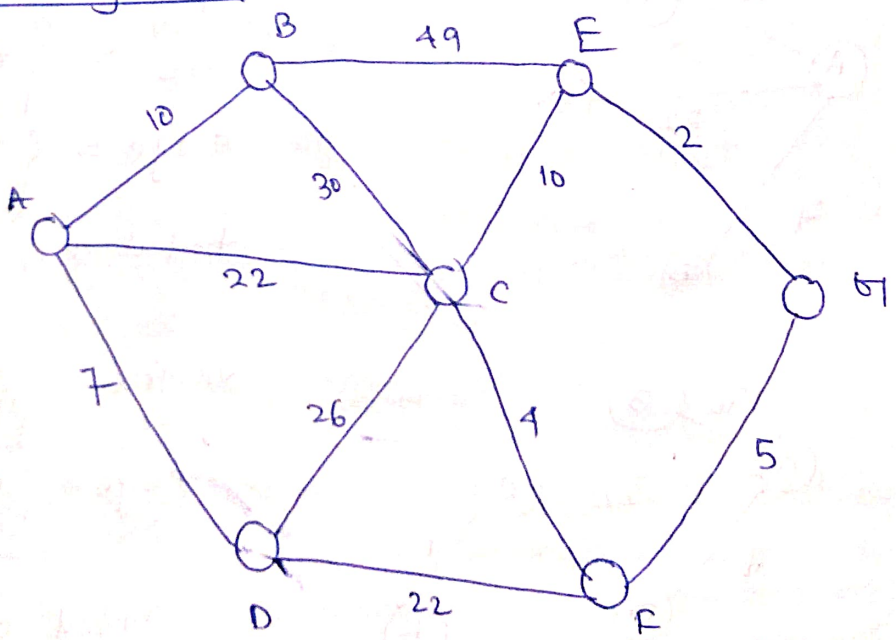
b) Graph with '6' nodes.

=> No. of spanning tree = $6^{6-2} = 6^4 = 1296$

c) what is a min cost spanning tree?

=> A MCST is a ST ~~that~~ of a weighted graph ~~that~~ connects all vertices with the min possible total edge weight.

d) Prim's algo :-



Step-1:- starting from vertex-A (A)

MST vertex set = {A}

MST edge set = ϕ

Total cost = 0

Step-2:- select min edge from A.

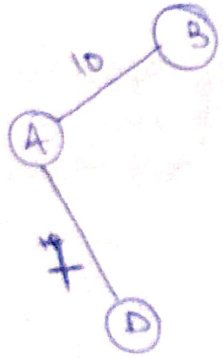


MST vertex set = {A, D}

MST edge set = {A-D}

Total cost = ~~10~~ 7

Step-3:- select Next min edge

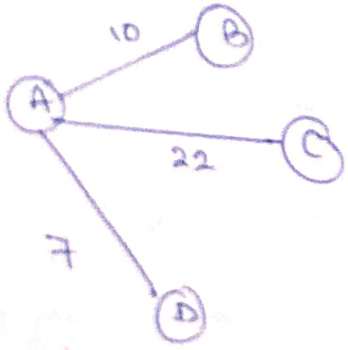


MST vertex set = {A, B, D}

edge set = {A-D, A-B

Total cost = 17

Step-4:

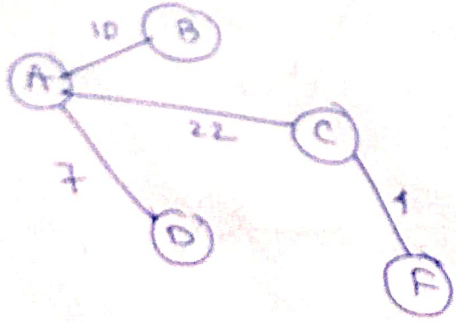


vertex set = {A, B, C, D}

edge = {A-B, A-C
A-D}

total cost = 39

Step-5:-

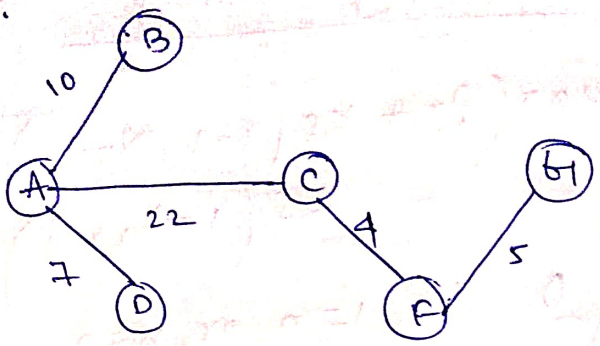


vertex set = {A, B, C, D, F}

edge = {A-B, A-C
A-D, C-F}

Total cost = 93

step-6:-

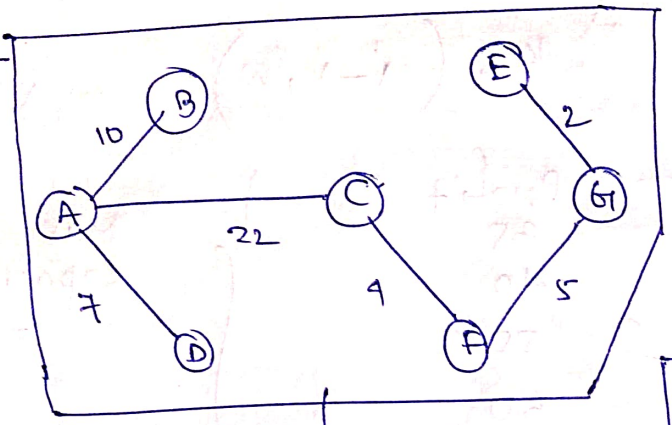


Vertex set = {A, B, C, D, F, G}

edge set = {A-B, A-C, A-D, C-F, F-G}

Total cost = 48

step-7:-



Vertex set = {A, B, C, D, E, F, G}

edge set = {A-B, A-C, A-D, C-F, F-G, G1-E}

Total cost = 50.

Ans.

5)

a) state 0/1 knapsack problem.

=> i) Given n items, each with a weight (w_i) & Profit (P_i)

ii) A knapsack with max capacity 'W' is given.

iii) Each item can be either included or excluded

iv) Items can not be divided.

v) objective is to maximize Profit without exceeding capacity.

b)

obj	obj-1	obj-2	obj-3	obj-4
P	35	40	55	60
W	20	30	40	50

M (total capacity) = 100.

Algorithm (Dynamic Programming)

$$KS(i,w) = \begin{cases} \max(P_i + KS(i-1, w-w_i),\ KS(i-1, w)) & ; w_i \le w \\ 0 & ; i = 0 \text{ or } w = 0 \\ KS(i-1, w) & ; w_i > w \end{cases}$$

single items/objects	wt	Profit
1	20	35
2	30	40
3	40	55
4	50	60

} Max capacity = 100

two items combinations :

	wt	Profit
1+2	50	75
1+3	60	90
1+4	70	95
2+3	70	95
2+4	80	100
3+4	90	115 ✓

three item combinations

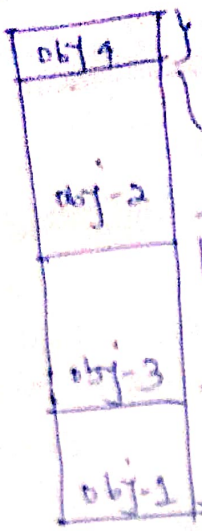
	wt	Profit
1+2+3	90	
1+2+4	100	130
1+3+4	110 ×	135 ✓
2+3+4	120 ×	

four items combinations

	wt	
1+2+3+4	140	×

So, obj 1 + obj 2 + obj 4 & total Profit = 135.

c) If fractional ...



10, Profit = 130 + 1.2 * 10 = 142

30, Profit = 90 + 10 = 130

40, Profit = 35 + 55 = 90

20, Profit = 35

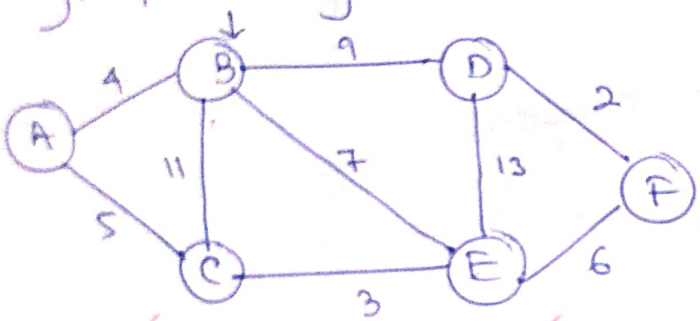
M = 100

object	Profit	wt	Profit/weight
✓ 1	35	20	1.75
✓ 2	40	30	1.33
✓ 3	55	40	1.37
- 4	60	50	1.2

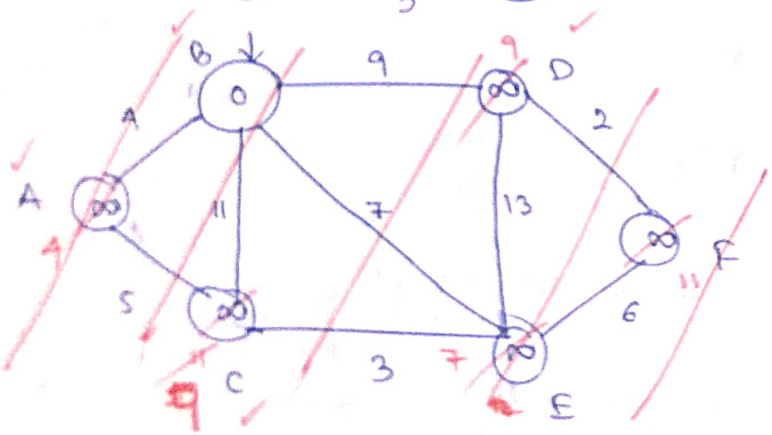
So, total Profit will be = 142 ✓

6) a) Find the shortest path from node-B using dijkstra's algo.

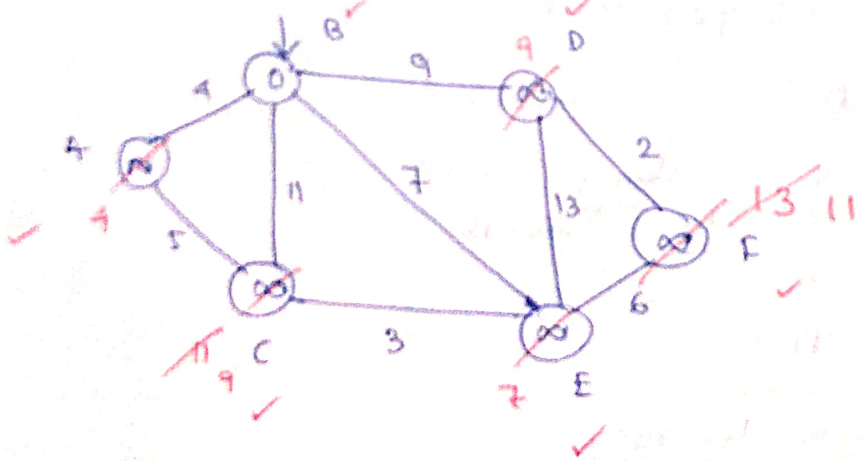
=>



=>



=>



Dynamic approach table

	A	B	C	D	E	F
	∞	(0)	∞	∞	∞	∞
B	(4)		11	9	7	∞
A			9	9	(7)	∞
E			(9)	9		13
C				(9)		13
D						(11)

5) i) $T(n) = 5T(n/5) + n\log n$

$\Rightarrow$ $f(n) = n\log n$

$a = 5$, $b = 5$ $\Big\}$ $n^{\log_b a} = n$

so, $n\log n > n$.

or $n < n\log n$

so, the T.C $= O(n(\log n)^r)$

(ii) $T(n) = 2T(n/2) + n^3$

$\Rightarrow$ $f(n) = n^3$

$a = 2$, $b = 2$ $\Big\}$ $n^{\log_b a} = n$

$\Rightarrow$ $n^3 > n$

so, T.C $= O(n^3)$

section-C

1) a). what is dynamic programming?

⇒ DP is an algorithmic technique used to solve problems by breaking them into smaller ~~parts~~ overlapping subproblems.

⇒ Each subproblem is solved once & its result is stored (Memoization/Tabulation) to avoid repeated computation & improve efficiency.

b) compare & contrast dp & greedy. ——— (2024)

c) describe the dp method to find LCS.

⇒ Given two strings:

$X = x_1 x_2 \cdots x_m$ (length m)

$Y = y_1 y_2 \cdots y_n$ (length n)

The LCS is the longest sequence of characters that appears in both strings in same order, but not necessarily contiguously.

DP table definition:—

dp[i][j] = length of LCS of substrings X[1--i] & Y[1--j]

Table size $(m+1) * (n+1)$

Recurrence relation:—

$$dp[i][j] = \begin{cases} 0 & ; \text{if } i=0 \text{ or } j=0 \\ 1 + dp[i-1][j-1] & ; \text{if } x[i] = y[j] \\ \max(dp[i-1][j], dp[i][j-1]) & ; \text{if } x[i] \neq y[j]. \end{cases}$$

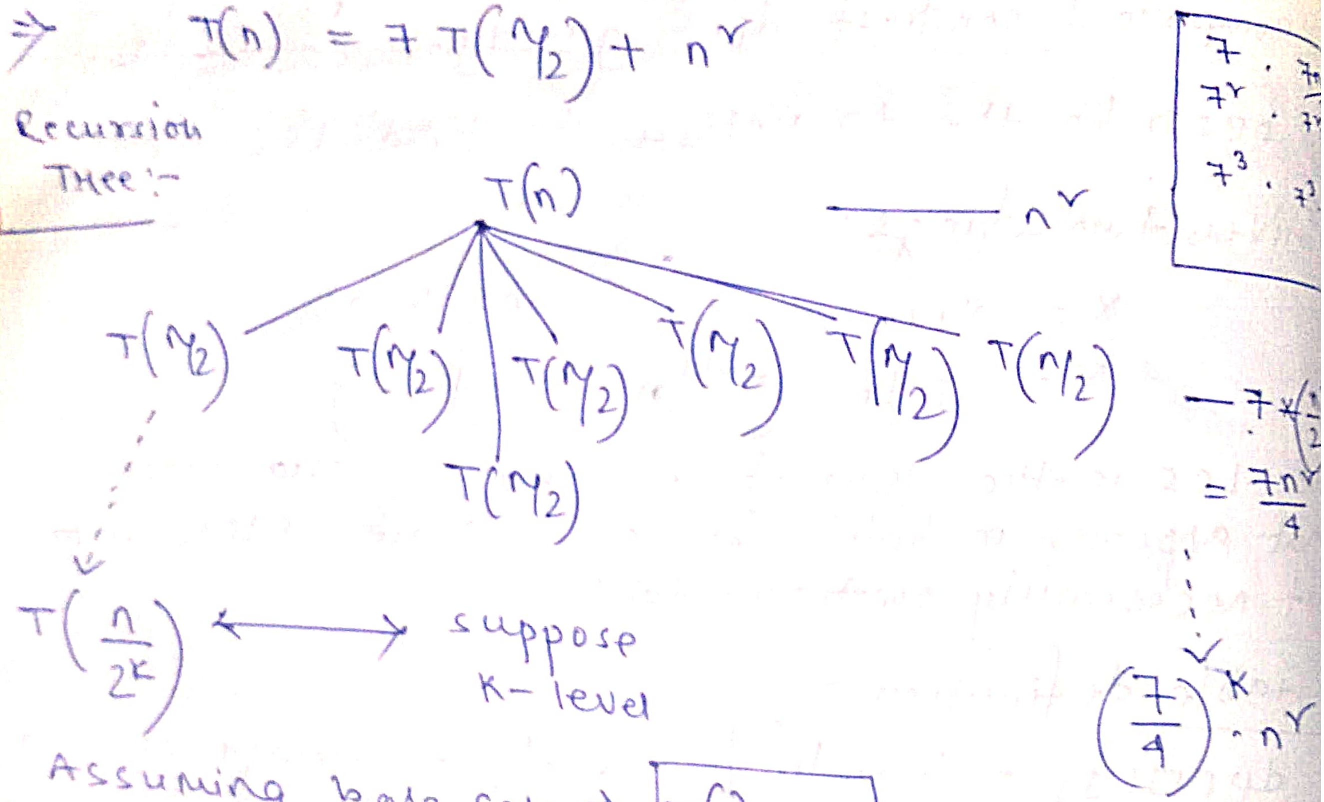
Algorithm steps:-

1) create a dp table dp[m+1][n+1]
2) Initialize 1st col & 1st row with 0
3) Fill the table row by row using Recurrence Relation.
4) The value of dp[m][n] gives the length of the LCS.

2) a) Deduce the TC of strassen's MCM.

$\Rightarrow \quad T(n) = 7\,T(n/2) + n^2$

Recursion Tree:-



Assuming base case $\Rightarrow$ $T(1) = 1$

$\frac{n}{2^K} = 1 \Rightarrow K = \log_2 n$

So, the TC (total cost) $= n^2 + \frac{7}{4}n^2 + \left(\frac{7}{4}\right)^2 n^2 + \dots \left(\frac{7}{4}\right)^K$

$= \sum_{K=0}^{\log_2 n} n^2 \left(\frac{7}{4}\right)^K$

So, $T(n) = n^2 \left(\frac{7}{4}\right)^{\log_2 n}$

$= n^2 \cdot n^{\log_2 (7/4)} = n^{2 + \log_2 \frac{7}{4}} = n^{2.81}$

$T.C = O(n^{2.81})$

b) A1 A2 A3 A4
10X5 5X20 20X10 10X5

MCM algo: $M[i,j] = \begin{cases} 0 ; & \text{if } i=j \\ \min\limits_{i \le k < j} \{ M[i,k] + M[k+1,j] + P_{i-1} * P_k * P_j \} ; & \text{otherwise} \end{cases}$

11 0	22 0	33 0	44 0
12 1000	23 1000	34 1000	
13 1500	24 1250		
14 1500			

The min scalar multiplication needed to multiply A1*A2*A3*A4 is 1500; M1(M2M3)M4 ✓

1) M[12]
= M1 * M2
=> 10 X 5 X 20 = 1000

2) M[23]
= M2 * M3
=> 5 X 20 X 10 = 1000

3) M[34]
= M3 * M4
=> 20X10 X 5 = 1000

4) M[13]
= M1 * M2 * M3

M1(M23) | (M12)M3
1000 + 10X5X10 | 1000 + 10X20X10
= 1500 | = 3000

5) M[24]
= M2 * M3 * M4

M2(M34) | (M23)M4
1000 + 5X20X5 | 1000 + 5X10X5
= 1500 | = 1250 ✓

6) M[14]

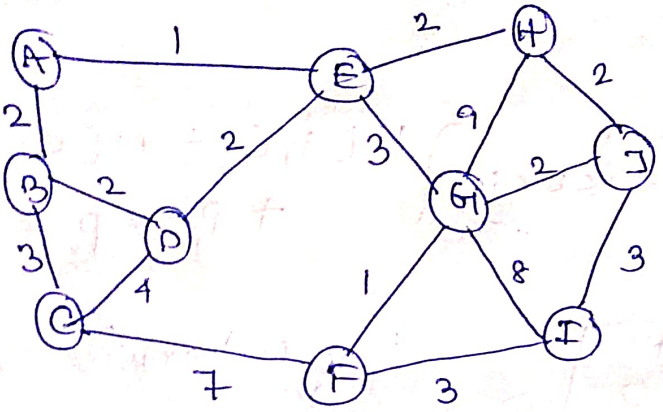
M1(M24) | M(12)M(34) | M(13)M4
1250 + 10X5X5 = 1500 ✓ | 1000 + 1000 + ( ) | 1500 + --

3) a) what is ST?

b) No. of ST $= 7^{7-2} = 7^5$

$= 16807$

c)

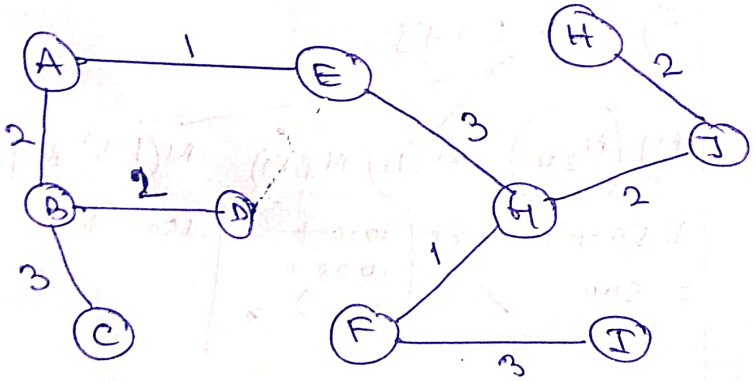


Applying kruskal :-

edges in decreasing order

{ A-E, 1 ; F-G, 1 ; B-D, 2 ; D-E, 2 ; G-J, 2 ; H-J, 2 ;
A-B, 2 ; B-C, 3 ; E-G, 3 ; F-I, 3 ; J-I, 3 ; C-D, 4 ;
C-F, 7 ; G-I, 8 ; G-H, 9 }

⇒



Min spanning Tree

4)

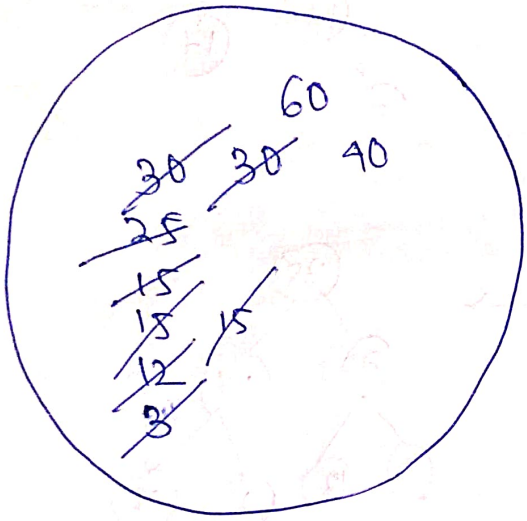
a	b	c	d	e	f
12	15	3	30	25	15

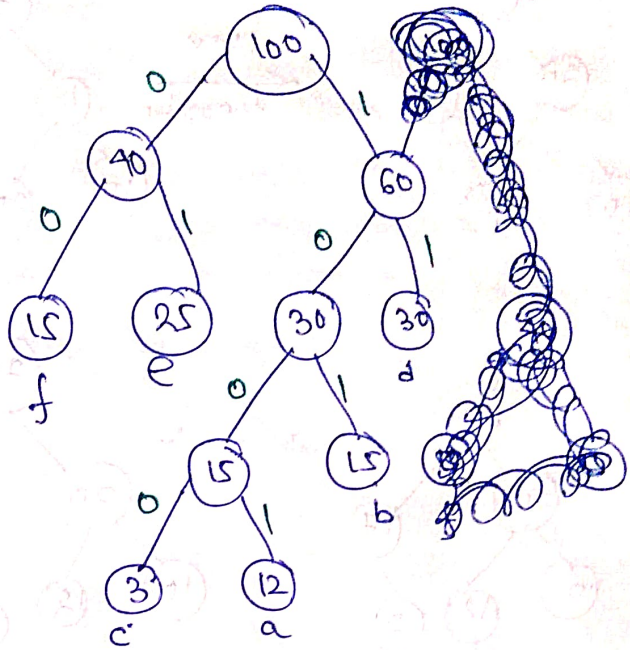
fixed length $= \lceil \log N \rceil$, $N = 6$

$= \lceil \log 6 \rceil = 3$

total bits $= 100 \times 3 = 300$.

Huffman encoding :-





char	bits	total bits
a	1001	$12 \times 4 = 48$
b	101	$15 \times 3 = 45$
c	1000	$4 \times 3 = 12$
d	11	$30 \times 2 = 60$
e	01	$25 \times 2 = 50$
f	00	$15 \times 2 = 30$

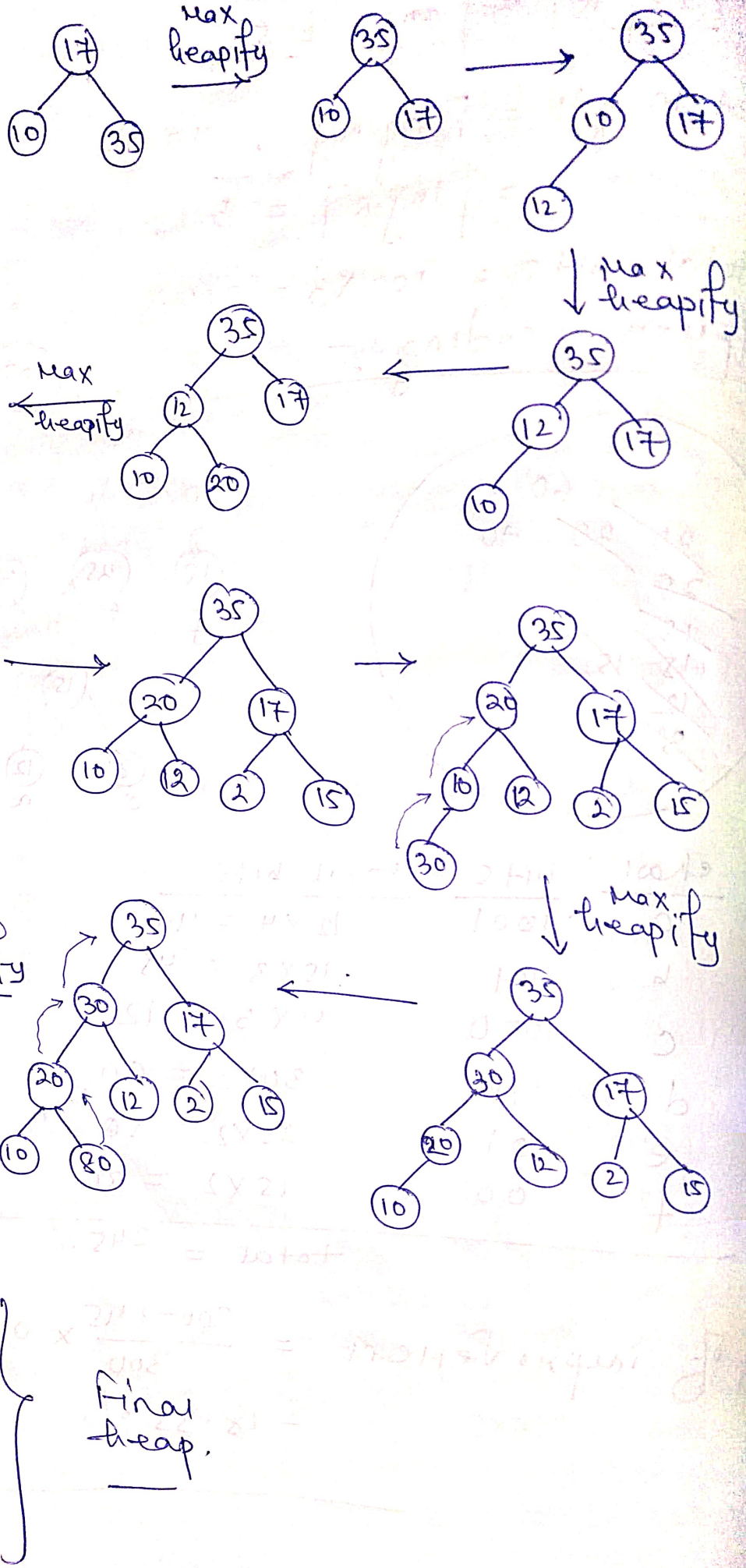
total $= 245$.

% of improvement $= \frac{300 - 245}{300} \times 100$

$= 18.33\ \%$

b) {17, 10, 35, 12, 20, 2, 15, 30, 80, 1}

⇒ Build heap:-



Final heap.

5) a) stat 0/1 Knapsack

b)

P	40	10	50	30	60
W	80	10	40	20	90

M = 110

taking single item :-

item	wt	Profit
1	80	40
2	10	10
3	40	50
4	20	30
5	90	60 ✓

Algorithm

KS(i, w)

if i=0 or j=0 → 0

if $w_i > w$ → KS(i-1, w)

Max(P_i + KS(i-1, w-w_i), KS(i-1, w)) if $w_i \le w$

taking two items

item	wt	Profit
1+2	90	50
1+3	120 x	
1+4	100	70
1+5	170 x	
2+3	50	60
2+4	30	40
2+5	100	70
3+4	60	80
3+5	130 x	
4+5	110	90 ✓

taking three items

items	wt	Profit
1+2+3	130 x	
1+3+4	140 x	
1+2+4	110	80
1+2+5	180 x	
1+3+5	x	
1+4+5	x	
2+3+4	70	90 ✓
2+3+5	x	
2+4+5	120 x	
3+4+5	x	

taking 4 items together

items	wt	Profit
1+2+3+4	x	
1+2+3+5	x	
1+2+4+5	x	
1+3+4+5	x	
2+3+4+5	x	

So max Profit Possible = 90

& items are, 2 → 3 → 4

5) Using fractional

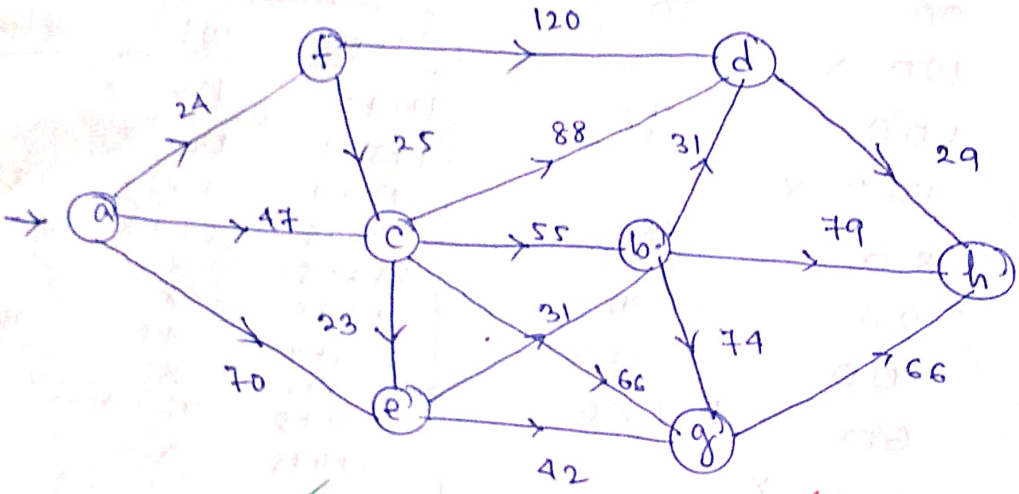
$M = 110$

	a	b	c	d	e
P	40	10	50	30	60
W	80	10	40	20	90
P/W	0.5	1	1.25	1.5	0.66

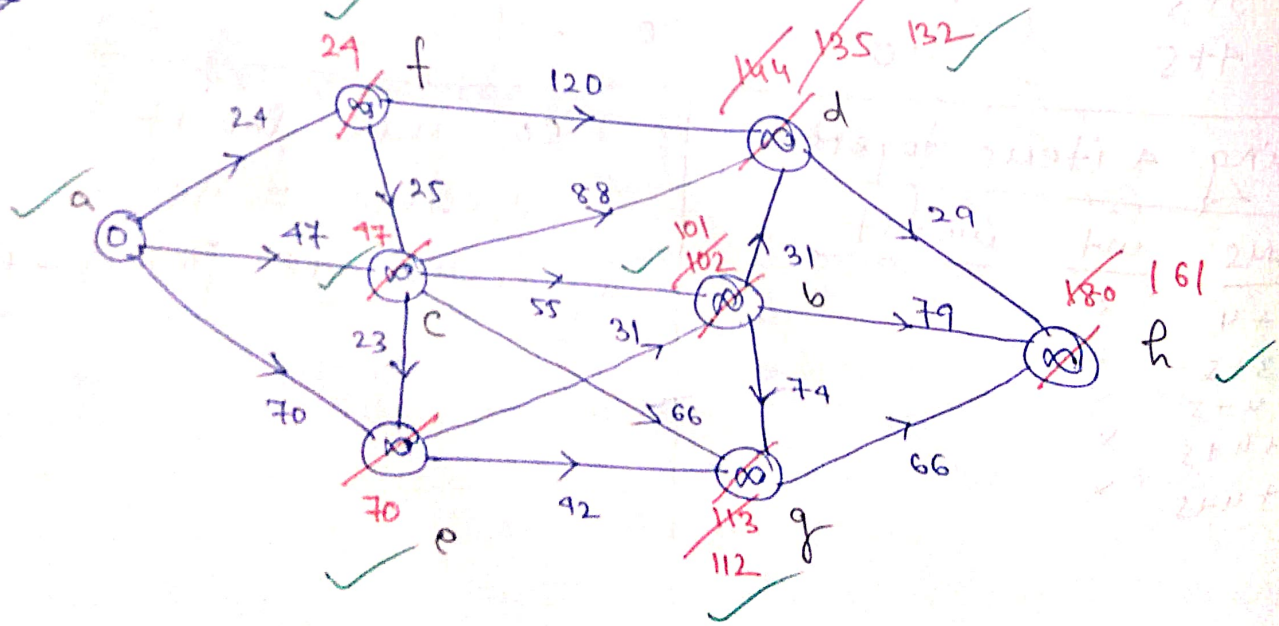
d	c	b	e
←20→	←40→	←10→	←40→
0 P = 30	P = 30 + 50 = 80	P = 90	P = 90 + 0.66 × 40 = 116.664

(ANS)

6) a) Dijkstra :-



⇒



Dynamic programming table

	a	b	c	d	e	f	g	h
	(0)	∞	∞	∞	∞	∞	∞	∞
a		∞	47	∞	70	(24)	∞	∞
f		∞	(47)	141	70		∞	∞
c		102		135	(70)		113	∞
e		(101)		135			112	∞
b				132			(112)	180
g				(132)				180
d								(161)

b) i) State the master's Theorem :-

Master's theorem provides a direct way to find TC of divide & Conquer recurrences of the form :

$$T(n) = a\,T(n/b) + f(n)$$

if $f(n) > n^{\log_b a}$

then $TC = O(f(n))$

else $TC = O(n^{\log_b a})$

(ii) $T(n) = 3T(n/2) + n\log n$

$\Rightarrow f(n) = n\log n$

$a = 3$, $b = 2$ $\Big\}$ $n^{\log_b a} = n^{\log_2 3} = n^{1.5}$

so, $\boxed{TC = O(n^{1.5}) = O(n^{\log_2 3})}$

iii) $T(n) = 9T(n/3) + n$

$f(n) = n$

$a = 9$, $b = 3$ $\Big\}$ $n^{\log_b a} = n^{\log_3 9} = n^2$

$\boxed{T.C = O(n^2)}$ ✓

Recurrence Relation:-

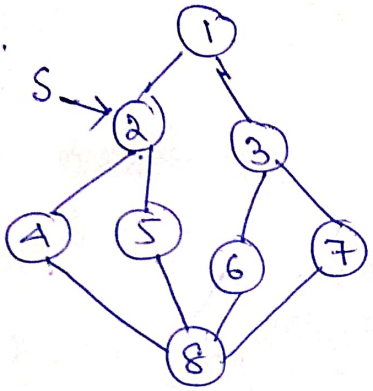
1)	BS :	$T(n) = T(n/2) + c$	$O(\log n)$
2)	MS :	$T(n) = 2T(n/2) + n$	$O(n\log n)$
3)	QS :	$T(n) = 2T(n/2) + n$	$O(n\log n)$
4)	strassen's MCM :	$T(n) = 7T(n/2) + n^2$	$O(n^{2.81})$
5)	Fibonacci :	$T(n) = T(n-1) + T(n-2)$	$O(2^n)$
6)	Factorial :	$T(n) = T(n-1) + c$	$O(n)$
7)	selection/ Insertion	$T(n) = T(n-1) + n$	

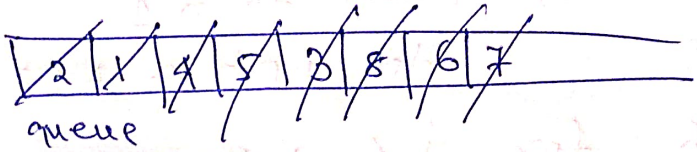
DFS & BFS

BFS :- Also called Breadth first search.
$G(n, e)$, 'n' vertices & 'e' edges & source vertex 's'.

BFS traversal visits all adjacency vertices, followed by visits to 2 edge length from source & so-on.

e.g.

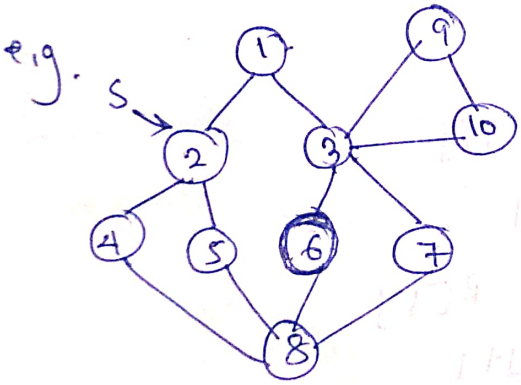




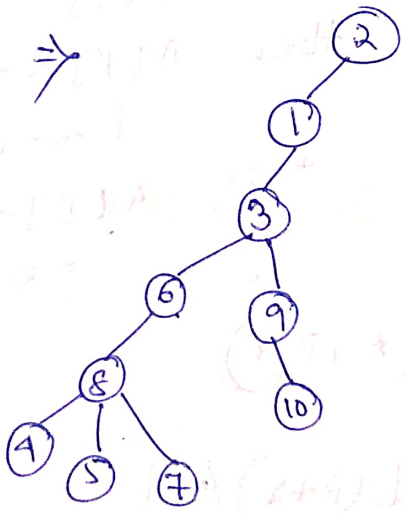
BFS sequence :- 2 1 4 5 3 8 6 7

DFS :- Also called Depth first search :-
Algo. start at the root node & explore as far as possible along each branch before backtracking

e.g.



DFS



one of the DFS sequence :-

2 1 3 6 8 4 5 7 9 10.

Algorithm

1) Merge sort :-

MERGE (A, P, q, r)

1. $n_1 \leftarrow q - p + 1$
2. $n_2 \leftarrow r - q$
3. Create arrays $L[1 \cdots n_1]$ & $R[1 \cdots n_2 + 1]$
4. for $i \leftarrow 1$ to n_1
5. do $L[i] \leftarrow A[p + i - 1]$
6. for $j \leftarrow 1$ to n_2
7. do $R[j] \leftarrow A[q + j]$
8. $L[n_1 + 1] \leftarrow \infty$
9. $R[n_2 + 1] \leftarrow \infty$
10. $i \leftarrow 1$
11. $j \leftarrow 1$
12. for $k \leftarrow p$ to r
13. do if $L[i] \leq R[j]$
then $A[k] \leftarrow L[i]$
$i \leftarrow i + 1$
else $A[k] \leftarrow R[j]$
$j \leftarrow j + 1$

MERGE SORT (A, P, r)

1. if $p < r$
2. $q = L(p + r)/2 \rfloor$
3. MERGE SORT (A, P, q)
4. MERE SORT (A, q+1, r)
5. MERGE (A, P, q, r)

2. Quick sort :-

=> PARTITION (A, p, r)

```
x = A[r]
i = p-1
for j = p to r-1
    if A[j] ≤ x
        i = i+1
        exchange A[i] & A[j]
exchange A[i+1] with A[r]
return i+1
```

=> QUICKSORT (A, p, r)

```
if p < r
    q = PARTITION (A, p, r)
    QUICKSORT (A, p, q-1)
    QUICKSORT (A, q+1, r)
```

