ADA Assignment

Q:-1 Derive T.C. of `for(int i=1; i<=n; i++) for(int j=1; j<=n; j++) for(int k=1; k<=j; k++) count++;`

$T.C. = O(n \cdot n \cdot n)$

$\simeq O(n^3)$

Q:2 Explain space complexity in

-|| Recursive Fibonacci

The recursive fibonacci fun is defined as:

```
int fib(int n) {
    if (n <= 1) return n;
    return fib(n-1) + fib(n-2);
}
```

Space Complexity

- Due to recursive calls, the function creates a call stack.
- The max depth of recursive call stack is n

$\Rightarrow S.C. = O(n)$

-(ii) Matrix chain Multiplication (MCM):

Definition:
We have to find the min # of scalar multiplications needed to multiple a chain of matrices. The dp soln uses table to store intermediate results

S.C.
└ If there are n matrices, we typically use:
└ A 2D DP table dp[i][j] where dp[i][j] stores the min cost of multiplying matrices i to j.

S.C. ∈ O(n²))

→ Optionally, a separate table used to store split positions for reconstruction

Q:3 Linear Search overview:
Linear search checks each element one by one from the beginning until finds the target.

Best case:
- When the element is at first position
- Best case comparisons: 1

- Worst case:
  When the element is at the last position or not present at all

worst case comparisons: 100

- Average case:
  Two scenarios:
  1. if element is present, the average number of comparisons
     $\frac{1 + 2 + 3 + \cdots + 100}{100} = 50.5$

  2. if element is not present, 100 comparisons needed.

If we assume 50-50 probability of being present or absent

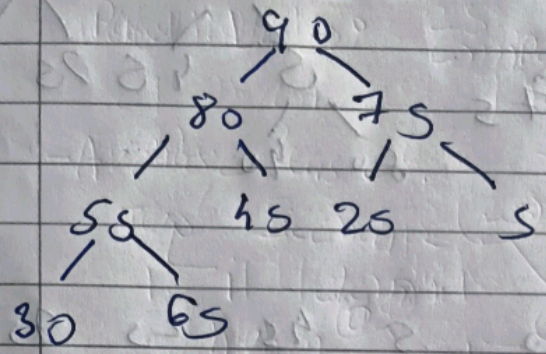
Avg. comparisons = $0.5 \cdot 50.5 + 0.5 \cdot 100 = 75.25$

case	comparisons
Best	1
worst	100
Avg. case (present)	50.5
Avg. case (present + equally)	75.25

Q:4 Sort [55, 25, 90, 65, 45, 75, 5, 30, 85]

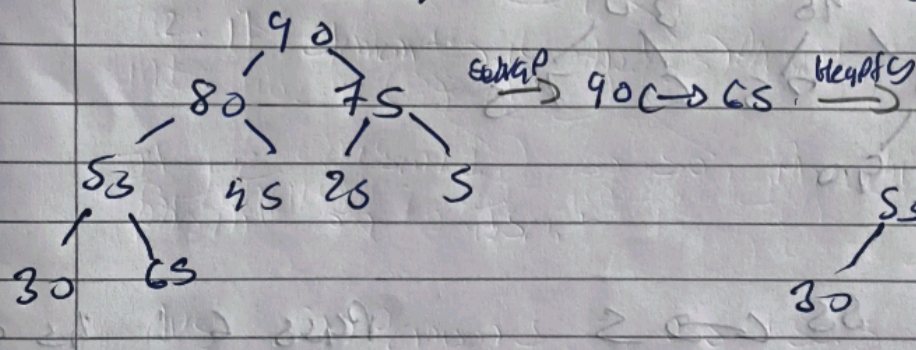
i) Heap sort

Step 1: Build Max Heap



Step 2: Heap sort

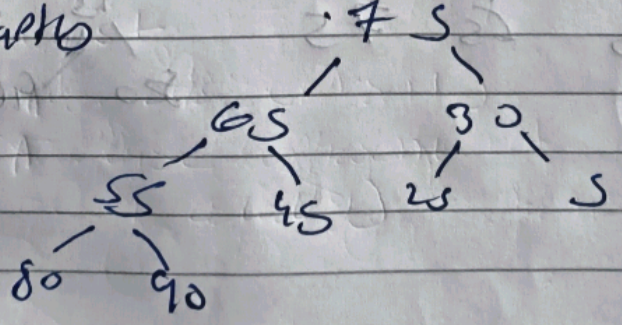
Pass 1: 90 80 75 55 45 25 5 30 65



Pass 2: ~~80~~ ~~65~~ 75 55 45 25 5 30 90

swap 80 ↔ 30

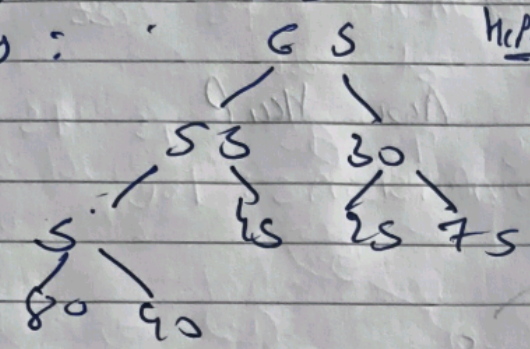
Heapify



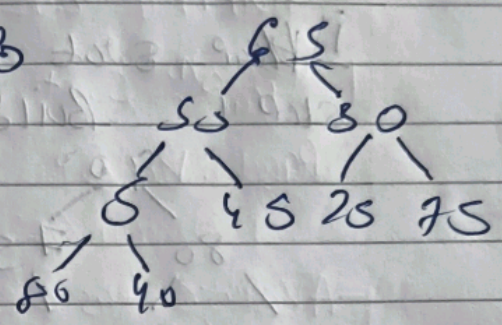
Pass 3 : 75 65 30 55 45 25 5 85 90

Swap : 75 ↔ 5

Heapify :



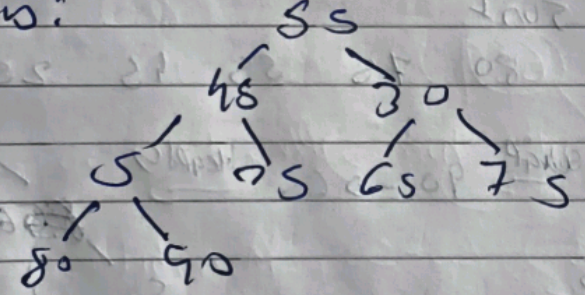
Heapify



Pass 4 : 65 55 30 5 45 25 75 85 90

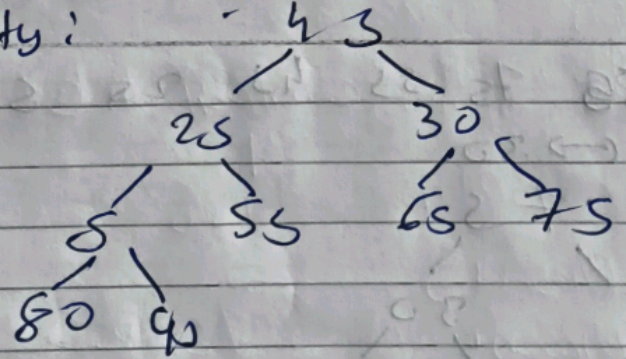
Swap : 65 ↔ 25

Heapify :

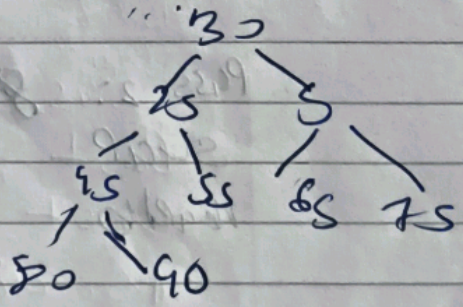


Pass 5 : 55 ↔ 5

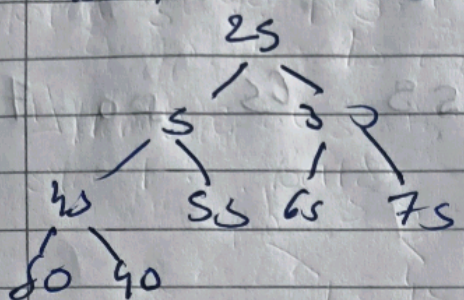
Heapify :



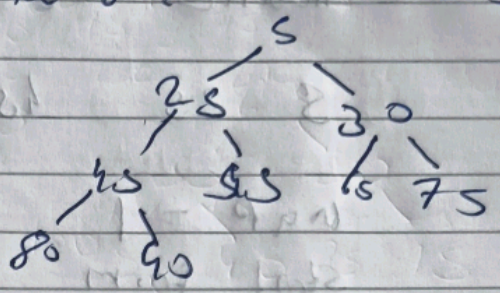
Pass 6 : 45 ↔ 5



Pass 7 : 30 ⟶ 5



Pass 8 : 25 ⟶ 5



Sorted array : 5, 25, 30, 45, 55, 65, 75, 80, 40

2) Shell Sort

55, 25, 40, 65, 45, 75, 5, 30, 80

→ Gap = 4

- Compare 55 & 45 → no change
- Compare 25 & 75 → no change
- Compare 40 & 5 → swap
  55, 25, 5, 30, 45, 75, 40, 65, 80
- Compare 60 & 30 → swap
  55, 25, 5, 30, 45, 75, 40, 65, 80
- Compare 45 & 80 → no change

→ Gap = 2

55 25 5 30 45 75 40 65 80

- Group 1 : 55 75 45 40 80
- Group 2 : 25 30 75 65

Sort these mini-set :

Group 1 : 5 45 55 80 40

Group 2 : 25 30 65 75

Rebuild the full array

5 25 45 30 55 65 80 75 90

unP → 1

Start from i = 1

- 25 > 5 → ok
- 45 > 25 → ok
- 30 < 45 → Shift → 5, 25, 30, 45, 55, 65, 80, 75, 90
- 55 > 45 → ok
- 65 > 55 → ok
- 80 > 65 → ok
- 75 < 80 → Shift → 5, 25, 30, 45, 55, 65, 75, 80, 90
- 90 > 80 → ok

Final sorted array:

5 25 30 45 55 65 75 80 90

Q: 5 use Counting Sort to sort:
6, 0, 0, 2, 0, 1, 3, 3, 6, 1, 3, 2

	0	1	2	3	4	5	6
Freq:	2	2	2	3	0	0	2

i = 0

Pass 1: i = 0

array: 0 0 2 0 1 3 3 6 1 3 2

j = 2

i = 1

Pass 2: i = 1

array: 0 0 1 1 1 3 3 6 1 3 2

j = 4

i = 2

Pass 3: i = 2

array: 0 0 1 1 2 2 3 6 1 3 2

j = 6

i = 3

Pass 4: i = 3

array: 0 0 1 1 2 2 3 3 3 6 2

j = 9

i = 4

Pass 5: i = 4

array → no change

j = 9

i = 5

Pass 6: i = 5

array → no change

j = 9

i = 6

Pass 7: i = 6

entries 0 0 1 1 2 2 3 3 3 6 6

j = 11

i = 7

Sorted ans: 0, 0, 1, 1, 2, 2, 3, 3, 3, 6, 6

Q:6 Sort [121, 235, 55, 971, 321, 176]

using Radix Sort

Pass 1: Sort by Units place

1 → 121, 971

2,5 → 235, 55

6 → 176

0, 2, 3, 4, 7, 8, 9 → none

Combined: 121 971 235 55 176 321

Pass 2: Sort by tens place

2 → 121, 176, 321

3 → 235

5 → 55

7 → 971

121 176 321 235 55 971

Pass 3: sort by hundreds place

0 → 55

1 → 121, 176

2 → 235

3 → 321

9 → 971

final sorted array :

55 121 176 235 321 971

When is Radix sort better than comparison sorts ?

Condition	Why
All numbers are fixed len	Radix sort in linear time
You need stable sort	Radix sort preserves relative order
You've large numbers of small ints	Efficient for datasets like [0-999], strings

Q:4 For an array of size 1024:

(a) How many recursive call are made in Binary search?

formula : $\log_2(n) + 1$

calls = $\log(2^{10}) + 1$

= 11

(b) Prove T.C. = O(log n)

→ At each level the Problem size becomes half.

∴ Total levels = $\log_2(n)$

There is only 1 recursive call. So

$T(n) = T(n/2) + O(1)$

Recursion tree:

T(n)

└ T(n/2)

└ T(n/4)

└

⋮

└ T(1)

So,

$T(n) = O(1) + O(1) + O(1) + \cdots + O(1)$

$= O(\log n)$

Q: 8 find max and min in
45, 85, 23, 67, 12, 90, 34, 55

- Divide into
  - Left → 45 85 23 67
    - Left → 45 85
      - ↳ max = 85, min = 45 (1 comparision)
    - Right → 23 67
      - ↳ max = 67, min = 23 (1 comparision)
    - max (85, 67) ⇒ max = 85
      min (45, 23) → min = 23 (2 comparision)
  - Right → 12 90 34 55
    - Left → 12 90
      - ↳ max = 90, min = 12
    - Right → 34 55
      - ↳ max = 55, min = 34
    - max (90, 55) → max = 90
      min (12, 34) → min = 12
  - max (90, 85) → max = 90
    min (23, 12) → min = 12

Each pair ⇒ 1 comparision
Each merge → 2 comparisions

Total = 4·1 + 3·2
= 10 comparisions

Q:4 Sort 12, 25, 5, 31, 10, 66, 90

// Merge Sort

12 25 5 31 10 66 90

→ left: 12 25 5 31

→ left: 12 25

→ left: 12

→ right: 25

→ merge: 12 < 25 : 12 25

→ right: 5 31

→ left: 5

→ right: 31

→ merge: 5 < 31 : 5 31

→ Merge: 5 < 12 : 5

12 < 31 : 5 12

25 < 31 : 5 12 25 31

→ right: 10 66 90

→ left: 10 66

→ left: 10

→ right: 66

→ merge: 10 < 66 : 10 66

→ right: 90

→ Merge: 10 < 90 : 10

66 < 90 : 10 66 90

→ Merge: 5 < 10 : 5

10 < 12 : 5 10

12 < 66 : 5 10 12

25 < 66 : 5 10 12 25

31 < 66 : 5 10 12 25 31 66 90

Q:1. Multiply two 2×2 matrix using divide and conquer method

→ Let two matrices

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}, \quad B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

and result $C = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$

→ Divide & Conquer

We treat 2×2 matrices as blocks of 4 submatrices

$$c_{ij} = A_{i1} B_{1j} + A_{i2} B_{2j}$$

so, $c_{11} = a_{11} b_{11} + a_{12} b_{21}$ $\quad c_{21} = a_{21} b_{11} + a_{22} b_{21}$

$c_{12} = A_{11} B_{12} + A_{12} B_{22}$ $\quad c_{22} = a_{21} b_{12} + a_{22} b_{22}$

→ MatrixMultiply(A, B)

if size of A = 1×1 :

return A[0][0] * B[0][0]

else

divide A into four parts : $A_{11}, A_{12}, A_{21}, A_{22}$

divide B into four parts : $B_{11}, B_{12}, B_{21}, B_{22}$

C_{11} = matrixmultiply(A_{11}, B_{11}) + matrixmultiply(A_{12}, B_{21})

C_{22} = matrixmultiply(A_{11}, B_{12}) + matrixmultiply(A_{12}, B_{22})

C_{21} = matrixmultiply(A_{21}, B_{11}) + matrixmultiply(A_{22}, B_{21})

C_{12} = matrixmultiply(A_{21}, B_{11}) + matrixmultiply(A_{22}, B_{21})

Combine $C_{11}, C_{12}, C_{21}, C_{22}$ into result matrix C

return C

Q:11 Implement recursive exponentiation a^b or $a \wedge b$ using divide-and-conquer and analyze time complexity

Computes $a^b = a + a + \ldots + a$ (b times)

Recurrence $O(b)$

→ Divide and Conquer

If b is even: $a^b = a^{b/2} \times a^{b/2}$

If b is odd: $a^b = a \times a^{b/2} \times a^{b/2}$

Algo:

```
Power(a, b)
  if b == 0;
      return 1
  if b == 1;
      Return a
  int temp = Power(a, b/2)
  if b%2 == 0;
      Return temp * temp
  else
      Return a * temp * temp
```

T.C.: Time per call = $O(1)$

$T(b) = T(b/2) + O(1)$

$\therefore T.C. = O(\log_2 b)$

Q:- A vending machine in your college's electronics lab only accepts coins of denominations ₹1, ₹5 and ₹10. You want to buy component worth ₹14.

→ Recurrence (for $x \geq 1$):

$$dp[x] = 1 + \min \begin{cases} dp[x-1], \\ dp[x-5] \text{ if } x \geq 5, \\ dp[x-10] \text{ if } x \geq 10 \end{cases}$$

Base $dp[0] = 0$

computing up to 14

$dp[1] = 1$ (1), $dp[2] = 1 + dp(2-1)$, $dp[3] = 1 + dp(3-1)$

$dp[4] = 1 + dp(4-1)$, $dp[5] = 1 + dp(5-1)$

$dp[6] = 1 + dp(6-1, 1-1)$, $dp[7] = 1 + dp(7-1, 1-2)$

$dp[8] = 1 + dp(8-1, 1-3)$, $dp[9] = 1 + dp(9-1, 1-4)$

$dp[10] = 1 + dp(10-1)$, $dp[11] = 1 + dp(10-1, 1-1)$

$dp[12] = 1 + dp(10-1, 1-2)$, $dp[13] = 1 + dp(10-1, 8-1)$

$dp[14] = 1 + dp(10, 4-1, 1-4)$

min coins needed = 5

Combination: ₹10 + ₹1 + ₹1 + ₹1 + ₹1

Q3 Explain and apply the principle of optimality for Matrix chain Multiplication

→ Given dimensions of matrices $A_1, A_2, \ldots A_n$ find the minimum numbers of scalar multiplications needed to compute $A_1 \cdot A_2 \cdot A_3 \cdots A_n$

→ Applying Priciple of optimality:

Suppose we sibe the chain at position k:

$(A_1 \cdot A_2 \cdot A_3 \cdots A_k)(A_{k+1} \cdots A_n)$

→ To get optimal solution for the whole chain
- The subsequence of let $(A_1 \ldots A_n)$ must also be optimal
- The same for right side $(A_{n+1} \ldots A_m)$
- Recurrence Relation

$$\cdot M[i,j] = \min_{i \le k < j} (M[i,k] + M[k+1,j] + P_{i-1} P_k P_j)$$

2) Knapsack Problem
→ Given items with weights w_i & w_i value v_i, maximize total value in a knapsack of capacity w.

→ Applying principle of optimality
At each stem of i^{th} item
- Either include item i,
- or exclude item i.

- If the overall soln is optimal:
- If i^{th} is included, then the remaining items must must form capicity $w - w_i$
- If i^{th} is exclude the remain items form optimal soln is capacity w.

Recurrence Relation :

$$k(n, w) = \begin{cases} 0 & i \le 0 \\ k(n-1, w) & w_n > w \\ \max(v_n + k(n-1, w-w_n), k(n-1, w)) & \text{otherwise} \end{cases}$$

Q:14 find C(10, 6), using

1) Recursion

$C(n, r) = \begin{cases} 1 & \text{if } r=0 \text{ or } r=n \\ C(n-1, r-1) + C(n-1, r) & \text{otherwise} \end{cases}$

so for C(10, 6)

C(10, 6) = C(9, 5) + C(9, 6)

C(9, 5) = C(8, 4) + C(8, 5)

C(9, 6) = C(8, 5) + C(8, 6)

o

:

base case C(n, 0) = 1 and C(10, 10) = 1

final evaluation gives 210.

2) DP

We build Pascal's Triangle using DP

Row \ Col	0	1	2	3	4	5	6	7	8	9	10
0	1										
1	1	1									
2	1	2	1								
3	1	3	3	1							
4	1	4	6	4	1						
5	1	5	10	10	5	1					
6	1	6	15	20	15	6	1				
7	1	7	21	35	35	21	7	1			
8	1	8	28	56	70	56	28	8	1		
9	1	9	36	84	126	126	84	36	9	1	
10	1	10	45	120	210	252	210	120	45	9	1

for C(10, 6) = dp[10][6]

= 210

Q15 Solve 0/1 Knapsack (Capacity = 10)

item	weight	value
1	5	30
2	4	20
3	6	25

dp Table

$$dp[i][w] = \begin{cases} dp[i-1][w] & \text{if } weight[i] > w \\ \max(dp[i-1][w], & \text{otherwise} \\ value[i] + dp[i-1][w - weight[i]]) \end{cases}$$

Table filling:

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	30	30	30	30	30	30
2	0	0	0	0	20	30	30	30	50	50	50
3	0	0	0	0	20	30	30	30	50	50	50

Max value = 50

Optimal set : Item 1 and item 2

Q16 Matrix dimensions = [40, 20, 30, 10, 30]

num of matrix = 4

→ $A_1 = 40 \times 20$, $A_2 = 20 \times 30$,

$A_3 = 30 \times 10$, $A_4 = 10 \times 30$

DP formula

$$M[i][j] = \min_{i \le k < j} [M[i][k] + M[k+1][j] + p[i-1]p[k]p[j]]$$

OP TABLE

i\j	1	2	3	4
1	0	24000	14000	26000
2	0	0	6000	12000
3	0	0	0	9000
4	0	0	0	0

Chain len = 1

M[1][1] = M[2][2] = M[3][3] = M[4,4] = 0

Chain len = 2

M[1][2] = 40 x 20 x 30 = 24,000

M[2][3] = 20 x 30 x 10 = 6,000

M[3][4] = 30 x 10 x 30 = 9000

Chain len = 3

M[1][3] = min (

(A1)(A2A3) : 0 + 6000 + 40 x 20 x 10 = 14,000 ,

(A1A2)(A3) : 24,000 + 0 + 40 x 30 x 10 = 36,000)

= 14,000

M[2][4] = min (

(A2)(A3A4) : 0 + 9000 + 20 x 30 x 30 = 27,000 ,

(A2A3)(A4) : 6000 + 0 + 20 x 10 x 30 = 12,000)

= 12,000

Chain len = 4

M[1][4] = min

(A1)(A2A3A4) : 0 + 12000 + 40 x 20 x 30 = 36,000 ,

(A1A2)(A3A4) : 24,000 + 9000 + 40 x 20 x 30 = 69,000 ,

(A1A2A3)(A4) : 14000 + 0 + 40 x 10 x 30 = 26,000)

= 26,000

final Ans = 26,000

optimal Parenthesization = ((A1 x (A2 x A3)) x (A4))

17 Use Floyd-Warshall Algo for this graph

M =

	A	B	C
A	0	3	∞
B	∞	0	1
C	2	∞	0

$$D_0 = M = \begin{bmatrix} 0 & 3 & \infty \\ \infty & 0 & 1 \\ 2 & \infty & 0 \end{bmatrix}$$

→ Iteration with k = A

$$D_1 = \begin{array}{c} \\ A \\ B \\ C \end{array} \begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{bmatrix} 0 & 3 & \infty \\ \infty & 0 & 1 \\ 2 & 5 & 0 \end{bmatrix} \end{array}$$

→ Iteration with k = B

$$D_2 = \begin{array}{c} \\ A \\ B \\ C \end{array} \begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{bmatrix} 0 & 3 & 4 \\ \infty & 0 & 1 \\ 2 & 5 & 0 \end{bmatrix} \end{array}$$

→ Iteration with C

$$D_3 = \begin{bmatrix} 0 & 3 & 4 \\ 3 & 0 & 1 \\ 2 & 5 & 0 \end{bmatrix}$$

→ And this is final answer

Q:18 We are given

X = "AGGTAB" (length 6)

Y = "GXTXAYB" (length 7)

→ DP Table formula

$$DP[i][j] = \begin{cases} 0 & \text{if } i=0 \text{ or } j=0 \\ DP[i-1][j-1]+1 & \text{if } X[i-1] = Y[j-1] \\ \max(DP[i-1][j], DP[i][j-1]) & \text{otherwise} \end{cases}$$

→ DP Table

		G	X	T	X	A	Y	B
	0	0	0	0	0	0	0	0
A	0	0	0	0	0	1	1	1
G	0	1	1	1	1	1	1	1
G	0	1	1	1	1	1	1	1
T	0	1	1	2	2	2	2	2
A	0	1	1	2	2	3	3	3
B	0	1	1	2	2	3	3	4

LCS len = dp[6][7] = 4

LCS str = "GTAB"

Q:19

Assembly station	Line 1	Line 2	Transfer lines
S1	7	9	2
S2	8	5	2
S3	6	4	1

→ Here 2 line and 3 station
we may either:
stay on the same line, or
switch to the other line

let
f1[i] = min time to reach station i on line 1
f2[i] = min time to reach station i on line 2

Recurrence:
f1[i] = min(f1[i-1] + time1[i],
f2[i-1] + transfer[i-1] + time1[i])
f2[i] = min(f2[i-1] + time2[i],
f1[i-1] + transfer[i-1] + time2[i])

→ initialization f1[1] = 7, f2[1] = 9

station 2:
f1[2] = min(f1[1] + 8, f2[1] + 2 + 8) = 15
f2[2] = min(f1[1] + 5, f1[1] + 2 + 5) = 12

station 3:
f1[3] = min(f1[2] + 6, f2[2] + 1 + 6) = 21
f2[3] = min(f2[2] + 4, f1[2] + 1 + 4) = 18

final answer = min(21, 18)
= 18

optimal path : line 2 → line 2 → line 2

Oct 20 Compare Greedy, divide and conquer, Dynamic Programming with real world examples

1. Greedy Algo:
Idea: Take the best local choice at each step hoping it leads to global optimum.
Property: works only if problem has greedy-choice property and optimal substructure.

- Real-world Example: Activity selection
- problem: select max num of non-overlapping meetings / intervals / activities.
- Why greedy work: locally choosing the earliest finish guarantees optimal ans.

Other Real-world cases: Huffman coding, dijkstra's shortest path.

2. Divide and conquer
Idea: Break problem into smaller subproblems, solve recursively, then combine sol.
property: subproblem are independent of each other

Real world Example: Merge sort
Problem: Sorting a large dataset
Approach: Divide array into halves, sort each half and then merge
Why work? subproblems are independent, merge give the result.

3) Dynamic Programming (DP)

Idea: Break problem into overlapping subproblems, store soln avoid recomputation

Property: optimal substructure + overlap subproblems

Real-world example: 0/1 knapsack Problem

Problem: chose items to maximize value without exceeding weight capacity.

Approach: Build a DP table to reuse subresult instead recalculate

T.C. = O(n·w)

other real world cases: MCM, floyd-warshall shortest path, LCS

Q:4) given,

start:	1	3	0	5	8	5
end:	4	5	6	7	9	9
Activity	A_1	A_2	A_3	A_4	A_5	A_6

step 1: sort by finish Time

Activity are in sorted order so no need to sort

step 2: select Activity greedily

pick first activity: A_1 (1 to 4)

A_2 (3 to 5) → overlap → no take

A_3 (0 to 6) → overlap → no take

A_4 (5 - 7) → pick A_4 (5-7)

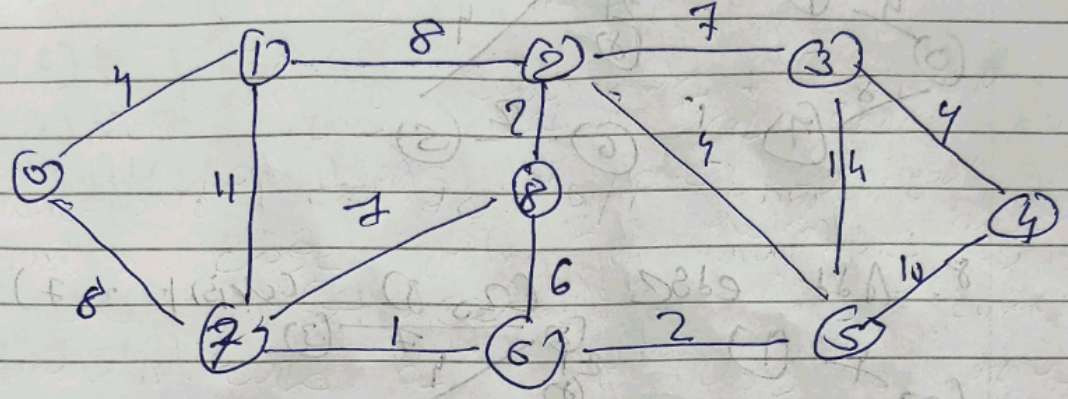
A_5 (8 - 9) → pick A_5 (8-9)

A_6 (5 - 9) → overlap → no take

Final Answer

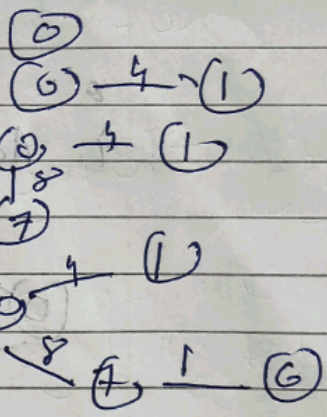
A(C1-4) A(4,5-7), A(5,C8-4)

Q.22

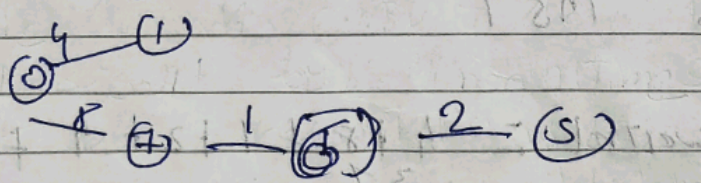


1) Apply Prim's Algo to vertex 0.

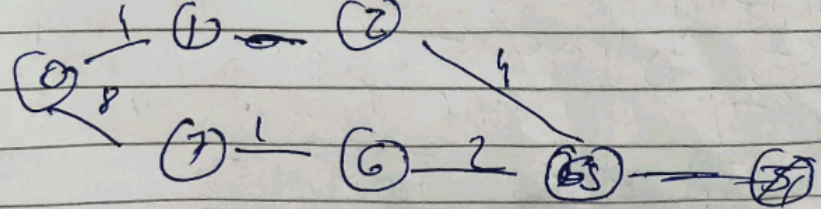
1. start with vertex 0



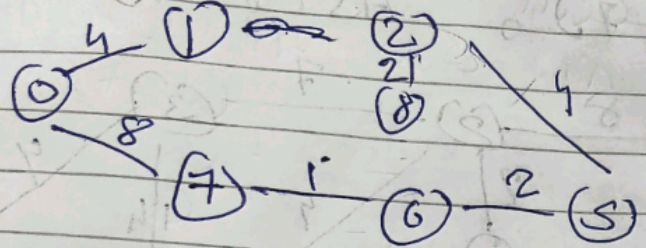
2. Add edge 0-1 (weight = 4)
3. Add edge 0-7 (weight = 8)
4. Add edge 7-6 (weight = 1)
5. Add edge (6-5) (weight = 2)



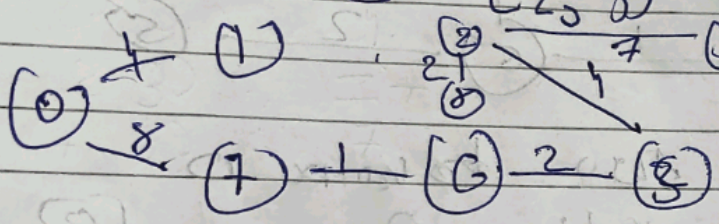
6. Add edge (2-5 (weight = 4)



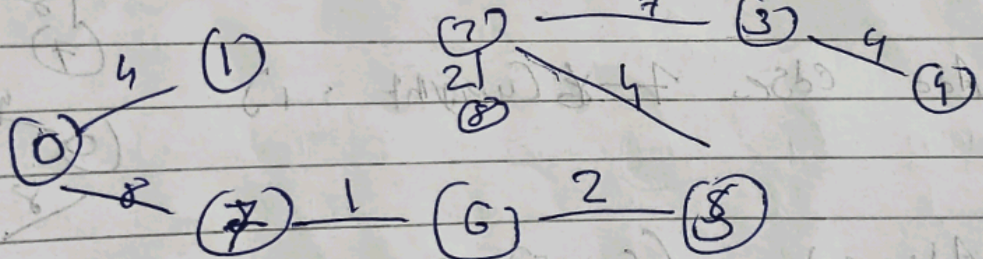
7. Add edge (2, 8) (weight = 2)



8. Add edge (2, 6) (weight = 7)



9. Add edge (3, 4) (weight = 9)



Final MST

Total weight = 4 + 8 + 1 + 2 + 4 + 2 + 7 + 9
= 37

2) Kruskal Algo.

-sort edges in ascending algo.

(7,6) : 1 , (6,5) : 2 , (2,8) : 2 , (6,8) : 6 ,
(0,1) : 4 , (2,5) : 4 ,
(8,7) : 7 , (2,3) : 7 , (1,2) : 8 , (0,7) : 8 ,
(3,4) : 9 , (5,4) : 10 , (1,7) : 11 , (3,5) : 14

Add (7,6) : 1. sets : {0}, {1}, {2}, {3}, {4}, {5}, {6,7}, {8}

Add (6,5) : 2. sets : {0}, {1}, {2}, {3}, {4}, {5,6,7}, {8}

Add (2,8) : 2 . sets : {0}, {1}, {2,8}, {3}, {4}, {5,6,7}

Add (0,1) : 4 . sets : {0,1}, {2,8}, {3}, {4}, {5,6,7}

Add (2,5) : 4 sets : {0,1}, {2,5,6,7,8}, {3}, {4}

Add (3,4) : 9 sets : {0,1}, {2,3,5,6,7,8}, {4}

Add (1,2) : 8 sets : {0,1,2,3,5,6,7,8}, {4}

Add (3,4) : 9 sets : {0,1,2,3,4,5,6,7,8}

MST Total = 1 + 2 + 2 + 4 + 4 + 7 + 8 + 9
= 37

Q:23

Job	Profit	Deadline
A	100	2
B	19	1
C	27	2
D	25	1
E	15	3

→ Sort Jobs in descending order wrt profit

Job	Profit	Deadline
A	100	2
C	27	2
D	25	1
B	19	1
E	15	3

→ Iterate through the sorted Job

Job A (P: 100, D: 2) The latest time slot is available is 2. The slots available at is 2 so Job A in slot 1.

Job C (P: 27, D: 2) : The latest time slot is available is 1 therefore Job C slot in slot 2.

Job D (P: 25, D: 1) ; There is no such slot available therefore so Job D can't schedule

Job B (P: 19, D: 1) can't schedule

Job E (P:15 D:3) Slot 3 is available then Job E schedule in slot 3.

Total Profit = 100 + 27 + 15
= 142

Slot 1 : Job C
slot 2 : Job A
slot 3 : Job E

Huffman given : A : 5
B : 9
C : 12
D : 13
E : 16
F : 45

1. Start with two smallest freq. Merge 'A' (5) and 'B' (9). The new node has a frequency of 5+9 = 14

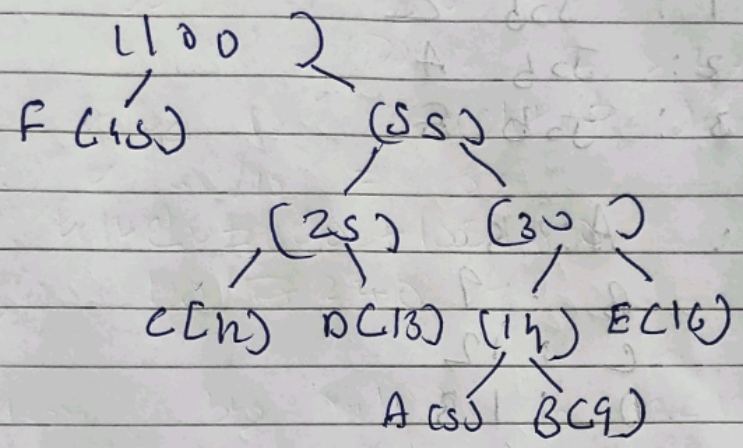
2. Next two smallest freq: Merge 'E' (12) and 'D' (13). The new node has a frequency of 12+13 = 25

3. Next two smallest freq: Merge node from step 1 (14) and 'E' (16) new node has freq of 14+16 = 30

4. Next two smallest freq: merge node from step 2 and step 3. 25+30 = 55

5. Final node
merge '45' (45) and 2nd node from step 4
new node has freq 45+55 = 100

Final Huffman Tree:



Find bits to encode "ABCDEF"

F: 0 (1 bit), C: 100 (3 bits),
D: 101 (3 bits), E: 111 (3 bits)
A: 1100 (4 bits), B: 1101 (4 bits)

Total bits: 1+3+3+3+4+4
= 18 bits

Q:25 Solve fractional knapsack:

Capacity = 50

item	weight	value
1	10	60
2	20	100
3	30	120

Calculate value to weight ratio (v/w)

item #1: 60/10 = 6

item 2: 100/20 = 5

item 3: 120/30 = 4

Sort item by ratio in decreasing order

item	1	2	3
ratio	6	5	4
weight	10	20	30

Greedily add the items the knapsack based on the sorted order

Add item 1:

Weight added : 10

value added : 60

Remaining capacity : 50 - 10 = 40

Add item 2:

weight added: 20

valued Added : 100

Remain capasity: 40 - 20 = 20

3. Add item 3:

fraction to take : $\frac{\text{Remaining capacity}}{\text{Item 3 weight}} = \frac{2}{3}$

value from fraction = $\frac{2}{3} \times 120 = 80$

weight from fraction : $\frac{2}{3} \times 30 = 20$

Total value = $60 + 100 + 80$

$= 240$