DESIGN AND ANALYSIS OF ALGORITHMS
BSE: 5th SEMESTER
(After Mid)

21/11/24

→ Insertion Sort : $O(n^2)$

→ Merge Sort : $\Theta(n \log n)$

⊙ **Space Complexity:**

(i) Inplace sorting Algorithm (use that space of specific Data structure)
(Do sorting within that structure)
(Insertion sort)

(ii) Not in Place sorting Algorith (Merge sort)

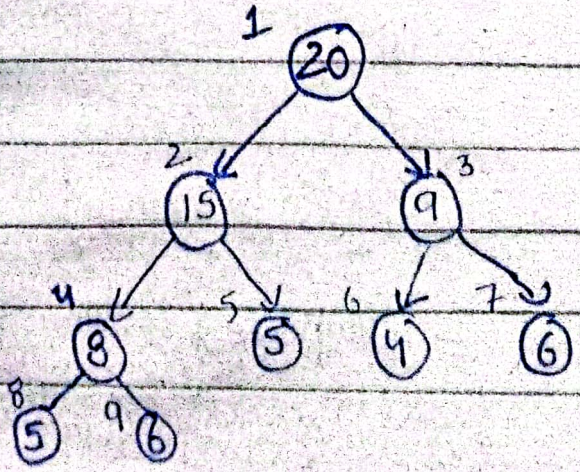
⊙ **Heap:**

→ Tree Data Structure - Heap

→ Binary Tree Visualization

→ A
(Storage like Array)

A.length
A.HeapSize
(less than length)



A

20	15	9	8	5	4	6
1	2	3	4	5	6	7

: left child - 2i (Greater)
Right child - 2i+1 (less)

: Complete Binary Tree (Either 2 child or leaf node)

→ $A[parent(i)] \geq A[i]$

: $Parent(i) = \lfloor i/2 \rfloor$

$Left(i) = 2i$

$Right(i) = 2i+1$

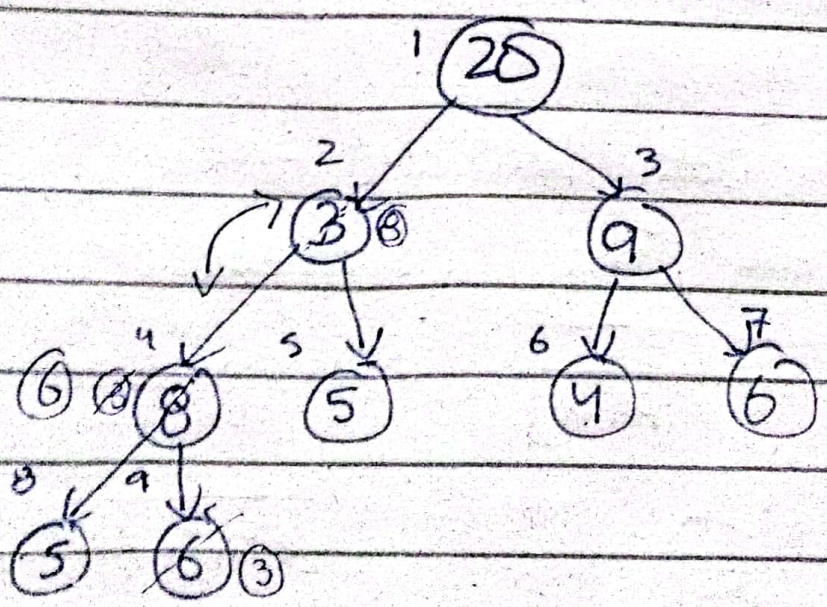
: Max or Min Heap

→ MaxHeapify (A, i) (A: Array, i: Node)

A

20	15	9	8	5	4	6	5	6
1	2	3	4	5	6	7	8	9

MaxHeapify (A, 2)



: Root Node should be greater than other two.

: Write algorithm.

: Find Time Complexity

→ Max Heapify (A[n], i)

{ While (i < heapsize)

→ Max Heapify (A, i)

{

: Two comparison
: One swapping

Max_Heapify();

}

: Not Specified that Tree is equally divided

$: Tn = aT\left(\frac{n}{b}\right) + C$

$Tn = 1\,T\left(\frac{n}{2}\right) + \Theta(1)$

$O(\log n)$

$: n^{\log_b a}$

$: n^{\log_b 1} = n^0 = 1$

$: 1.\log n$

⊙ Build Heap :

→ Random Array is given and MaxHeapify on each element.

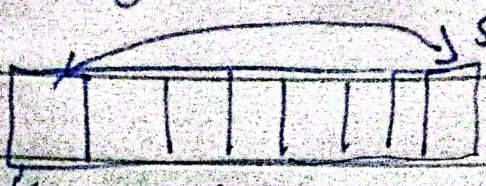
→ Where leaf nodes are present?

Not call MaxHeapify on leaf nodes.

⊙ Heap Sort :

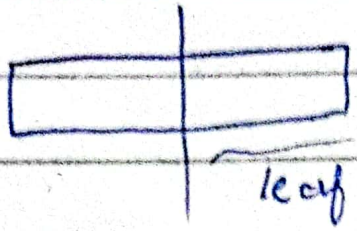
(nlogn)

→ Ascending order (Large value at end) (swap Root with last value and call max Heapify for now Root value)



: Overall Heapsort cost
$(n \cdot 1 \cdot n \log n)$
$O(n \log n)$

3/12/24

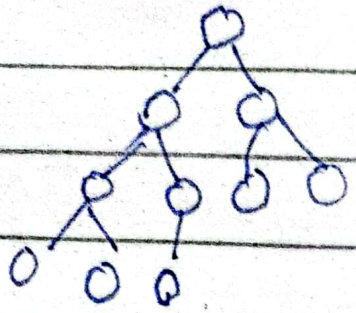


$n \log n$

$\frac{n}{2} \log n$

(*) It can further be made better from $n \log n$:

: Suppose:



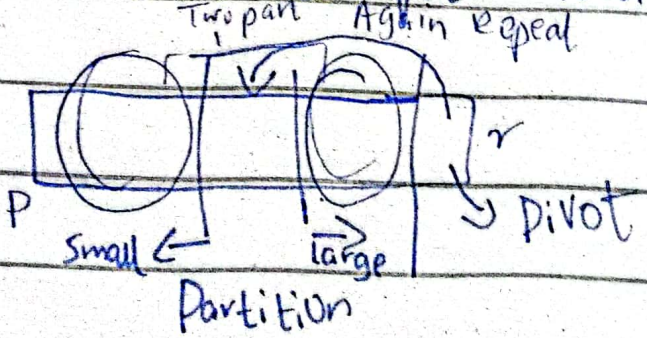
: When we go up the tree value gradually decrease

⊙ Quick Sort:

→ Sorting Algorithm

→ Divide and Conquer Rule

→



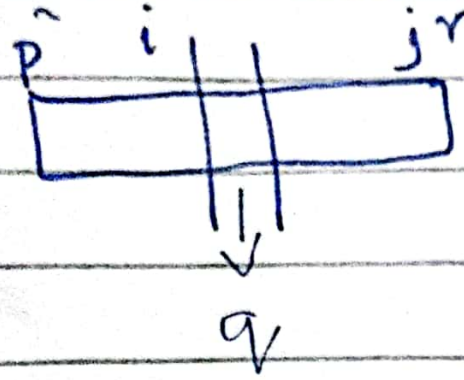
→ Main is Partition Algorithm.

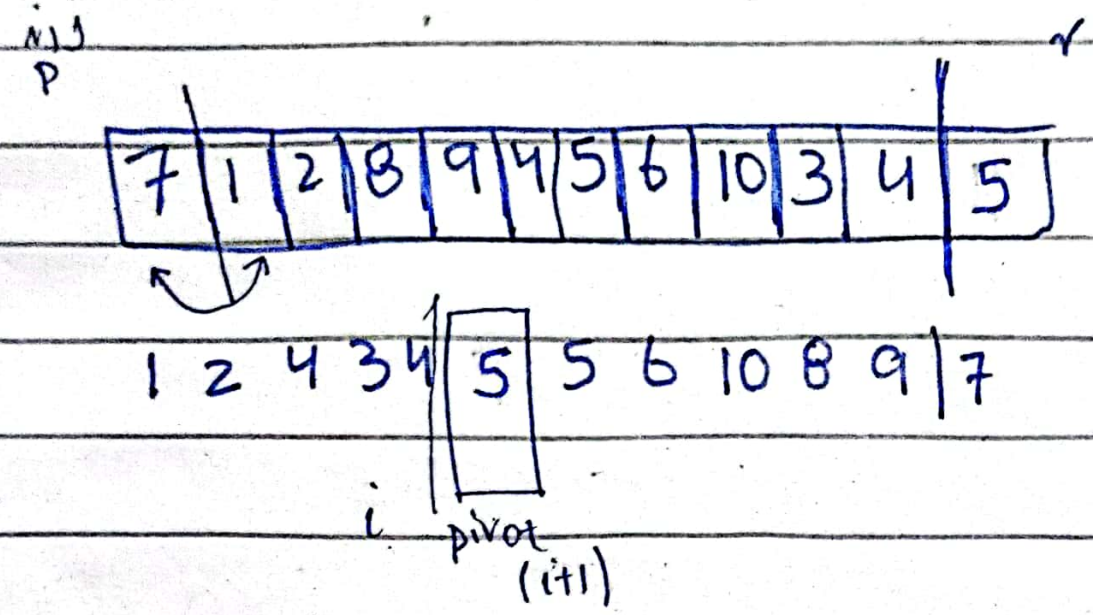
→ q = Partition (A, p, r)

Quicksort (A, p, q-1)

Quicksort (A, q+1, r)

↘ Main algorithm Pseudo code.





: Compare with 5
: If greater than keep there
: If less than swap and i float next
: j float continously

④ Strict Bounds: $\Theta(n)$

⊙ Quick sort (Complete Read it).
(Ex: 7.1 - 7.4) (Do it hand-written)

5/12/24

⊙ Performance Analysis of Quick-Sort:

→ Worst-Case Partitioning:

⊙ All problems shift on one side. Suppose on right there is no problem but maximum weight of problem is on left.

$$T(n) = T(0) + T(n-1) + \Theta(n)$$
$$= T(n-1) + \Theta(n)$$
$$\Theta(n^2)$$

⊙ Best-Case Scenario Partitioning:

→ When problems is divided into equal problem size two parts.

→ $T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)$

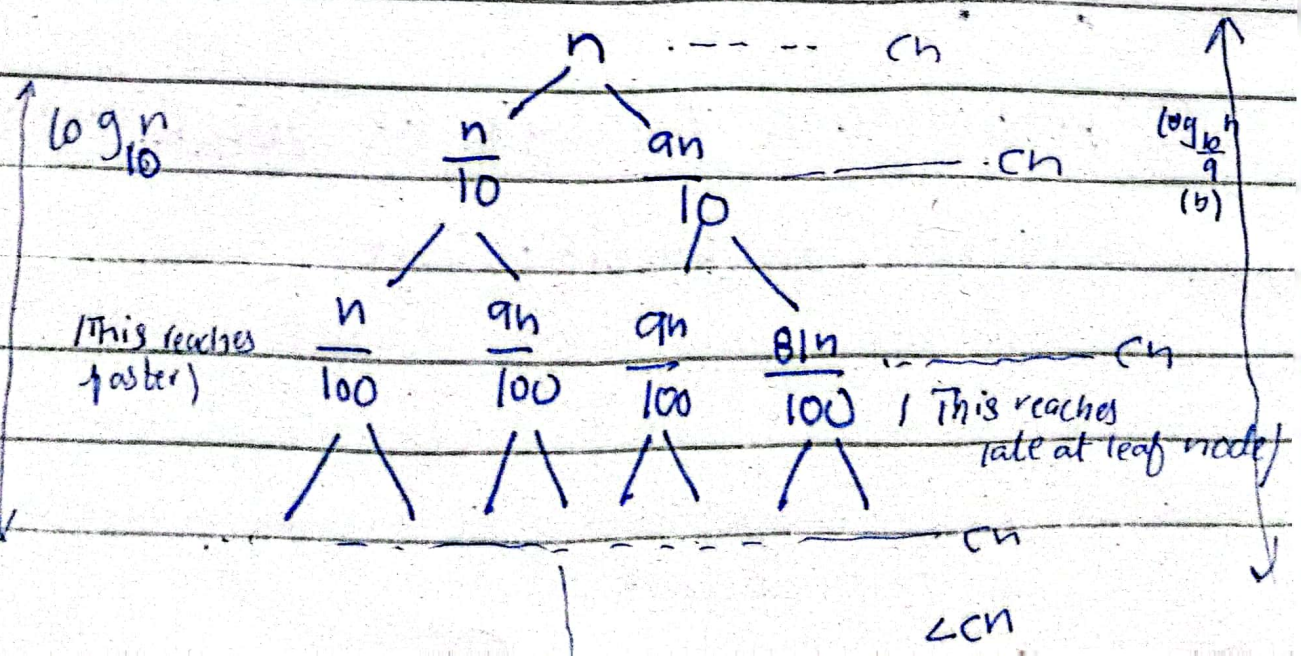
$\Theta(n \log n)$

⊙ Balance Scenario:

→ E.g there is ratio of 1:9 (10%, 90%)

: We are more concerned about worst-case

→ $T(n) = T\left(\frac{n}{10}\right) + T\left(\frac{9n}{10}\right) + \Theta(n)$



: $\Theta(n \log n)$

: $\leq_{(\text{Big O})} cn \log_{\frac{10}{9}} n$: $\log_{10} n$

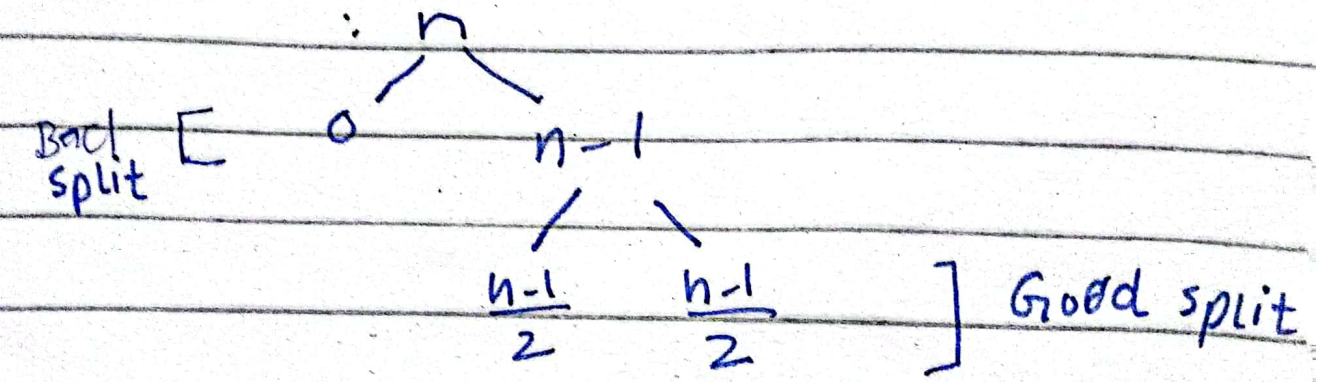
: $\log_{\frac{10}{9}} n$

$\boxed{\Theta(n \log n)}$ (log Base 2)

∴ : But Asymptotically it is $n \log_2 n$ (Why base 2) : (So Prove it)

OFor Average Case:

→ It is mix of good and bad split



→ Bad split followed by good split

: $\Theta(n) + \Theta + \Theta(n-1)$

: Choose random number (pivot) then swap it.

: We finally : intuite that $n \log n$

→ Worst Case Analysis:

$$T(n) = \max T(q) + T(n-q-1)$$

$$0 \leq q \leq n-1$$

(By substitution)

(It's guess would be worst-case (n^2)

$$\leq \max_{0 \leq q \leq n-1} \{ cq^2 + c(n-q-1)^2 \}$$

$= c \cdot \max \left(q_1^2 + (n - q_1 - 1)^2 \right) + \Theta(n)$

maxima (First Derivative → Equal to zero
Second Derivative ← Stationary points
It is positive or negative
(max) (min))

$\rightarrow c(n-1)^2 + \Theta(n)$

$\rightarrow cn^2 - 2n + c + \Theta(n)$

$\rightarrow cn^2 - c(2n-1) + \Theta(n)$

(Big) - Small = Smaller values

$< cn^2$

$T(n) = O(n^2)$

: Precise
Average
Late Analysis
(Randomized)

10/12/24

O(nlogn) 1:1 (Perfect / Equal)

O(nlogn) 1:99 (Different / Bad Balance)

(But performance is same)

→ BigO is upper bound so 1:1 and 1:99 will be below constant factor nlogn but not ~~similar~~ same. according to performance and can be similar (growth rate) as belong to similar family.

: T(n)

$0 \leq f_a(n) \leq c_1 n \log n$

$0 \leq f_b(n) \leq c_2 n \log n$

⊙ Average Case / Expected Case :

→ Under all circumstances how average case | Expected case works.

→ On average it should be mix of good split and bad split but not exact clear.

→ Random variable possible outcomes. $\boxed{E(x) = \sum_{i=1} x P(x)}$

(1 of 6 dices rolled)

→ On average partition would be prediction.

→ Partition would be better if we got median in less than (nlogn)

→ Arrangement possible would be found by permutation.

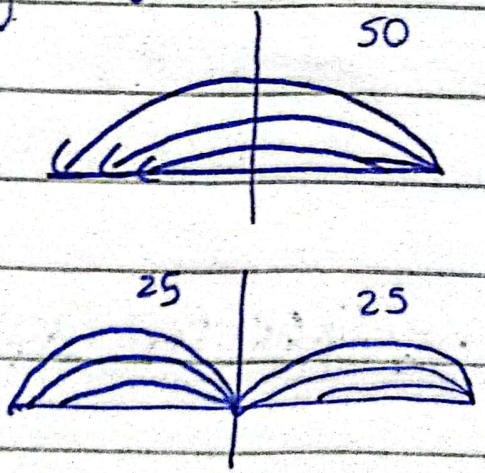
→ In Algorithm computational steps are:

(i) Comparison of values

(More comparison, more computational steps)

④ In Bubble sort, comparison are easily found

→

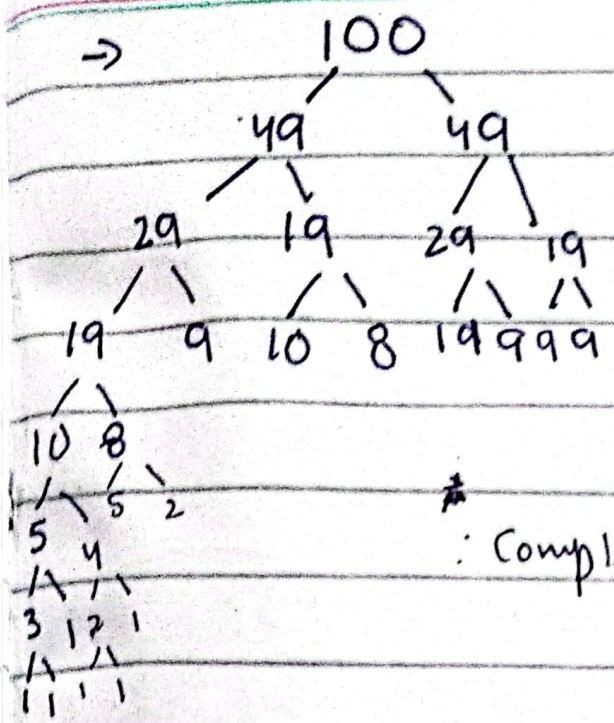


: Size: 50
How many comparison in Bubble sort
: Also for QuickSort

12/12/24

No. of elements	Comparison
100	99
(77) 78 (21)	76+20=96
(55) 56 (20) (12) 13 (7)	54+19+11+6=90
(33) (20)	

→



Comparison
99
96

* : Complete it

(*) z_{ij} $z_i, z_{i+1}, z_{i+2}, \ldots, z_j$

X_{ij} z_i is compoired with z_j

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}$$

: Expected $$E(X) = E\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij} \right]$$

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E(X_{ij})$$

$Pr($ z_i is compaired with z_j $)$

(z_i is chosen as pivot and compaired with z_j)

(z_j is chosen as pivot and compared with z_i)

$$= \frac{1}{j-i+1} + \frac{1}{j-i+1}$$

$$= \frac{2}{j-i+1} \qquad \because (k = j-i)$$

$$\sum_{i=1}^{n-1} \sum_{j-i=1}^{n-i} \frac{2}{j-i+1}$$

$$= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} \leq \sum_{i+1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k}$$

$$\therefore O(n \log n)$$

⊙ Dynamic Programming:

: Bucket sort

: Asymptotic Recurrence (Analysis)

: Divide & conquer (Design)

→ This technique is also related to Divide and conquer.

→ Some unnecessary calculations which increases the Time Complexity. (May be Exponential: 2^n)

→ Dynamic Programming teaches use the repeating thing one time For the scenario dynamic programming handles better are called optimization problem. (Maximize and Minimize)

: Ch#15

→ Rod Cutting:

i	1	2	3	4	5	6	7	8	9	10
Pi	1	5	8	9	10	17	17	20	24	30

: Maximize

: More than one optimization solutions.

→ 4 m = 9

1, 3 = 9

2, 2 = 10 · Best

3, 1 = 9

1, 1, 2 = 7

1, 2, 1 = 7

1, 2, 1 = 7

2, 1, 1 = 7

1, 1, 1, 1 = 4

: Size = n
So From (1-n) (No cut)

$n = i_1 + i_2 + \dots + i_k$

(rate/revenue) $r_n = P_{i_1} + P_{i_2} + \dots + P_{i_k}$

$r_1 = 1$ (No cut)

$r_2 = 5$ (")

$r_3 = 8$ (")

$r_4 = 10$ (2, 2)

$r_5 = 13$ (2, 3)

$r_6 = 17$ (~~1, 2, 3~~) (No cut)

$r_7 = 18$ (2, 5)

$$\max(P_4, P_1+P_3, P_2+P_2, P_3+P_1, P_1+P_1+P_2 \ldots)$$

→ Generalized Expression:

$$r_n = \max_{1 \le i \le n} (P_i + r_{n-i})$$

:15.1 (Pg#390) Before 15.2

17/12/24

* Doing unnecessary steps recursively will increase time complexity and for this we use dynamic programming.
* In general, dynamic problem is best for optimization.
* Revenue compute for 400 m Long Rod ((1,399), (2,388))

*

$$r_n = \max(P_n, r_1+r_{n-1}, r_2+r_{n-2}, \ldots, r_{n-1}+r_1)$$

: r: revenue
n: Length

(Generic Expression)

E.g: $n=2$

$$r_2 = \max(P_2, r_2+r_1)$$

: Considering rod with no cut to smaller cut

$n=3$

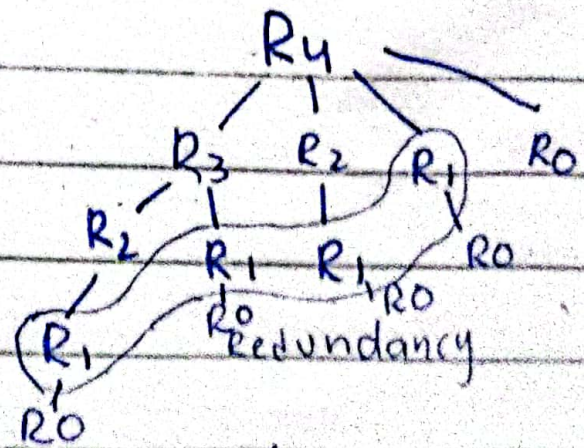
$r_3 = \max(p_3, r_1+r_2, r_2+r_1)$: Brute force (All combination)

(*) $r_n = \max_{1 \le i \le n} (p_i + r_{n-i})$: $p_1 + r_{n-1}$: $p_2 + r_{n-2}$

(*) Pseudo code:

```
R_n ( P, n )
if(n==0)
    return 0;
q = -∞
for i = 1 to n
    q = max( q, P[i] + Rod_cut(P, n-i) )
return q
```

$R_4 \begin{cases} R_3 \\ R_2 \\ R_1 \end{cases}$



$\rightarrow T(n) = \begin{cases} 1 & T(n) = 1 + \sum_{j=0}^{n-1} T(j) \end{cases}$

$T(0) = 1$
$T(1) = 2$
$T(2) = 4$
$T(3) = 8$
$T(4) = 16$
$T(n) = (2^n)$ (complexity)

: $S_n = a\left(\frac{1-r^n}{1-r}\right)$

⊙ Two-Approaches

(i) Top-Down (Memorization) (entries is needed to be computed)

(ii) Bottom-Up (starts from 1) (All are covered in it)

→ We need to store results and need data structure.

→ Space is increasing but Time is decreasing.

→ $P = [1 \ldots n]$

$P_i \rightarrow r = [0 \ldots n]$

r_i

: Top-Down :

$R_n((P, n), r)$

if $n == 0$

$q = 0$

if $r_n > 0$ return r_n; else

$q = -\infty$

for $i = 1$ to n

$q = \max(q, P_i + R_{n-i})$

$r_i = q$;

: $O(n^2)$ (Nesting affect)

: Bottom-up :

→ More simpler starts from 0

$T[0]$

$T[1]$

$T[2]$ —

: Matrix Chain Multiplication

: IS-2 (Check for it)

19/12/24

⊙ Matrix-Chain Multiplication:

→ Matrix (2-D structure)

$$[M_1]_{(r_1 \times c_1)} \times [M_2]_{(r_2 \times c_2)} = [\ ]_{(r_1 \times c_2)}$$

∴ ═══ (Three loops for multiplication)

$$\Theta(n^3)$$

$: r_1 \times c_1 \times c_2$ (Cost formula)

→ **Associativity:**

$$(A_1 A_2) A_3 = A_1 (A_2 A_3)$$

A_1 (100×100) r_1 c_1 — A_2 (100×5) r_2 c_2 — A_3 (5×50) r_3 c_3

→ $(A_1 A_2) A_3$

$= 5000$

$10 \times 5 \quad 5 \times 50$

$= 2500$

$5000 + 2500 = 7500$

$A_1 (A_2 A_3)$

$= 25000$

$10 \times 100 \quad 100 \times 50$

$= 50000$

$= 75000$

→ Multiplication sequence is judged by parenthesis.

⇒ **For three matrices:**

(i) $(A_1 (A_2 A_3))$ (ii) $((A_1 A_2) A_3)$

→ For Four Matrices:

A_1, A_2, A_3, A_4

(i) $((A_1 A_2) A_3) A_4$: Total 5

(ii) $(A_1 (A_2 A_3)) A_4$

(iii) $(A_1 (A_2 (A_3 \times A_4)))$

(iv) $(((A_1 A_3) A_2) A_4)$: Minimum cost

(v) $(((A_1 A_4) A_3) A_2)$.

→ (It is optimization Problem best for dynamic programming)

→ Minimization.

* After all multiplication, last step also contains two matrix multiplication:

P_1 P_2

(P_1 = Sub Matrix Result) (P_2 = Last Matrix)

(This sequence can be changed)

→ Possibilities

$$P(n) = \begin{cases} 1 & n=1 \\ \sum_{K=1}^{n-1} P(K)\, P(n-K) & n \geq 2 \end{cases}$$

$\downarrow$

2^n.

⊛ Guess: $O(2^n)$

$T(n) = 2^n$

$T(n) \leq c \cdot 2^n$

$P(k)\ (P(n-k))$

$c \cdot 2^k \left(c \cdot 2^{n-k}\right)$

$c \cdot \left(2^k \cdot \frac{2^n}{2^k}\right)$

$c \cdot 2^n$

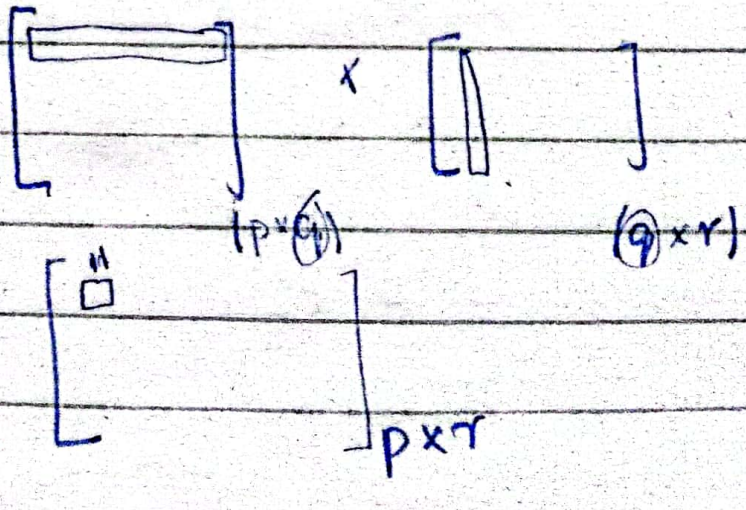
$c \cdot 2^n \leq c \cdot 2^n$ (Proved)

31/12/24

⊙ Chain-Matrix Multiplication:

→ Matrix property of Associativity

$$A_1(A_2A_3) = (A_1A_2)A_3$$



→ Way to parenthesize n Matrix.

$A_1, A_2, A_3, \ldots, A_n$

: $(A_1)(A_2 A_3 \ldots A_n)$

$(A_1 A_2)(A_3 \ldots A_n)$

$(A_1 A_2 A_3)(\ldots A_n)$

$(A_1 A_2 A_3 \ldots)(A_n)$

: Not Proper Approach

: Sub-problems are of overlapping nature

⊙ $P(n) = \begin{cases} 1 & n=1 \\ \sum_{k=1}^{n-1} P(k)(P(n-k)) & n \geq 2 \end{cases}$

→ Catalin Number = $C(n) = \Omega\left(4^n / n^{3/2}\right)$

$P(n) = C \cdot (n-1)$

→ Optimal substructure (In dynamic programming, there will be overlapping while in divide and conquer there will be distinct problems).

⊙ $A_{i..j}$ (products of all Matrix from i till j)

$A_1 A_2 \ldots (A_i \ldots A_j) \ldots A_n$

(A_i A_j)

→ A_1 Dimension = $P_0 P_1$

A_2 " = $P_1 P_2$ $\quad 1 \leq i \leq j \leq n$

$A_{i..j} = P_{i-1} \cdot P_j$

→ Substructure → optimal substructure → Build a Recursive solution

$A_i \cdots\cdots A_j$: $A_{i..j}$

↑ Optimal split (K)

(Thinking that it is optimal, its substructure will also be optimal)

$A_{i..j} = (A_i, A_{i+1}, \ldots A_K)(A_{K+1}, A_{K+2} \ldots A_j)$

$i \le K \le j$

→ $A_{i..j} = m[i, j]$ (Optimal number of computations to compute scalar multiplication)

(Scalar Multiplication)

$= (A_i, A_{i+1}, \ldots, A_K)(A_{K+1}, A_{K+2} \ldots A_j)$

$m[i,K]$ $\quad$ $m[K+1, j]$

$A_{i..K}$ $\quad$ $A_{K+1..j}$

$(P_{i-1} \times P_K)$ $\quad$ $(P_K \times P_j)$

Dimension → $P_{i-1} \times P_K \times P_j$

$$m[i,j] = \begin{cases} 0 & i = j \\ \min\limits_{i \le K < j} \{ m[i,K] + m[K+1,j] + (P_{i-1} \times P_K \times P_j) \} & i < j \end{cases}$$

→ Bottom-up approach for optimization

A_1 A_2 A_3 A_4 A_5 A_6

P_0 P_1 P_1 P_2 P_2 P_3 P_3 P_4 P_4 P_5 P_5 P_6

→ $m[1,2] = m[1,1] + m[2,2] + P$

→ $m[2,5] = \begin{matrix} 2 \\ 3 \\ 4 \end{matrix} \begin{matrix} m[2,2] + m[3,5] \\ m[2,3] + m[4,5] \\ m[2,4] + m[5,5] \end{matrix}$

$A_{2..5}$

	1	2	3	4	5	6
1	0	15750	7875	9375	11,875	15125
2		0	2625	4375	7125	10500
3			0	750	2500	5375
4				0	1000	3500
5					0	5000
6						0

2/1/25

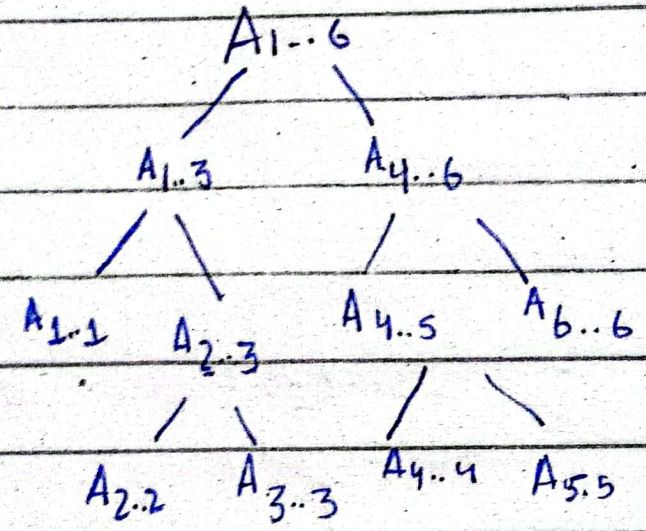
→ $A_1 \times A_2 \times A_3 \times A_4 \times A_5 \times A_6 \times \cdots \times A_n$

K-Table

	1	2	3	4	5	6
1				3	3	3
2			2	3	3	3
3				3	3	3
4					4	5
5						5
6						

6x6

→



(A1) (A2) (A3) (A4) (A5) (A6)

⊙ Greedy Algorithm:

→ Solves problem with more speed than dynamic programming

→ This approaches us to pick the choice of min and max by ourselves.

→ Dynamic programming gives us global optimal value while in Greedy Algorithm we get local optimal which belongs to global.

→ Its not necessary that it can be applied on all problems but can be checked.

→ Activity Selection Problem:

: List of activities

$(a_1, a_2, a_3, \ldots\ldots, a_n)$ ... a_i :: a: activity

$s_1, s_2, s_3 \quad s_n \quad s_i$ S: Start

$f_1 \; f_2 \; f_3 \quad f_n \; f_i$ F: Finish

⊙ Maximum set of compatability activities.

(Independent, Not overlapping, Distinct)

⊙ $[s_i, f_i)$

$a_j \; [s_j, f_j)$

$s_i \geq f_j$

$s_j \geq f_i$

→ i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

$\{a_1, a_4\}$ $\{a_1, a_4, a_8, a_{11}\}$, $\{a_2, a_4, a_8, a_{11}\}$

Compatability activities

S_{1n} $a_1, a_2, a_3, \ldots\ldots, a_n$

S_{ij} A_{ij} (solution of S_{ij}) (Maximum set of compatability)

(a_k) (optimal activity)

S_{ik} S_{kj}

A_{ik} A_{kj}

→ Chapter #16 (16.1 (Activity Selection Problem), Elements of Greedy) *Quiz (16.1)

7/1/25

→ In Dynamic Programming, there is option of backtracking while in Greedy no backtracking.

→ Sometimes Greedy don't work to give optimal value (Heuristic)

→ If we wanted to say Greedy Algorithm is Greedy and working properly, (we get that local optimal value belongs to global or not (it is Greedy or not)).

⊙ Local optimal obtaining from Greedy strategy at ~~that~~ stage what is obtained.

⊕ For obtaining local optimal, we use Greedy strategy, less (earliest finish time) activity time will give us more time to do other activities.

→ It will become Greedy algorithm when optimal value from Greedy strategy or approach will be part of global part.
: Global also will contain local optimal value to be Greedy.

→ That decision will be part of Global solution, then it will be Greedy.

→ Graphs:

(Data structure contain vertices, edges, arcs)
(Directed or undirected edge

⊙ Minimum Spanning Tree (MST):

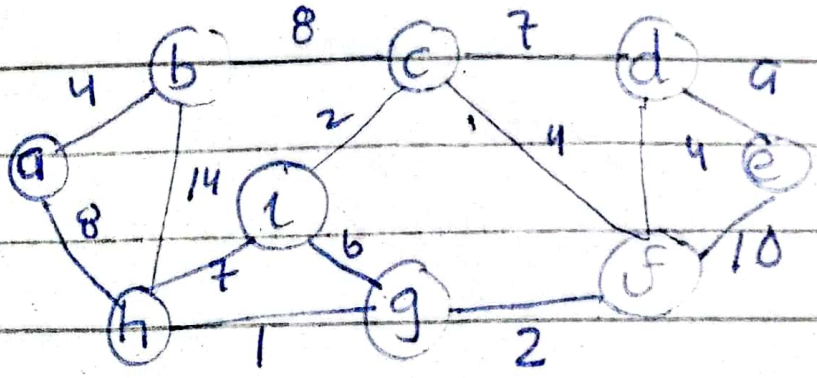
→ Graph can be represented:
(i) Adjacency Matrix
(ii) Adjacency List

→ Graph is a set of vertices and Edge : $G : (V, E)$

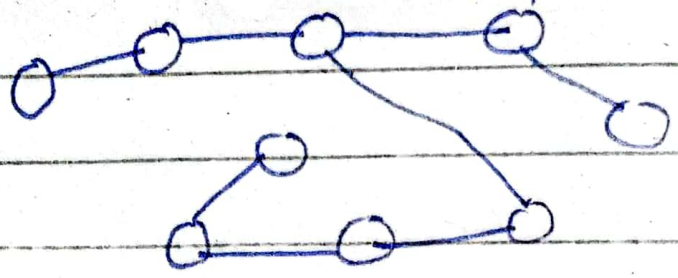
: MST
(5 Real Life Application (Description))

: Tree: Only one (No cycle) way to other node
: Graph: Multiple paths.

: Trees: No cycle
Graph: May or may not cycle.



-> Find tree from graph which includes all nodes. (Spanning Tree)

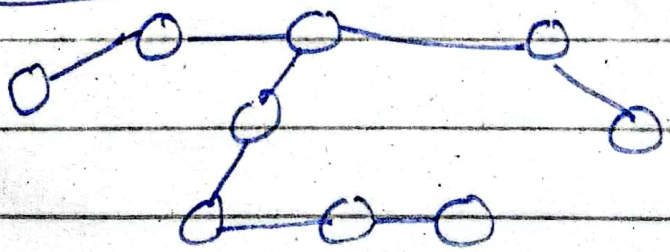


W=37

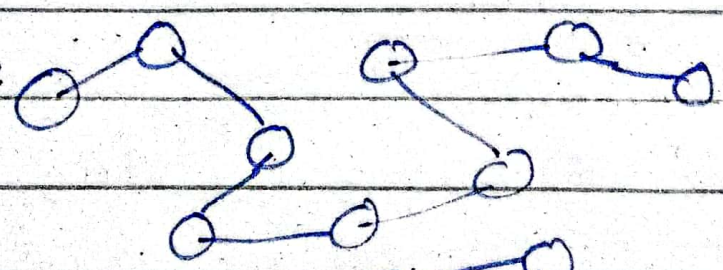
-> Minimum ST (whose edges are minimum and gives optimal value)
: Weight is overall minimized

-> Other ST:

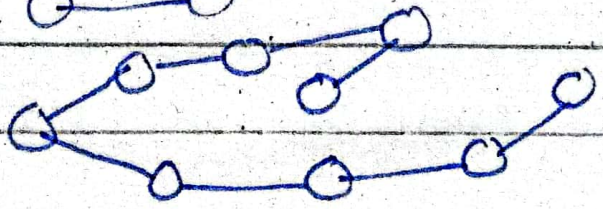
:



:



:

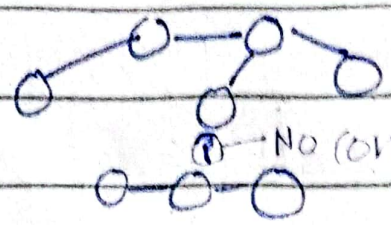


→ Free Tree (where root node is not designated)

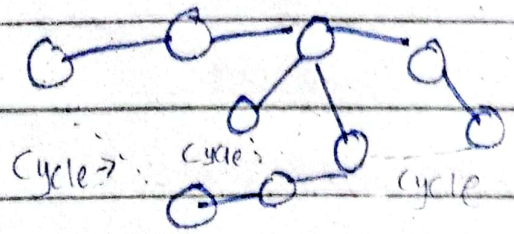
: N: Node

Edges = N-1

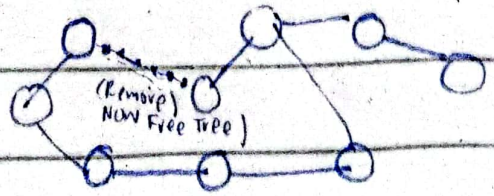
→ Forest (If Multiple trees are there:



→ If we add one edge more in generalization, then it will be cycle.



→ If we remove one edge from cycle it will be tree.



→ E(Set of Edges), A(Subset of Edges)
(All possible edges of a Tree) (Pair of MST)

: There can be variable many MST.

Ch#23 (MST)
(Complete Read it)

→ Subset of edges, which are part of MST (Viable set)
: Viable set of edges are edges that cannot make cycle.

→ More than one possible MST can occur because of similar weights.

→ Edge (U,V) → Safe Edge if after puting this edge in A. it will remain viable set.

: viable is related to cycle. (No cycle should be formed) (Greedy)

→ Steps:
(i) Take Empty set.
(ii) condition while MST terminates.
(iii) Selection of (U,V) edge and check it viability.
(iv) Repeat.

* Example:
Graph of start:

$E = \{(a,b), (b,c)$

$A = \{(a,b), (b,c), (c,d), (d,e),$
(Fig)

⊙ Application of MST:

(i) Network Design
(ii) Road and Railway Construction
(iii) Electrical Grid Design
(iv) Cluster Analysis in Data Mining.
(v) water supply Systems.

9/1/25

E - Set of edges : Edges Talk
A - Subset Minimum Entry of Edges (Part of MST)

→ Verify Safe Edge and add it to the 'A'.

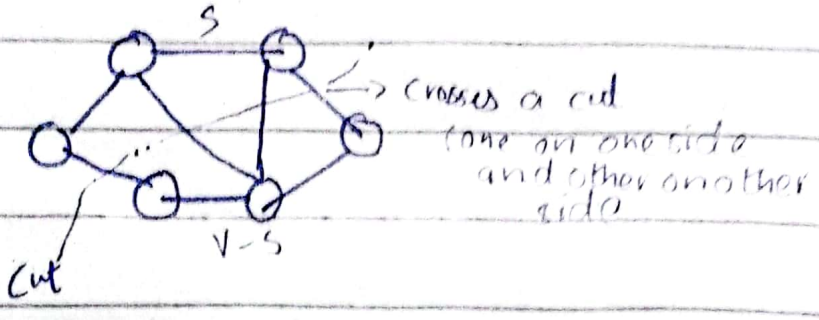
→ Minimum edge would be picked and repeat process and the global solution will be optimized.

→ **Theorems:** : Now vertices Talk

⊙ **Cut:** Distributing the vertices into two disjoint sets is called cut.

⊙ If one vertex is S, then othe will be V-S

(S, V-S)



⊙ (cut) It respects the A, if because of cut, all edges of A are on one side.

: If one edge is one side and on other side also, so it donot respect A.

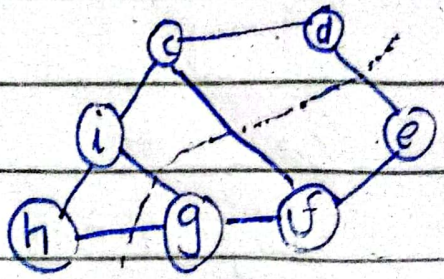
⊙ safe edges will be that edges which are crossing the cut.

⊙ we pick cut respects A, then find safe edge and that will be which are on cut.

⊙ Light edge: The cut which respects A, weight is less than than, this safe edge will be light edge.

⊙ Proof by Contradiction:

→ suppose T is tree and w is minimum weight and this is not on cut side, so we will prove it by contradiction (Prove it wrong)



(i,g) , (h,g)

→ If there is cycle and we cut then there would be even edges on cut side below.

→ If edge is added, the weight of added edge is less than removed edge. Then, it will not be MST before.

$w(e_1) < w(e_2)$

⊙ **Kruskal's Algorithm**: defines disjoint set data structure, in which we:

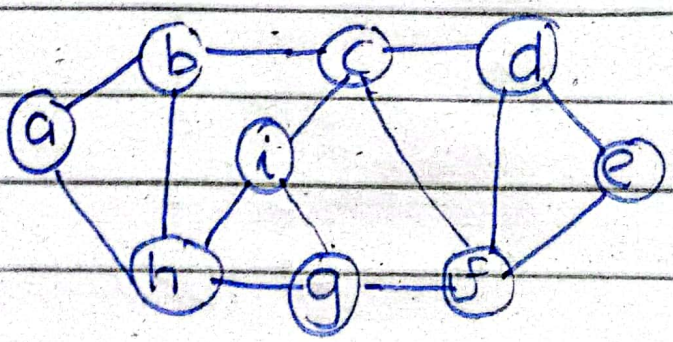
(i) Create set (Create_Set())
(ii) Find set (Find_set())
(iii) Union set (Union_set())

: less than $O(\log n)$

→ {a, b, c}

{d, e}

{f, g, h, i}



→ Steps:

1) Create Disjoint Against Vertices
2) Sort each Edges on basis of weight (In Asconding Order)
   → We should have viable set which is empty in start:

$$A = \phi$$

→ If we apply on edge vertices find-set(), it would be selected

: Kruskal (Make Forest & merge in last)
: Prism (In start make true)

if it would be part of distinct trees (so no cycle).

→ After selecting edge, we would join 'h' or 'g', so we call Union_set() function.

⇒ Then we add it to viable set:

A = { (h,g) ....

(Now we repeat this step)

for (c,i)

→ A = { (h,g), (c,i)

: Now we add {f,g,h}

→ A = { (h,g), (i,k), (g,f), (a,b), (c,f), (c,d) }

: Overall Running Time will be O(ElogE).

→ Prims Algorithm Read by yourself and included in exam.

→ MinHeap and Priority Queue will be part of it.