Jai Shree Shyam

DAA - GS

Prepared By - Manny Garg

Lec - 1

DAA Syllabus

1) Asymptotic Notation

2) Time and space complexity

3) Divide and conquer, All sorting algos, Heap Tree

4) Greedy Methods (Job Sequencing, Knap sack, optimal Merge Pattern, Huffman encoding, Dijkstra's, MST)

5) Graph Traversal (DFS, BFS, Connected components)

6) Dynamic Programming (All Pair Shortest Path, Multistage ~~Path~~, graph, optimal Binary Search tree, TSP, 0/1 Knap sack, LCS, Matrix Chain, Multi, SOS)

7) Hashing

8) P, NP, NPH, NPC

Lec-2

What is Algorithm

⇒ Finite set of steps to solve a problem is called Algorithm.

→ Analysis is process of comparing two algo's w.r.t time, space, etc.

→ Analysis

Priory Analysis	Posterior Analysis
• It give Approximate value of time.	• It ~~is~~ gives Exact value of time.
• It is independent of hardware.	• It is dependent of Hardware.
• It give uniform value.	• It gives non-uniform value.

Priory is good.

Lecture-3

Asymptotic Notations

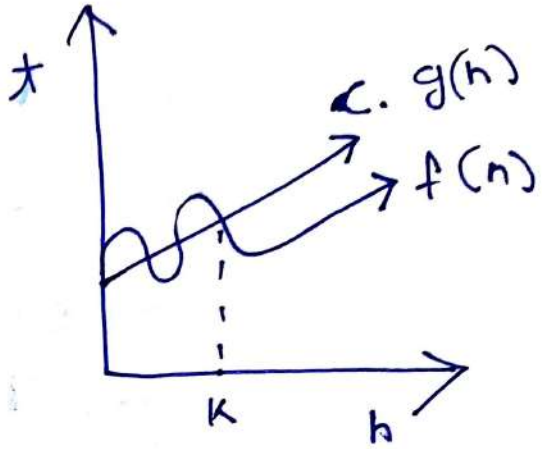
1) Big Oh (O)

~~$f(n) = O(g(n))$~~ $f(n) = O\, g(n)$

$f(n) \le c \cdot g(n)$

$c > 0$

$n \ge k \quad k \ge 0$



→ worst case

→ Least upper bound

Let $f(n) = 2n^2 + n$, $f(n) = O(\ ) = ?$

$f(n) \le c \cdot g(n)$

$2n^2 + n \le c \cdot g(n)$

$g(n) = n^2$

$\Rightarrow 2n^2 + n \le c \cdot n^2$ — (1)

min value of $c = 3$, so that (1) be true

$2n^2 + n \le 3n^2$

$n \le n^2$

$1 \le n$

$\therefore n \ge 1$

$\therefore$ for all $n \ge 1$ it is true

and $f(n) = O(n^2) \quad n \ge 1$

) Big Omega (Ω)

→ Best case

→ Greatest Lower bound

$$f(n) = \Omega g(n)$$

$$f(n) \geq c \cdot g(n)$$

$$2n^2 + n \geq c \cdot n^2$$

max value of $c = 2$, to be true

$$2n^2 + n \geq 2n^2$$

$$n \geq 0$$

$\therefore$ for $\forall n \geq 0$ $f(n) = \Omega(n^2)$

③ Big Theta (Θ)

→ Average case → Exact time

$$c_1 \cdot g(n) \leq fn \leq c_2 \cdot g(n)$$

$$2n^2 \leq 2n^2 + n \leq 3n^2$$

Lec-4

Various Properties of Asymptotic Notations

	Reflexive	Symmetric	Transitive
Big Oh (O)	✓	✗	✓
Big Omega (Ω)	✓	✗	✓
Theta (Θ)	✓	✓	✓
Small (o)	✗	✗	✓
Small (ω)	✗	✗	✓

Lec- 5

Comparison of various Time Complexities

$$O(1)$$

$$O(c) < O(\log\log n) < O(\log n) < O(n^{1/2}) < O(n) < O(n \log n) <$$

$$O(n^2) < O(n^3) < O(n^k) < O(2^n) < O(n^n) < O(2^{2^n})$$

Lec-6

Time complexities of all searching and sorting algorithms.

1. Binary Search - $\log_2 n$ in Avg, worst case | Best case = $O(1)$
2. Sequential Search - $O(n)$ in Avg, worst case | Best case = $O(1)$
3. Quick Sort - $O(n \log n)$ in best, Avg case | worst = $O(n^2)$
4. Merge Sort - $O(n \log n)$ in all cases
5. Insertion Sort - $O(n^2)$ in Avg, worst | Best = $O(n)$
6. Bubble Sort - $O(n^2)$ ~~in Avg, worst | Best = O(n)~~ in all cases
7. Heap Sort - $O(n \log n)$ in all case
8. Selection Sort - $O(n^2)$ in all case

9. Height of CBT $\rightarrow O(\log n)$ in all cases (complete binary tree)

10. Insertion in heap - $\log(n)$

11. Construct heap - $(n \log n)$

12. Delete from heap - $(\log_2 n)$

13. Huffman $(n \log n)$

V → Vertex
E → Edges

→ Prims Matrix - $O(n^2)$, if using heap → $O[(E+V)\log V]$

→ Kruskal - $O(E \log E)$

→ DFS, BFS - $O(V+E)$

→ All Pair Shortest - $O(n^3)$

→ Dijkstra → $O(V^2)$

Lec - 7

Questions of comparison of various Time complexities.

Q1 $f_1(n) = n^2 \log_{10} n$ $\quad$ $f_2(n) = n(\log_{10} n)^{10}$

choose correct option

(a) $f_1(n) = O\, f_2(n)$ $\quad$ ✓ (c) $f_2(n) = O\, f_1(n)$

(b) $f_1(n) = \Omega\, f_2(n)$ $\quad$ (d) $f_2(n) = \Omega\, f_1(n)$

⇒ $f_1(n) = n^2 \log_{10} n$ $\quad$ $f_2(n) = n(\log_{10} n)^{10}$

Let $f_1(n) = f_2(n)$

$$n^2 \log_{10} n = n(\log_{10} n)^{10}$$

$$n \cdot \cancel{n \log n} = \cancel{n \log n}\,(\log n)^9$$

$$n = (\log n)^9$$

Take log both side

$$\log_{10} n = \log_{10} (\log_{10} n)^{9}$$

$$\log n = 9 \log(\log n)$$

As we know

T.C of $\log n > \log(\log n)$

$\therefore$ T.C of $f_1(n) > f_2(n)$

$\therefore$ $f_2(n) = O\, f_1(n)$

$f_2(n) \leq c.f_1(n)$

Lec - 9

What is Recurrence Relation?

⇒ which repeats itself.

→ Binary Search Recurrence Relation.

```
BS(a, i, j, k)
  mid = (i + j)/2;
  if(a[mid] == k)
    return mid;
  else
    if (a[mid] > k)
      BS(a, i, mid-1, k)
    else
      BS(a, mid+1, j, k)
```

Suppose
array = 10 20 30 40 50 60 70

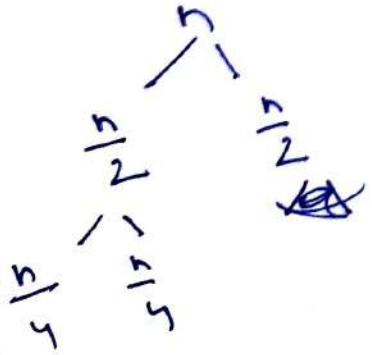
i = 10
j = 70

Suppose k = 30
i.e k = ~~for~~ searching element

Here we divide Problem into two parts.

Recurrence relation is

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



Lec- 10

Solution of Recurrence Relation of binary Search using substitution method. ie Find T.C of Recc. Relation.

Recurrence Relation of binary Search

$$\Rightarrow \begin{cases} T(n) = T(n/2) + c & \text{if } n > 1 \\ \quad = 1 & \text{if } n = 1 \end{cases}$$

n = no of element in array

$$T(n) = T\left(\frac{n}{2}\right) + c \quad - (1)$$

If we put $n = \frac{n}{2}$

$$T\left(\frac{n}{2}\right) = T(n/4) + c \quad - (2)$$

Put $n = \frac{n}{4}$

$$T(n/4) = T(n/8) + c \quad - (3)$$

$\vdots$

Now Put (2) in (1)

we get

$$T(n) = T(n/4) + c + c$$

$$T(n) = T\left(\frac{n}{2^2}\right) + 2c \quad - (4)$$

Now Put (3) in (4)

$$T(n) = T\left(\frac{n}{8}\right) + c + 2c = T\left(\frac{n}{2^3}\right) + 3c \quad - (5)$$

now we find that we get

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

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

So next iteration will be

$$T(n) = T\left(\frac{n}{2^4}\right) + 4C$$

$$\vdots$$

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

To get $T(1)$ Put $n = 2^k$

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

$$T(n) = T(1) + kc$$

$$T(n) = 1 + kc.$$

To find value of k

Take log both sides

$$\log_2 n = \log_2 2^k$$

$$\log_2 n = k \log_2 2 = k$$

$$k = \log_2 n$$

$\therefore \quad 1 + \log_2(n)\, c$

1, c are constant term $\therefore$ avoid it

$T(n) = O(\log_2 n)$

$\therefore$ Time complexity of Recurrence Relation of binary Search is $O(\log_2 n)$

Lec-11

Example of Soln of Recurrence Relation by Substitution method.

function we have given is

$$T(n) = \begin{cases} 1 & \text{if } n=1 \\ n * T(n-1) & \text{if } n>1 \end{cases}$$

$T(n) = n * T(n-1)$ — (1)

Put $n = n-1$

$T(n-1) = (n-1) * T(n-1-1)$
$= (n-1) * T(n-2)$ — (2)

Put $n = n-2$

$T(n-2) = (n-2) * T(n-2-1) = (n-2) * T(n-3)$ — (3)

Now
Put (2) in (1)

$T(n) = n * (n-1) * T(n-2)$ — (4)

Put (3) in (4)

$T(n) = n * (n-1) * (n-2) * T(n-3)$.

⋮

upto $(n-1)$ steps

$\Rightarrow n * (n-1) * (n-2) * (n-3) \cdots T(n-(n-1))$

$\Rightarrow n * (n-1) * (n-2) * (n-3) \cdots T(1)$

$\Rightarrow n * (n-1) * (n-2) * (n-3) \cdots\cdots 1$

$\Rightarrow n * (n-1) * \quad '' \quad '' \quad \cdots \quad 3 * 2 * 1$

$\Rightarrow n * n\left(1-\frac{1}{n}\right) * n\left(1-\frac{2}{n}\right) * n\left(1-\frac{3}{n}\right) \cdots n\left(\frac{3}{n}\right) * n\left(\frac{2}{n}\right) * n\left(\frac{1}{n}\right)$

Now $\left(1-\frac{1}{n}\right), \left(1-\frac{2}{n}\right) \cdots \left(\frac{1}{n}\right)$ terms

are minority $\therefore$ remove

$\Rightarrow n * n * n * \cdots n$

$\Rightarrow n^n$.

So $T.C = O(n^n)$

Advantage of Substitution method

1. It give exact, correct ans
2. It can solve all types of problems.

Disadvantage is, it is mathematical calculative.

Lec-12

Example of Recc. Relation soln by Substitution method.

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2T\left(\frac{n}{2}\right) + n & n > 1 \end{cases}$$

$$T(n) = 2T(n/2) + n \quad -(1)$$

Put $n = n/2$

$$T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{4}\right) + \frac{n}{2} \quad -(2)$$

$$T\left(\frac{n}{4}\right) = 2T\left(\frac{n}{8}\right) + \frac{n}{4} \quad -(3)$$

Put (2) in (1)

$$T(n) = 2\left[2T\left(\frac{n}{4}\right) + \frac{n}{2}\right] + n$$

$$\Rightarrow 2^2 T\left(\frac{n}{2^2}\right) + 2n \quad -(4)$$

Put (3) in (4)

$$T(n) = 2^2\left[2T\left(\frac{n}{8}\right) + \frac{n}{4}\right] + 2n$$

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

in same way
next iteration will be

$$2^4 T\left(\frac{n}{2^4}\right) + 4n$$

$\vdots$

upto K steps

$$2^K T\left(\frac{n}{2^K}\right) + Kn$$

now put $2^K = n$ — (5)

$\therefore$ ~~$n \cdot T\left(\frac{2^K}{2^K}\right) + Kn$~~

$\Rightarrow nT\left(\frac{n}{n}\right) + Kn$

$nT(1) + Kn = n + Kn$

Now to find K
take log in (5)

$\log_2 2^K = \log_2 n$

$K = \log_2 n$

$\therefore T(n) = n + n\log_2 n$

As $n \log_2 n$ is dominating

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

Lec- 13

Ques on Recur. Relation ~~to~~ by Substitution method

$$T(n) = \begin{cases} 1 & , \text{if } n=1 \\ T(n-1) + \log n & , \text{if } n>1 \end{cases}$$

$$T(n) = T(n-1) + \log n \quad - (1)$$

$$T(n-1) = T(n-1-1) + \log(n-1)$$
$$= T(n-2) + \log(n-1) \quad - (2)$$

$$T(n-2) = T(n-2-1) + \log(n-2)$$
$$= T(n-3) + \log(n-2) \quad - (3)$$

Put (2) in (1)

$$T(n) = T(n-2) + \log(n-1) + \log n \quad - (4)$$

Put (3) in (4)

$$T(n) = T(n-3) + \log(n-2) + \log(n-1) + \log n$$

In same way
next iteration will be

$T(n-4) + \log(n-3) + \log(n-2) + \log(n-1) + \log n$

$\vdots$

upto k steps.

$T(n-k) + \log(n-(k-1)) + \log(n-(k-2)) + \log(n-(k-3))$
$- - - - + \log n$

Now Put $n-k=1$

$\therefore$

$\Rightarrow T(1) + \log(1+1) + \log(1+2) + \log(1+3) + - - -$
$\log n$

$\Rightarrow 1 + \log 2 + \log 3 + \log 4 + - - \log n$

$1 + \log(2 \cdot 3 \cdot 4 - - n)$

$1 + \log(n!)$

$\Rightarrow 1 + \log n^n$

$1 + n \log n$

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

Lec-14, 15

Recurrence Relation soln. by Master Method Theorem.

→ This method is faster than Substitution.

→ This method can't solve all types of problems.

We can solve problems which is given in the form $f(n) = n^k \log^p n$

$$T(n) = aT(n/b) + f(n)$$

$$a \geq 1 \quad b > 1$$

solution to this type of problem is

$$T(n) = n^{\log_b a} [U(n)]$$

$U(n)$ depends on $h(n)$

$$h(n) = \frac{f(n)}{n^{\log_b a}}$$

→ Relation between $h(n)$ and $U(n)$ is

if $h(n)$	$U(n)$
$n^r, r > 0$	$O(n^r)$
$n^r, r < 0$	$O(1)$
$(\log n)^i, i \geq 0$	$\frac{(\log_2 n)^{i+1}}{i+1}$

Lets do an example

① $T(n) = 8T(n/2) + n^2$

$a = 8$, $b = 2$, $f(n) = n^2$

Soln :-

$n^{\log_b a} \cdot u(n)$

$\Rightarrow n^{\log_2 8} \, u(n)$

$\Rightarrow n^3 \, u(n)$

$u(n)$ depends on $h(n)$

$$h(n) = \frac{f(n)}{n^{\log_b a}} = \frac{n^2}{n^3} = \frac{1}{n} = n^{-1}$$

$h(n)$ is in form of n^{γ} $\gamma < 0$ as $-1 < 0$

$\therefore u(n) = O(1)$

$\therefore T.C = n^3 . 1$

$\Rightarrow O(n^3)$

(2) $T(n) = T(n/2) + c$

$T(n) = T(n/2) + c$

$a = 1$, $b = 2$ $\quad f(n) = \text{constant } c$

$a \geq 1$, $b > 1$ $\therefore$ master method works

Soln :- $n^{\log_b a}\, u(n)$

$\Rightarrow n^{\log_2 1}\, u(n)$

$\Rightarrow n^0\, u(n) = 1 \cdot u(n)$

$u(n)$ depends on $h(n)$

$$h(n) = \frac{f(n)}{n^{\log_b a}} = \frac{c}{n^0} = \frac{c}{1} = c = \text{constant}$$

$$\Rightarrow (\log_2 n)^0 \cdot c$$

As it is in form of

$(\log n)^i$ and $i = 0$.

$$\therefore u(n) = \frac{(\log_2 n)^{i+1}}{i+1} = \frac{(\log_2 n)^1}{1} = \log_2 n$$

$\therefore T.C = u(n) = O(\log_2 n)$

Lec - 16

Recurr Relation by Master Theorem

$$T(n) = \begin{cases} T(\sqrt{n}) + \log n & \text{if } n \geq 2 \\ O(1) & \text{else} \end{cases}$$

$$T(n) = T(\sqrt{n}) + \log n$$

Put $n = 2^m$

$$T(2^m) = T\left((2^m)^{\frac{1}{2}}\right) + \log 2^m$$

$$T(2^m) = T(2^{m/2}) + m$$

Put $T(2^m) = S(m)$

$$S(m) = S\left(\frac{m}{2}\right) + m$$

Now $a = 1$ $b = 2$ $f(m) = m$

Soln :- $n^{\log_b a}\, U(n)$

$$\Rightarrow m^{\log_2 1}\, U(m)$$

$$= m^0\, U(m) = U(m)$$

$u(m)$ depend on $h(m)$

$$h(m) = \frac{f(m)}{m^{\log_b a}} = \frac{m}{1} = (m)^1$$

It is form of n^r $r>0$

so $u(n) = O(n^r)$

$\therefore$ T.C $= O(m)$

Now to find m

$\log n = \log 2^m$

$\log n = m$

$m = \log n$

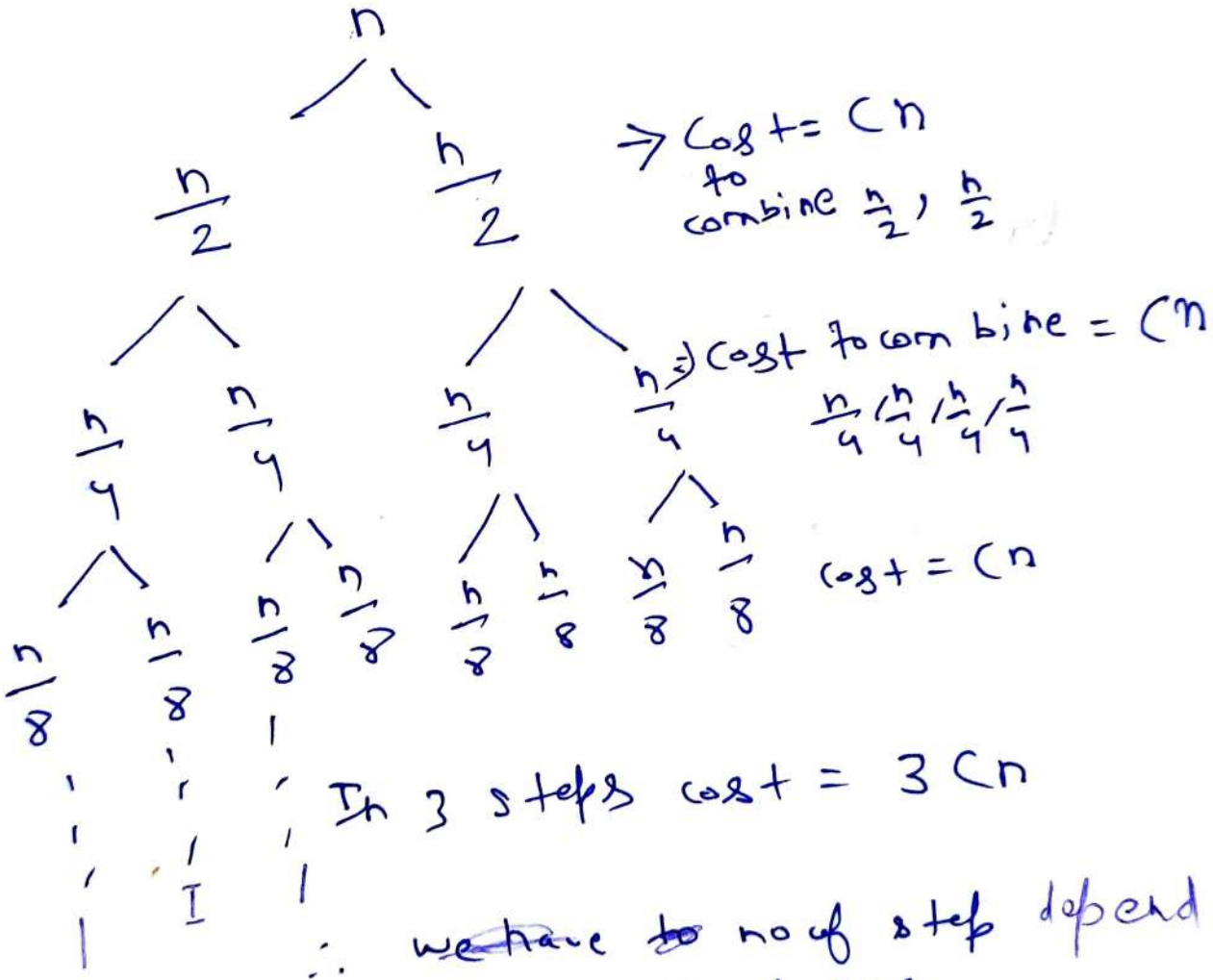
$\therefore$ TC $= O(\log n)$

Lec- 17

Recurrence Relation by Recursive tree method.

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

Here our tree will be like



In 3 steps cost = 3 cn

∴ we have to no of step depend on ∴ height of tree

Let suppose n = 16

∴ 16

8

4

2

height = 4

$\therefore \quad \log_2 16 = 4$

$\log_2 2^2 =$

$2^n = 2^2$

$n = 2$

So height will be $\log_2 n$

2^2

Cost $= c\, n \log n$

$n \log n$

c = constant

$\therefore$ T.C $= O(n \log n)$

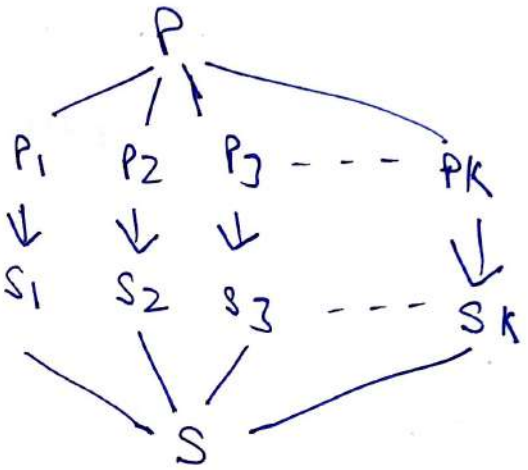
$n \cdot$

Lec - 18

Divide and Conquer (DAC) Approach.

```
DAC(P)

  if(Small(P))
  {
    S(P);
  }
  else
  {
  divide P into P1, P2, P3 -- PK
   Apply DAC(P1), DAC(P2) --- DAC(PK)
  Combine DAC(P1), DAC(P2) --- DAC(PK)
  }
}
```



→ Application of DAC

1) Binary Search
2) Find max and min
3) Quick, merge Sort
4) Strassen's Matrix Multiplication.

Lec 19, 20 Quick Sort Already Done

Lec-21

Ques on Merge Sort Algo

The average no of comparisons performed by merge sort algorithm in merging ~~to~~ 2 sorted lists of length 2 is = ?

⇒ 2 sorted list of length can be given in 3 cases

For example

case (1) [1 2] [3 4]

[1|2|3|4]

No of comp = 2

case (2) [1 3] [2 4] [1|2|3|4]

no of comp = 3

case (3) [1 4] [2 3]

[1|2|3|4]

no of comp = 3

$$\text{Avg comp} = \frac{2+3+3}{3} = 2.67$$

$\frac{1.09}{1.78}$

Lec-22 Bubble sort done

Lec-23 Insertion sort done

Lec-24 Selection Sort done

Lect 25 Binary Tree done

Notes of bubble,Insertion,Selection sort ,binary tree is at the end ,Dont get confuse with lec no of these topics.

Lec - 26

Introduction to heap tree

⇒ To check that given tree is heap tree or not It must follow 2 property.

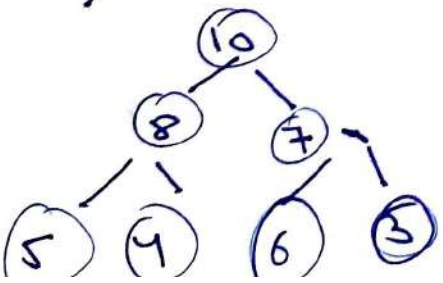
→ (1) Structural property

It should be complete binary tree

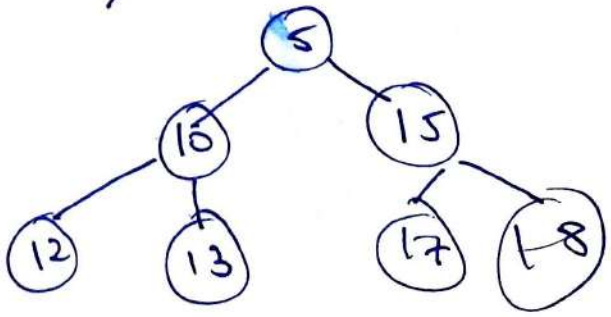
(2) Ordering Property

consider most of the time heap tree

Max heap
→ Root node should be maximum
→ Parent > child

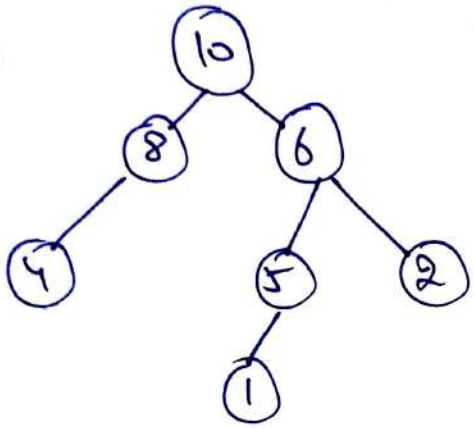


Min heap
→ Root node should be minimum
→ Parent < child

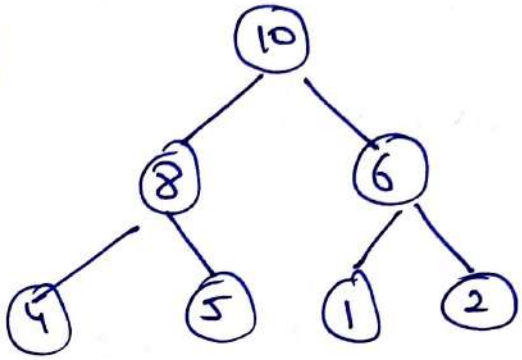


Que) which of the following is Max heap?

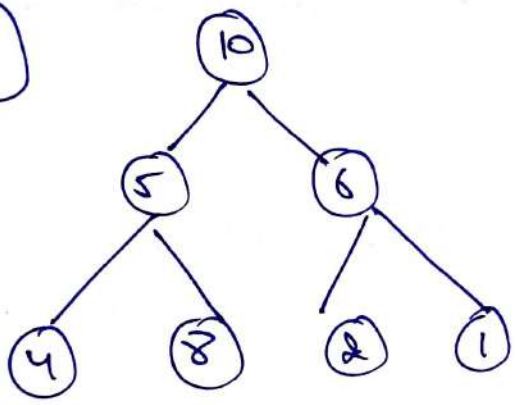
(A)



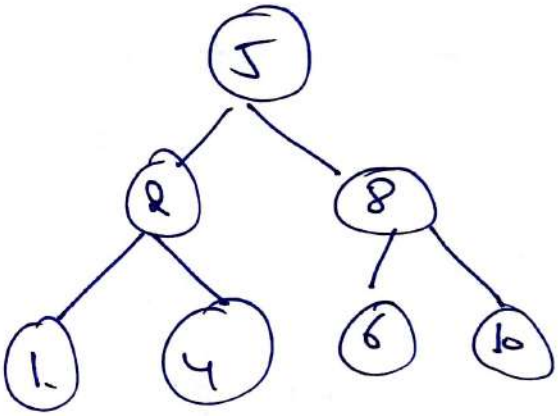
(B)



(C)



(d)



Lec-27

Insertion in heap Tree | Max heap, Min heap creation | Time Complexities.

=) Heap Tree construction

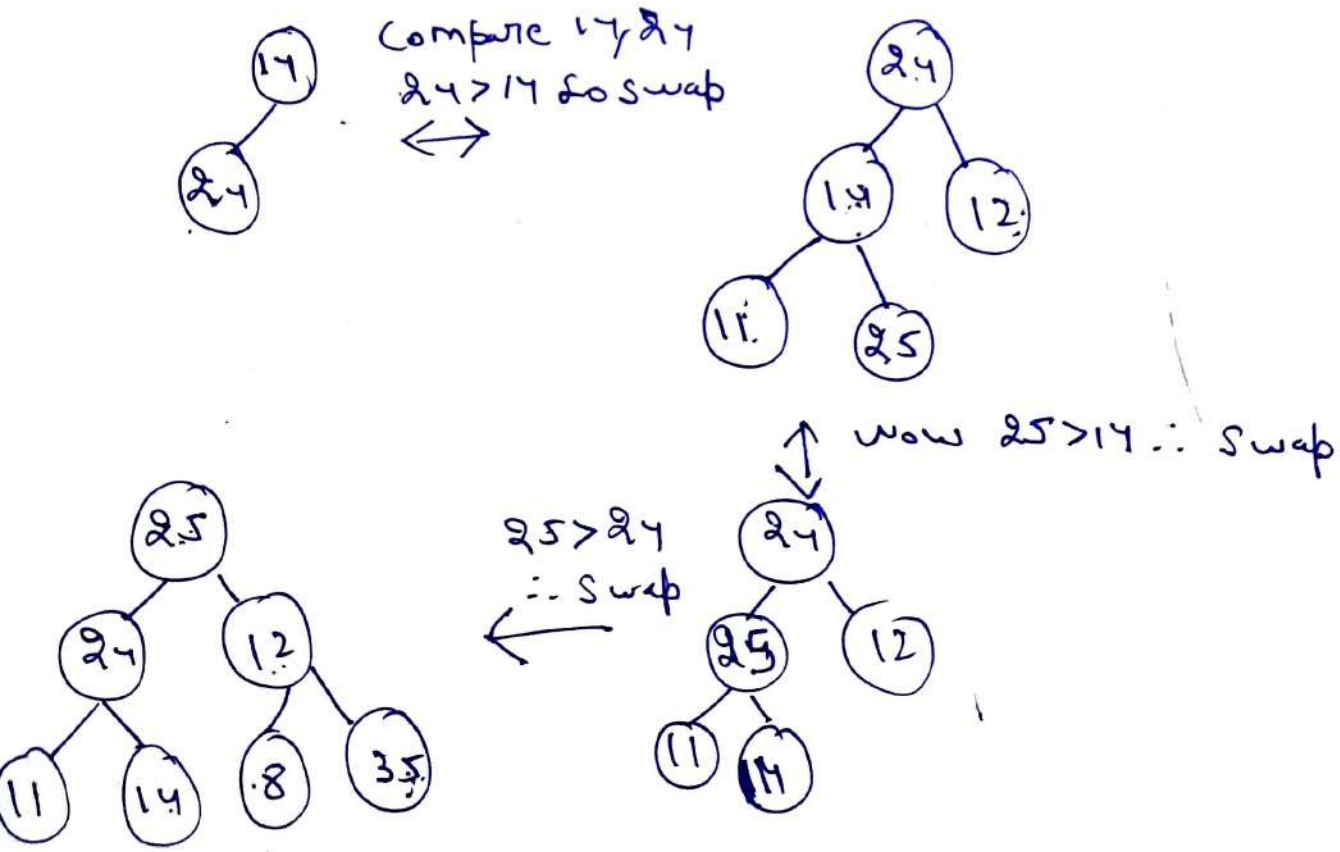
Method ① by

⊛ Insert key one by one in the given order

T.C = O(n log n)

from given keys, construct a max heap tree

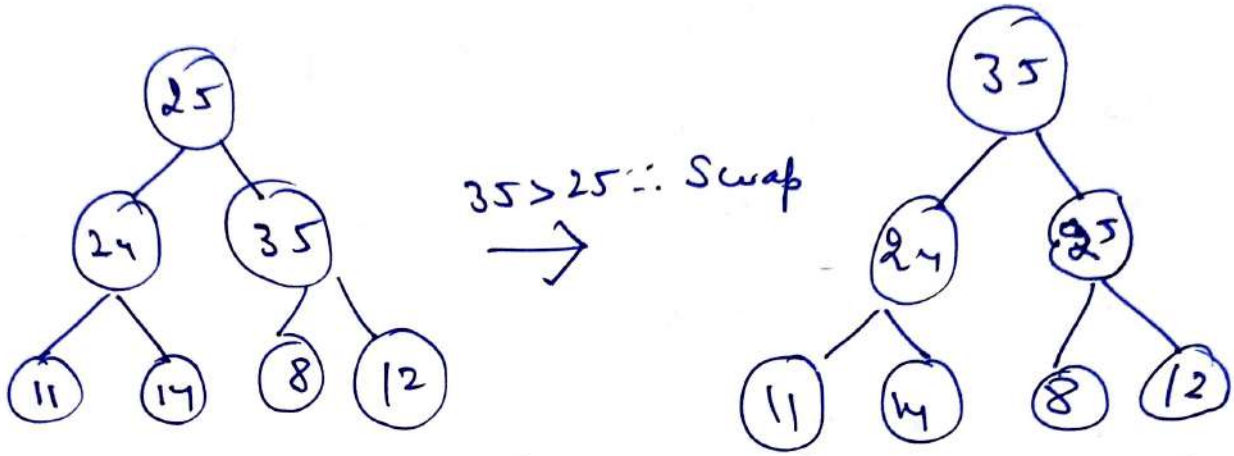
14, 24, 12, 11, 25, 8, 35



on Next Page

Compare 12, 35

35 > 12 ∴ Swap



Now finally we got Max heap tree

Algo Analysis

① Insert Key one by one. To insert a key into empty heap takes $O(1)$ time.

② To insert a key into already constructed heap in worst case

Time for comparison → $O(\log n)$

" " swapping → $O(\log n)$

Total time for 1 element $= 2\log n$

ie $O(\log n)$

for n elements

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

Lec-28

Ques on max heap / Min heap

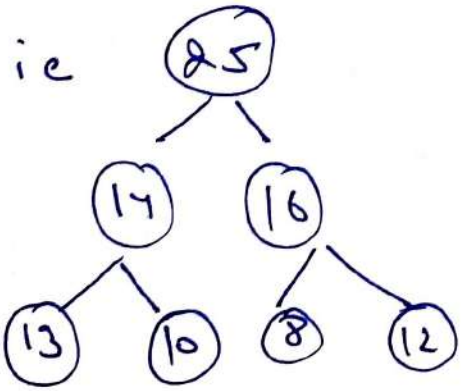
① consider a binary max-heap implements using an array.
Which one of the following array represents a binary max heap?

a) 25, 12, 16, 13, 10, 8, 14
b) 25, 14, 13, 16, 10, 8, 12
✓ c) 25, 14, 16, 13, 10, 8, 12
d) 25, 14, 12, 13, 10, 8, 16

② what is the content of array after 2 delete operations on correct answer of above question.

⇒ Above ans is 25, 14, 16, 13, 10, 8, 12

ie

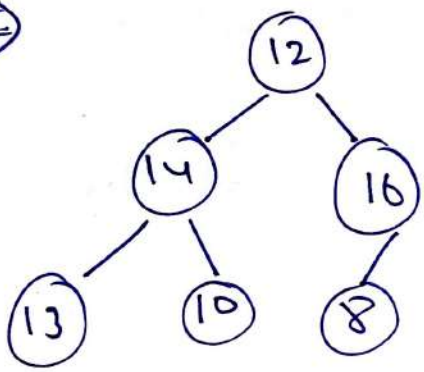


⇒ In deletion from heap tree first of all we have to delete root node.

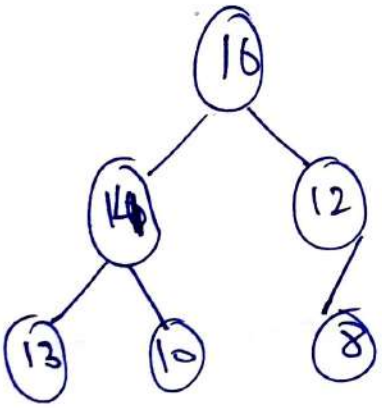
∴ delete (25) from tree

Now make right most leaf node as root node of tree

⇒

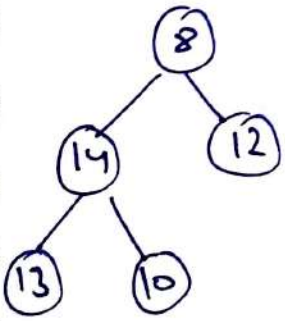


Now (16) > (12) ∴ Swap it



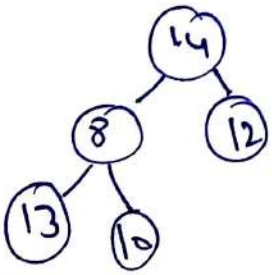
Now delete (16) and make (8) as root node

⇓

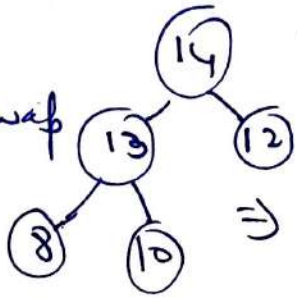


⇒

14>8 ∴ Swap



13>8 ∴ Swap



∴ final array after 2 deletion

⇒ 14, 13, 12, 8, 10

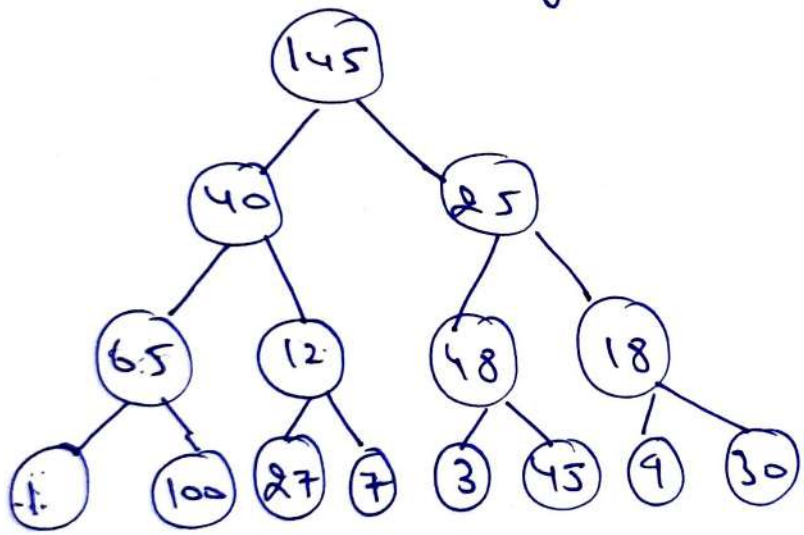
Lec- 29

Heapify method to build heap tree

Given keys are

145, 40, 25, 65, 12, 48, 18, 1, 100, 27, 7, 3, 45, 9, 30

→ first of all make a simple tree and insert key one by one without caring about all conditions.



Now we will min heap tree from above tree.

we will start comparison from 2nd last level from depth of tree

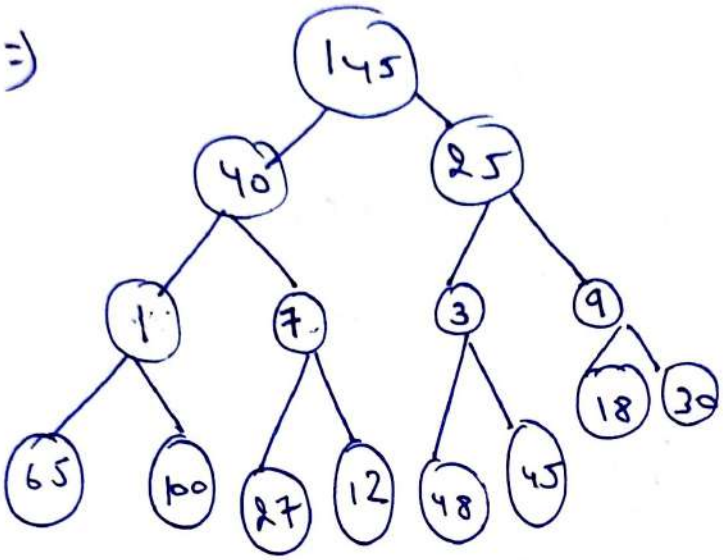
18 > 9 ∴ swap them.

48 > 3 ∴ swap them.

12 > 7 ∴ swap them.

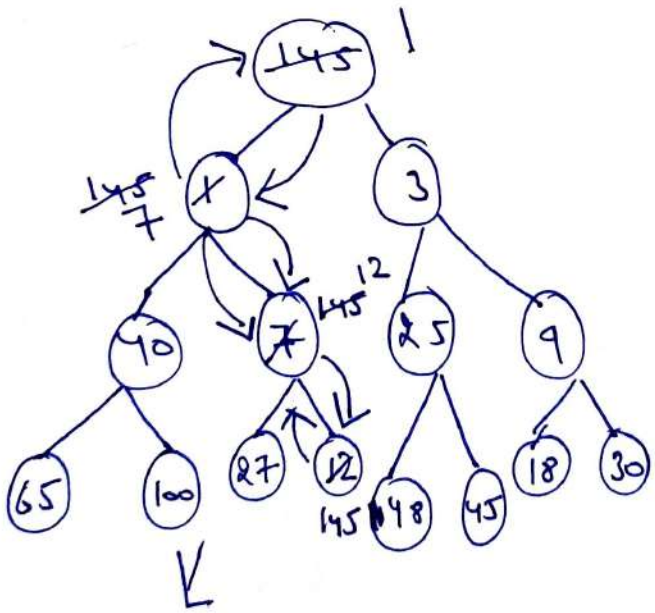
65 > 1 ∴ swap them.

⇒

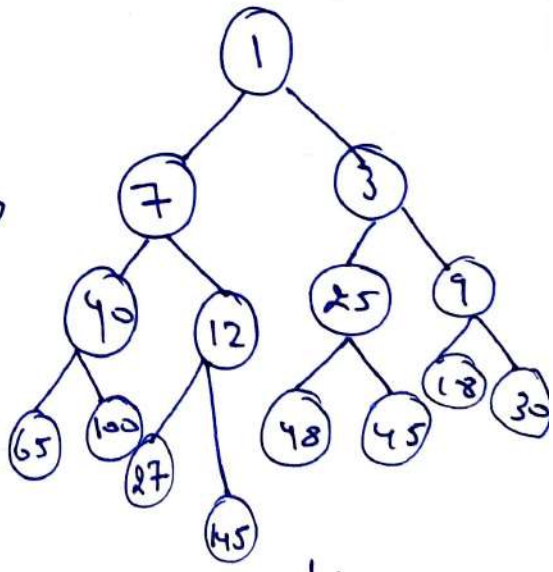


Now 25 > 3 ∴ swap them

40 > 1 ∴ swap them



⇒



145 > 1 ∴ swap them

after swapping 145 > 7 ∴ swap them

" " 145 > 12 ∴ swap them

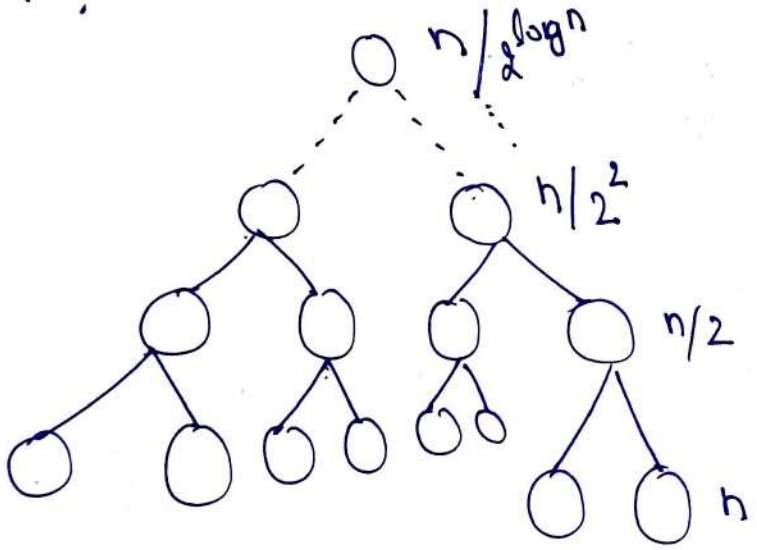
⇓

This is our final min heap tree

The no of comparison and swapping on a particular level depend on its height $\therefore$ TC = $O(\log n)$ of comparison and swapping.

Now The Time Complexity by heapify method for Heap tree = $O(n)$

Proof :-



Total Swaps = S

$$S = \frac{n}{2^0} \times 0 + \frac{n}{2} \times 1 + \frac{n}{2^2} \times 2 + \frac{n}{2^3} \times 3 + \cdots \frac{n}{2^{\log n}} \times \log n$$

$$S = n\left[\frac{1}{2} + \frac{2}{2^2} + \frac{3}{2^3} + \frac{4}{2^4} + \cdots \frac{\log n}{2^{\log n}}\right] - (1)$$

Multiply eq (1) by 2

$$\frac{S}{2} = n\left[\frac{1}{2^2} + \frac{2}{2^3} + \frac{3}{2^4} + \frac{4}{2^5} + \cdots \frac{\log n - 1}{2^{\log n}} + \frac{\log n}{2^{\log n + 1}}\right] \quad (2)$$

Subtract (2) from (1)

$$\frac{S}{2} = n\left[\left(\frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \frac{1}{2^4} + \cdots \frac{1}{2^{\log n}}\right) - \frac{\log n}{2^{\log n + 1}}\right]$$

$$= n\left[\left(\frac{\frac{1}{2}\left(1 - \frac{1}{2^{\log n}}\right)}{1 - \frac{1}{2}}\right) - \frac{\log n}{2^{\log n + 1}}\right]$$

$$= n\left[\left(1 - \frac{1}{2^{\log n}}\right) - \frac{\log n}{2^{\log n + 1}}\right]$$

$$= n\left[\frac{2^{\log n} - 1}{2^{\log n}} - \frac{\log n}{2^{\log n + 1}}\right] \qquad 2^{\log n} = n$$

$$= n\left[\frac{n-1}{n} - \frac{\log n}{n \cdot 2}\right]$$

$$\frac{S}{2} \Rightarrow n - 1 - \frac{\log n}{2} = \frac{2n - 1 - \log n}{2}$$

$$S = 2n - 1 - \log n$$

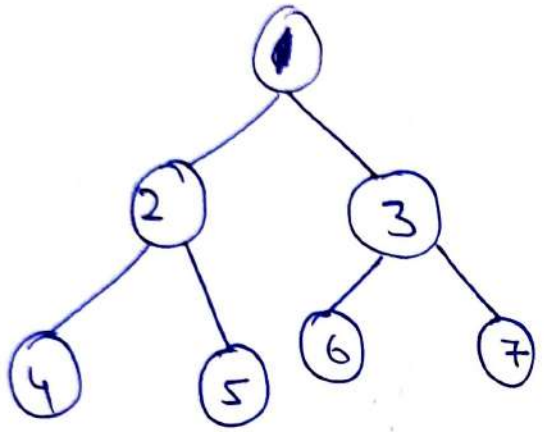
Now n is dominating on $\log n$

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

Lec-30

Deletion in Heap tree

using min heap

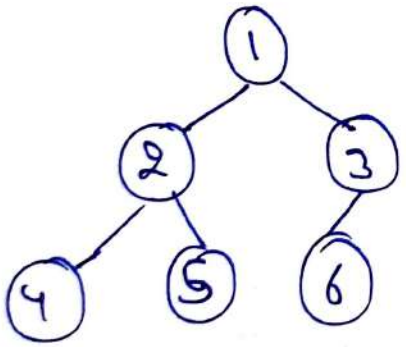


It is min heap tree

In deletion we can't remove directly any node.

We can only remove root node, and rightmost leaf node directly

Case 1 :. Remove Rightmost leaf node i.e (7)

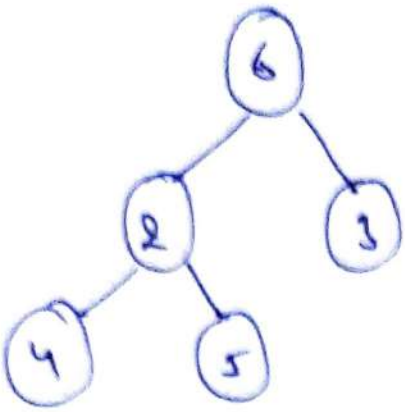


Best case

TC = O(1)

Case 2 worst case :. remove root node.

=) Now remove (1) and make (6) as new root node



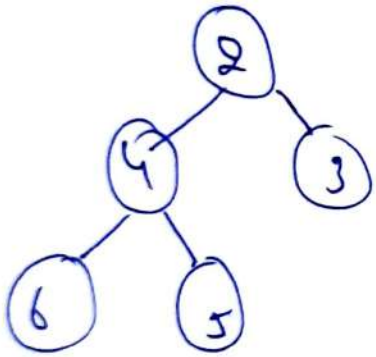
Now it is not min

So

$6 > 2$ ∴ swap them

after swapping $6 > 4$ ∴ swap them

∴ final min heap tree after deletion



Now for 2 level

no of comparison = 2

ie it depend on height

∴ for n no of level $= O(\log n)$

∴ $T.C = O(\log n)$

Lec - 31

Heap Sort (Inplace, Unstable, $O(n \log n)$)

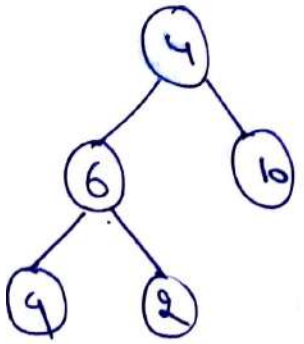
Given Keys =>

4 6 10 9 2 => ∴ Output should be

=> 2 4 6 9 10

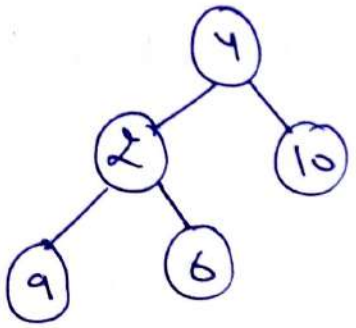
After sorting

First build heap tree by heapify method.

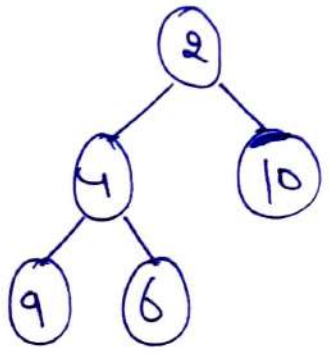


Now convert it into Min heap tree

6 > 2 ∴ swap it



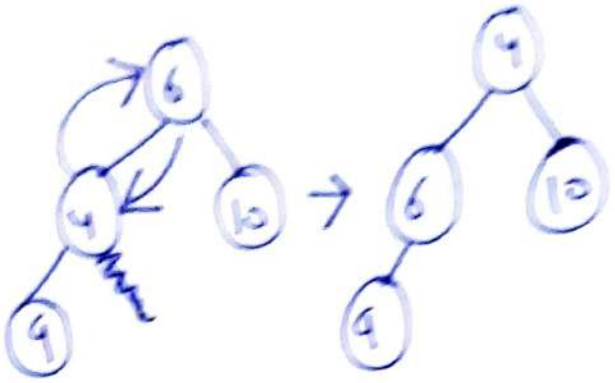
4 > 2 ∴ swap it



Min heap tree

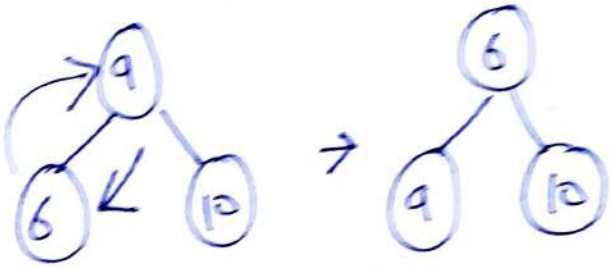
Now start deleting all element one by one

delete (2) and make (6) as root node

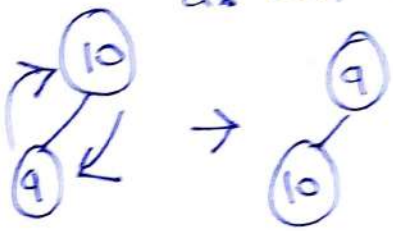


Deleted items
↓
2
4
6
9
10

Now delete (4) and make (9) as root node



Now delete (6) and make (10) as root node



Now delete (9) and then (10)

Deleted items 2 4 6 9 10

∴ we get sorted keys.

Time complexity = TC to make tree + To delete tree

$\Rightarrow O(n) + O(n \log n)$

nlogn is dominating

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

Lec- 32

Introduction to the Greedy Technique

⇒ Follows local optimal choice of each stage with intend of finding global optimum.

→ Feasible Solution (on basis of Selection)

→ optimal Solution (on basis of Min cost)

Applications :-

1) Knap sack problem
2) Job sequencing
3) Minimum Spanning Tree
4) optimal Merge Pattern
5) Huffman coding
6) Dijkstra's Algorithm

Greedy Algorithm approaches to

→ (1) Min. Cost
(2) Max. Profit
(3) Min Risk

Lec - 33

Knapsack Problem with Example

→ What is knapsack Problem?

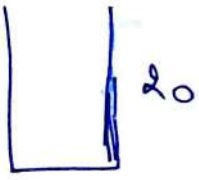
⇒ To describe Problem. we have taken a numerical.

In this numerical, we have 3 objects

objects	ob1	ob2	ob3
Profit	25	24	15
weight	18	15	10

Knapsack means bag

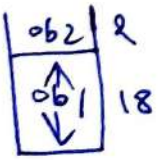
It has capacity of = 20 (M)



The problem says that when we put object, it should has maximum Profit.

⇒ Soln :- we have to implement in optimal way.

Case ① Greedy about Profit



Maximum Profit is in ob1

∴ Put it in box

Profit of ob1 = 25

Now 18 kg of weight over

2 kg left to Put 2 kg of ob2

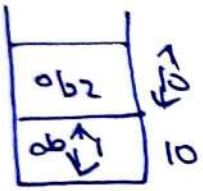
Profit of ob2 = $\frac{24}{15} \times 2 = 3.2$

Total Profit = 28.2

Case (2) Greedy about weight

⇒ It means low weight value should put first in box.

So Put obj first Profit of obj = 15



then Put 10 Kg of ob2

Profit of " " " " = $\frac{24}{15} \times 10$

= 16

Total Profit = 15 + 16

= 31

but case (1) (2) is not optimal

Optimal way is that we greedy about Profit/weight both.

Case (3) Greedy about Profit/weight both

We calculate Profit/weight ratio

	ob1	ob2	ob3
P/w	1.3	1.6	1.5

P/w of Ob2 is 1.6 which is high

∴ Put ob2 in bag

obj3	5
obj2	15

Profit of obj2 = 24

then P/w of obj3 is high

∴ Put 5 kg of obj3

Profit of " " " " = $\frac{15}{10} \times 5 = 7.5$

Total Profit = 24 + 7.5
= 31.5

which is higher than case 1, 2

and optimal too.

→ Calculation of T.C

```
for i = 1 to n
    calculate P/w
```
→ $O(n)$

[Sort object in ↓ order of P/w Ratio → $O(n\log n)$

```
for i = 1 to n
    if M > 0 and wi ≤ (m)
        M = M - wi;
        P = P + Pi;
    else break;
    if (M > 0)
    P = P + Pi (M/wi);
```
→ $O(n)$

Total = $O(n) + O(n\log n) + O(n)$

nlogn is dominating ∴ T.C = $O(n\log n)$

Lec-34

Huffman Coding (Greedy Technique)

⇒ It is used to encode the data and to compress the data

Let's do it with the help of an example

We have a Message M which consists of total 100 characters.

a=50
b=16
c=10
d=5 f=2
e=3

Here there are total 6 types of characters ie a, b, c, d, e, f whose frequency is also given

M = 100 characters

Now we have to encode this message in digitized form

→ one of the encoding techique ie ASCII technique which has range of (0 to 127) so maximum no here is 127 to represent it binary → 1111111 we need 7 bits.

∴ from 0 to 127 to represent in binary form we need 7 bits

We have total 100 characters

∴ To Represent 100 characters = 100 × 7
we need = 700 bits

→ If we talk about another technique
ie As we know that there are only 6 types of characters in message
So To identify them uniquely we can represent them as a = 000
b = 001
c = 010
d = 011
e = 100
f = 101

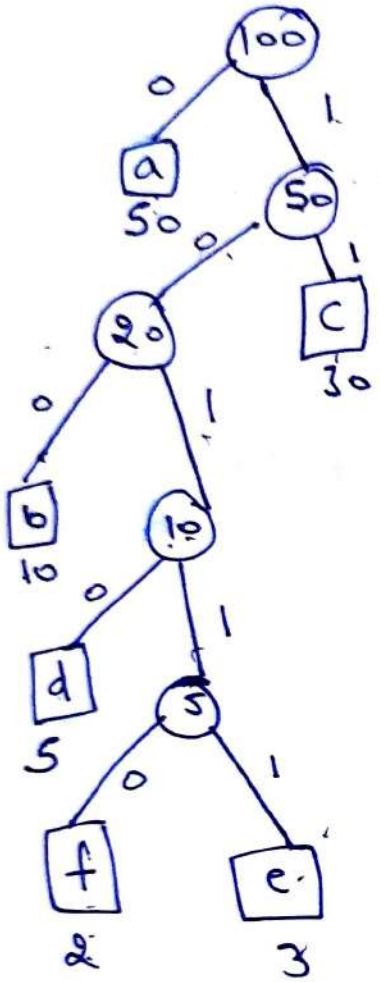
∴ To represent 1 character we need = 3 bit
" " 100 " " " = 3 × 100
= 300 bits

→ Now huffman coding says that 300 bits though very much amount
∴ Acc to huffman coding
first we take out last 2 lowest freq characters from Message ie e, f

Now combine [e], [f]

it make 2+3 = 5

and Put (5) in Message



M [a=50, b=10, c=30, d=5, (5)]

Now take out (5) and d from M as last 2 low freq no are d, (5)

then combine (5) and d

∴ (5) + 5 = 10

Now Put (10) in M

Now

M = [a=50, b=10, c=30, (10)]

Now take out (10) and [b] and then combine them (10) + 10 = 20

Now Put (20) in M

Now M = [a=50, c=30, (20)], Now take out (20) and c, then combine

20 + 30 = 50

Now (50) in M

M = [a=50, (50)]

Now take out [a], (50) from M and then combine

50 + (50) = 100

Now finally we got Huffman tree

Now make all left sided edges as 0

" " " right " " as 1

Now

Name of characters	Path to Reach Particular chara.	No of bit for 1 charac.	No of bit for freq charac.
a	0	1	50×1 = 50
b	100	3	10×3 = 30
c	11	2	30×2 = 60
d	1010	4	5×4 = 20
e	10111	5	3×5 = 15
f	10110	5	2×5 = 10
		Total bits	185

Hence To encode M in digital form we need = 185 bits

Averge bit required to represent each character = $\frac{185}{100}$ = 1.85 bits/chara.

Advantage is
It is efficient in space
Transmission time is low

Lec- 35

Ques on huffman coding

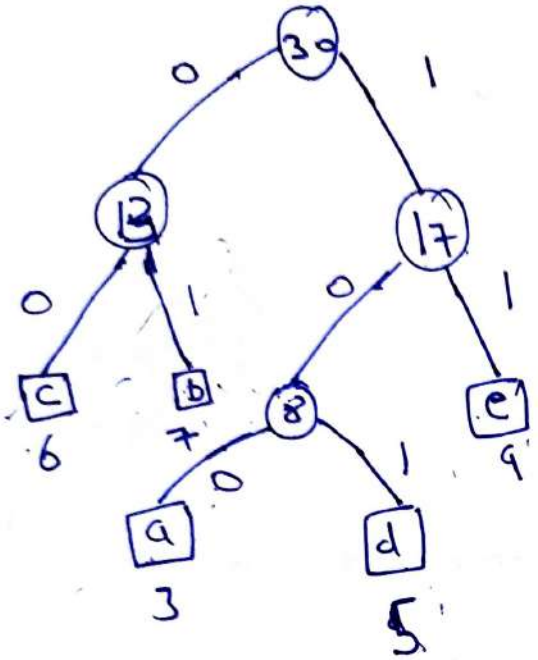
→ Consider the following message

aa bb bba bbb ccc ddd eee ccc eee dd eee

Find the no of bits required for huffman encoding of above msg?
also find the avg no of bits required to represent a character?

	freq
a =	3
b =	7
c =	6
d =	5
e =	9

soln ⇒ Now acc to huffman technique we will make 11 tree



edges = no of edges to particular chara. from root

Character	No of bit Total = (edges x freq)
a	$3 \times 3 = 9$
b	$2 \times 7 = 14$
c	$2 \times 6 = 12$
d	$3 \times 5 = 15$
e	$2 \times 9 = 18$

Total = 68 bits

$$\text{Avg bits} = \frac{68}{30} = 2.26$$

30 = no of total chara.

Lec- 36

Job Sequencing (Greedy Technique)

⇒ let do it with help of an example

	J_1	J_2	J_3	J_4
Profit	50	15	10	25
Deadline	2	1	2	1

we have 4 Jobs, with 2 Parameters ie Profit and Deadline in (months)

It is uniprocessor ie we have only one Software Engineer to handle all 4 jobs

It is non Premptive ie we ~~can~~ have to finish one job before starting other one.

Each job takes one unit of time
suppose = 1 month

Now we will make Gant chart

J_1	J_3

0 1 2

Now J_1 has max. Profit so Put it in 0 – 1

Now 1 month has completed Now J_2, J_4 can't be attended now

(1-2) ~~has~~ as J_2, J_4 has deadline of 1 month and 1 month over.

So J_3 can be attempted as it has deadline of 2 months and 1 month is left.

So

J_1	J_3

0 1 2

Profit from $J_1 = 50$

" " $J_3 = 10$

Total Profit = 60

But it is not maximum profit
It is not optimal way.

∴ Acc to Job Sequencing,
first we choose max. Profit Job
ie J_1 Now J_1 has deadline of 2 months and has max deadline
∴ It can be done lastly month.

So

	J_1

0 1 2

Now next max Profit job is J_4

$\therefore$

J_4	J_1
0	1 2

Profit from $J_4 = 25$

" " $J_1 = 50$

Total $= 75$

ie Maximum Profit.

Calculation of T.C

→ Arrange all jobs in ↓ order of Profit

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

→ For each Job(m i), do linear search to find particular slot in array of size (n),

where $n =$ max deadline

$m =$ total jobs

$n \times m = n^2$ $(m = n)$

$O(n^2)$

n^2 is dominant to $n\log n$

$\therefore$ TC $= O(n^2)$

Lec- 37

Optimal Merge Pattern using greedy method

⇒ lets us understand the problem by help of an example

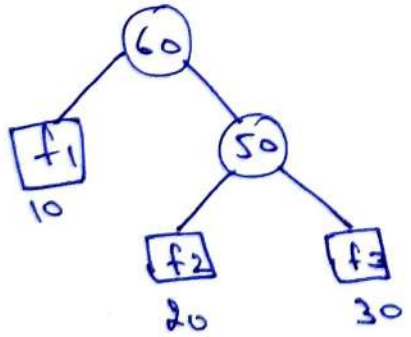
we have three files

files	f_1	f_2	f_3
no of records	10	20	30

We have to merge 3 files in a single file which contain single record

We can use only 2 files at a time and we have to find no of record movement?

For example



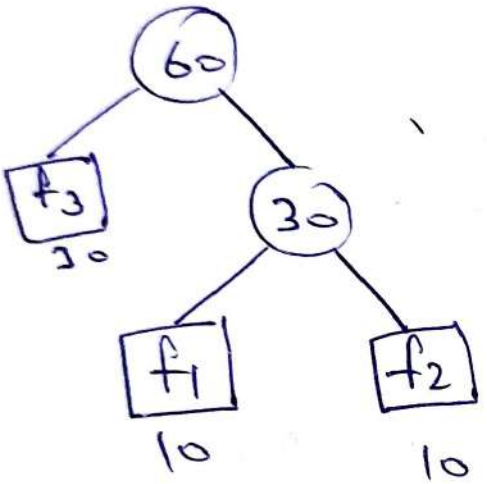
no of record movement $= 60 + 50$
$= 110$

but 110 is no optimal value

It is feasible solution
It is one of the soln.

Now we will find optimal solution by optimal merge pattern.

=) Acc. to optimal merge pattern, first take 2 minimum no of records files and then merge them.



then the left file merge with previously merged file

no of record movement = 60 + 30 = 90

which is minimum and optimal.

Time complexity finding

Algorithm

1) create Min heap with n files
TC = O(n)

2) At each level remove two minimum 'a' and 'b' from min heap and put (a+b) again into heap.

T.C to remove 'a' = $O(\log n)$

TC to " 'b' = $O(\log n)$

TC to Put (a+b) again = $O(\log n)$

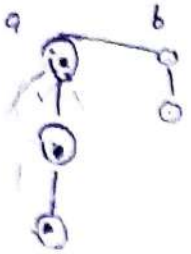
no of level = $(n-1)$

So TC = $O(3 \log n (n-1))$

$= O(n \log n)$

Total TC = $O(n \log n + n)$

$\Rightarrow O(n \log n)$



Lec- 38

Spanning Tree

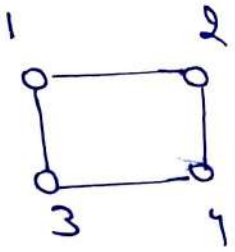
Spanning Tree :- A connected Subgraph 'S' of Graph $G(V, E)$ is said to be spanning iff, $V \rightarrow$ vertex, $E \rightarrow$ Edges

(1) 'S' should contain all vertices of 'G'

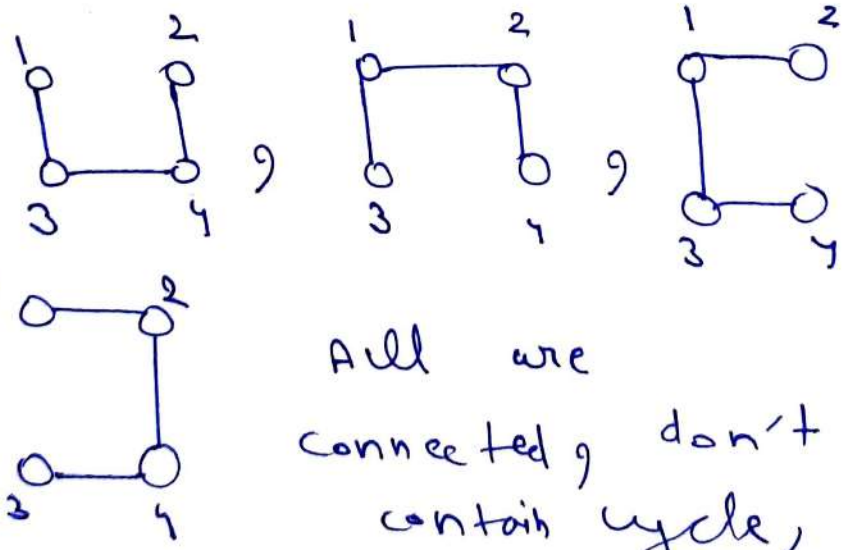
(2) 'S' should contain $(|V| - 1)$ edges

For example

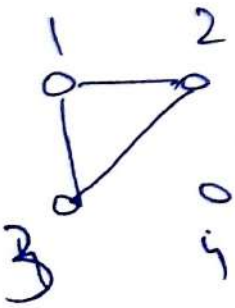
Graph G →



Corresponding Spanning Tree →



All are connected, don't contain cycle, contain $4 - 1 = 3$ edges



Not a spanning Tree

If we have give a complete graph

vertices n

then no of spanning tree = n^{n-2}

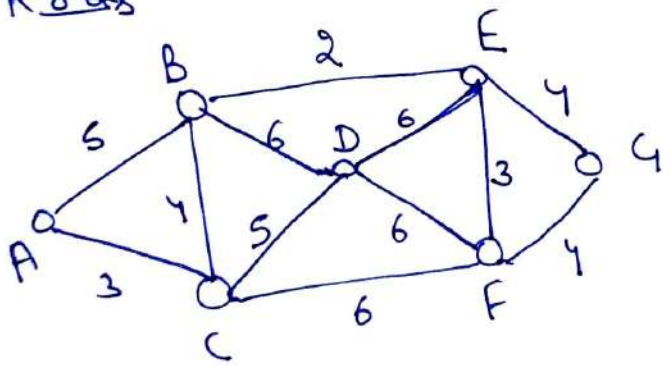
Lec- 39

Kruskal Algorithm ~~in~~ for

Minimum Spanning tree

=) We have given a graph.

~~Krus~~



Each edge has its cost

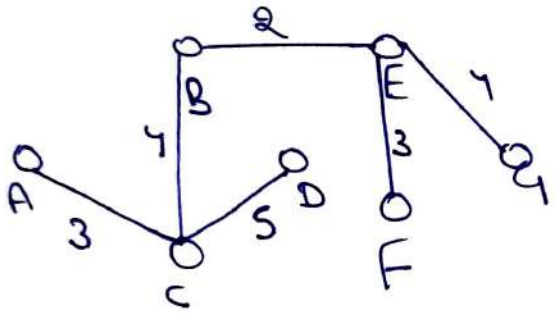
Kruskal Algorithm is used to make minimum cost spanning tree.

=) From above given graph, we will make minimum cost spanning tree

Kruskal algorithm says that take edges in ascending order (accordance to cost) and connect them in such ways should be made.

Avoid those edges which led to make tree cyclic.

So BE has minimum cost. Start with it



Cost = 3 + 4 + 5 + 2 + 3 + 4
= 21

It is minimum cost spanning tree for the given graph.

Calculation of TC

1) Construct min heap with 'e' edges = $O(e)$

2) Take one by one edge and add in spanning tree (cycle should not be created).

Best case has = $(n-1)$ edges

To make 1 edge as root node of min heap
T.C = $\log e$

for $(n-1)$ edges T.C = $O((n-1)\log e)$

worst case has e edges

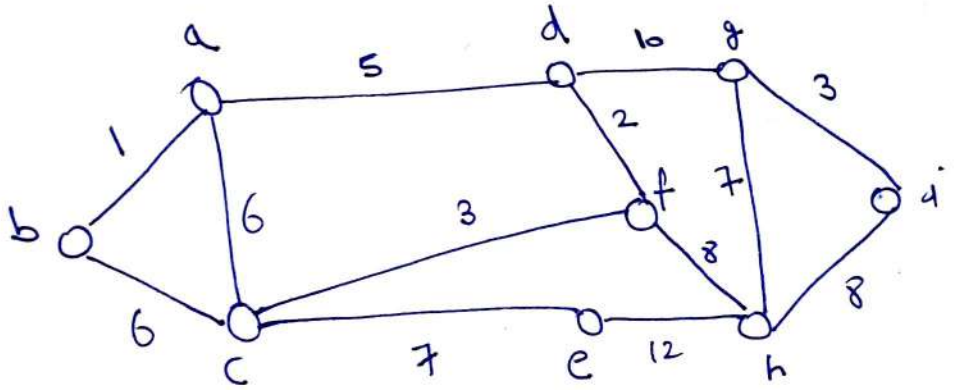
$\therefore$ T.C = $O(e \log e)$

Lec- 40

Prims Algorithm for Minimum cost spanning tree

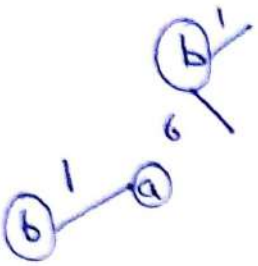
It is also uses to make minimum cost spanning tree.

We have given graph



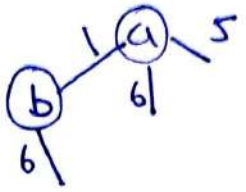
Prims algorithm says we can start making tree from any vertex.
Then, we will see edges which go out from that vertex and we will make edge, which has less cost then from next vertex of first edge we will follow same procedure and we will compare with all previous edges.
and avoid those edges which make tree cyclic.

Suppose we will start with (b)



1 < 6 so consider 1

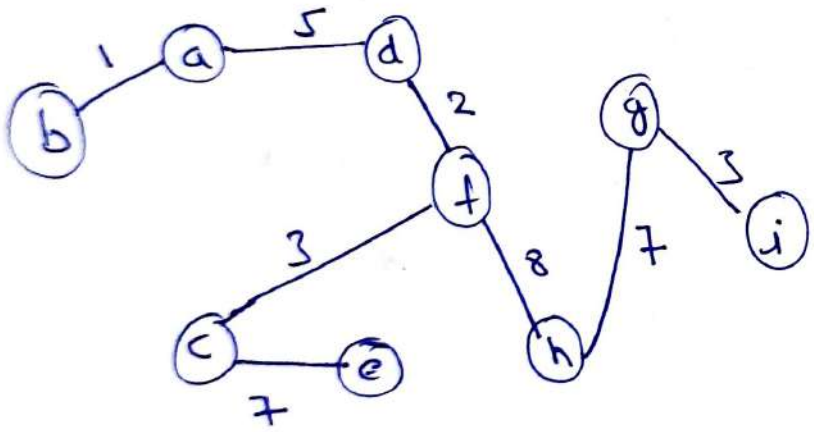
Now at (a)



5 is lesser than 6, 6

So

Now we will follow same logic and finally our tree will be



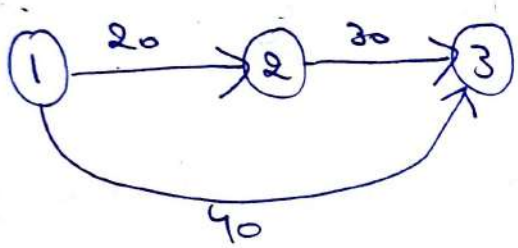
Lec-41

Dijkstra's Algorithm (Single Source Shortest Path)

⇒ It is used to find shorest path from single source.

Working of Algo

we have given a graph (directed)



This algo works on Relaxation concept of vertex.

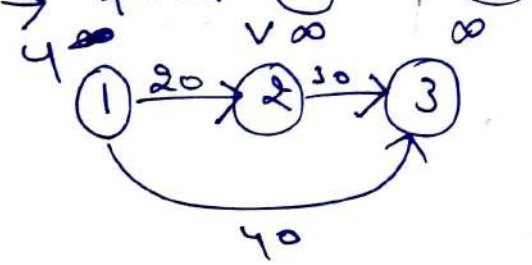
Relaxation → if $d(u) + c(u,v) < d(v)$

then $d(v) = d(u) + c(u,v)$

u, v are two vertex

Initially suppose cost (in term of distance) from ① to ② and ① to ③ is ∞

→ from ① to ② ① → u , ② → v



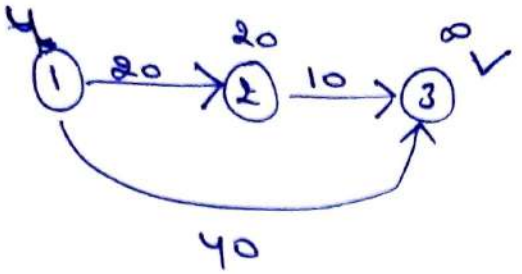
$d(u) + c(u,v) < d(v)$

$0 + 20 < \infty$

$20 < \infty$

so $d(v) = 0 + 20 = 20$

$\therefore$



① to ②

distance = 20

→ from ① to ③ direct, ① → 4, ③ → V

$d(4) + c(4, V) < d(V)$

$0 + 40 < \infty$

$40 < \infty$

So $d(V) = 0 + 40 = 40$ — (a)

→ Now from ① to ③ via ② (indirect)

$\therefore$ ② → 4 ③ → V

$d(4) + c(4, V) < d(V)$

$20 + 10 < 40$

$30 < 40$

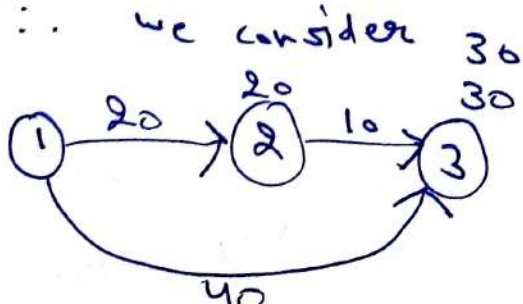
$\therefore d(V) = 20 + 10 = 30$ — (b)

Now from (a) (b)

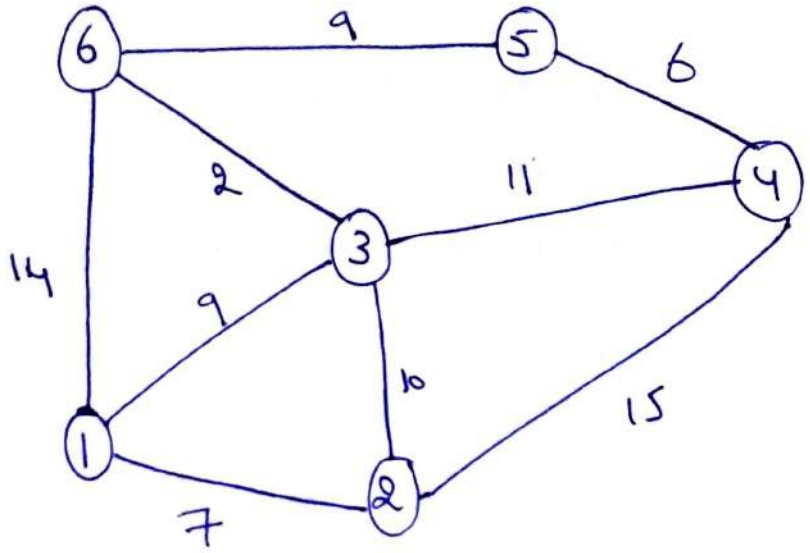
30 distance < 40 distance

$\therefore$ we consider

So final



Now lets do an example of undirected Graph.



Source	Destination				
	2	3	4	5	6
1	∞	∞	∞	∞	∞
	7	9	∞	∞	14
1, 2	7	9	22	∞	14
1, 2, 3	7	9	20	∞	11
1, 2, 3, 6	7	9	20	20	11
1, 2, 3, 6, 4	7	9	20	20	11
1, 2, 3, 6, 4, 5					

All relaxed.

Initially from source 1 to all destination 2, 3, 4, 5, 6, cost is ∞

→ Starting from 1, to 2 cost → 7

to 3 cost → 9

to 4 No direct connection

∴ cost $= \infty$

to 5 No direct connection

∴ cost $= \infty$

to 6 cost → 14

out of these cost minimum is 7

∴ 7 is fixed and vertex 2 is related, Now 2 also comes under source

Now we can go to 3, 4, 5, 6 via 2 also if possible.

→ Now from 1 to 2 cost = 7

1 to 3 cost is 9 and via 2 is 17

∴ $9 < 17$

∴ cost = 9

from 1 to 4 via 2 cost = 7 + 15 = 22

from 1 to 5 cost = ∞

from 1 to 6 cost = 14

Now from 7, 9, 22, ∞, 14

minimum is 9 ∴ 9 is fixed and vertex 3 relaxed and undergo source

→ Now 1, 2, 3 are source

we can go to 4, 5, 6 via 2, 3

from 1 to 2 cost = 7

1 to 3 cost = 9

1 to 4 via 3 cost = 9 + 11 = 20

via 2 cost = 7 + 15 = 22

Now 20 < 22

∴ cost = 20

1 to 5 cost = ∞

1 to 6 via 3 cost = 9 + 2 = 11

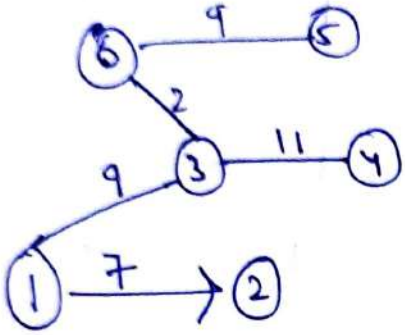
11 < 14

∴ cost = 11

Now out of them minimum is 11

∴ In this way we follow some procedure until all vertex get relaxed and we get minimum cost

Minimum cost spanning tree



Lec- 42

Dijkstra Algorithm Analysis

⇒ Dijkstra (Graph, Source)

create vertex set Q

for each vertex v in graph

dist [v] = ∞

dist [Source] = 0

add v to Q → Build min heap] – TC – $O(V)$

TC = $O(V)$

while Q is not empty

u = Extract-min [Q] → T.C = $O(V \log V)$

for each neighbour v of u

Relax [u, v] → T.C = $O(E \log V)$

Total T.C = $O(V) + O(V) + O(V \log V) + O(E \log V)$

Dominant = $O(E \log V)$

as $E > V$

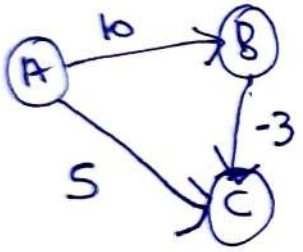
∴ T.C = $O(E \log V)$

Lec-43

Why does Dijkstra Algo fail on Negative weights ?? or edges

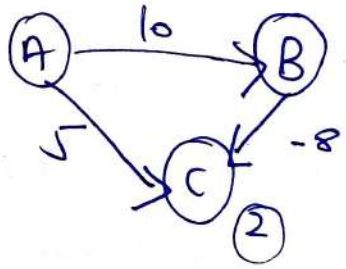
-ve weight Edge

Algo works



Source	Destination	
	B	C
A	∞	∞
A	10	5
AC	10	5
ACB		

Algo not works



Source	Destination	
	B	C
A	∞	∞
AC	10	5
ACB	10	5

Now After Relaxing B
we can go to C via B
also and cost = 10 − 8 = 2
2 < 5

but Acc to algo once we relaxed a vertex once then we can't change its cost
∴ Algo fails in -ve edge

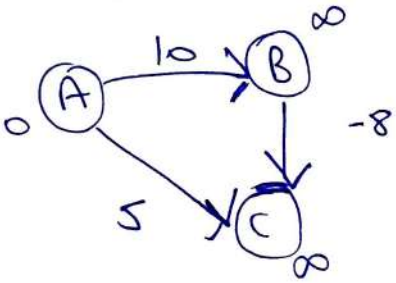
Lec -44

Bellman Ford Algorithm

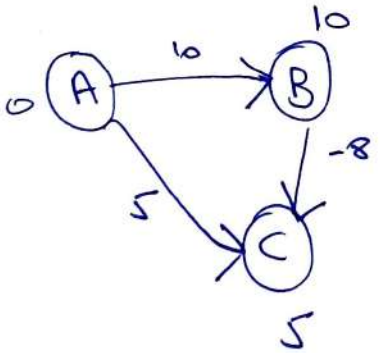
→ This is solution to -ve edges.
It gives correct ans in -ve edges

In Bellman Ford we will relaxe each edges (V-1) no of Times V → no of vertices.

Graph Given



↓ Relax 1



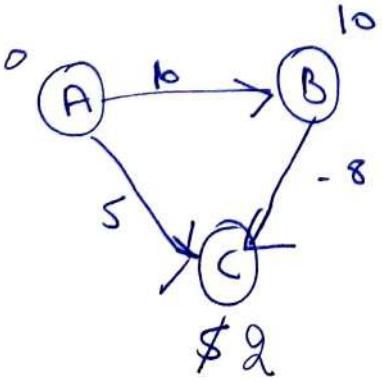
AB → 0+10 < ∞ ∴ AB = 0+10 = 10

BC → ∞ − 8 ≮ ∞ ∴ no effect on distance

AC → 0+5 < ∞

∴ AC = 0+5 = 5

↓ Relax 2



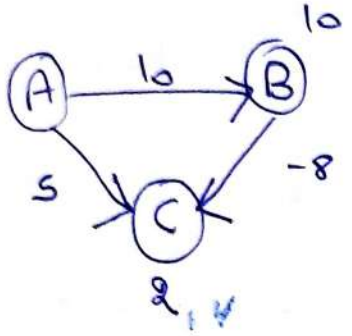
AB → 0+10 ≮ 10 ∴ No effect

BC → 10−8 < 5 ∴ BC = 10−8 = 2
2 < 5

AC → 0+5 ≮ 2 ∴ No effect

To check -ve edge cycle

~~check~~ Relax it once more.



AB → 0 + 10 ≮ 10 No effect

BC → 10 - 8 ≮ 2 No "

2 ≮ 2

AC → 0 + 5 ≮ 2 No Effect

5 ≮ 2

So No Effect ∴ No -ve edge cycle

If we found any changes on cost then -ve edge cycle is present and we will stop there.

Lec - 45

Time complexity of Bellman Ford

Psudo code

```
Bellman Ford (G, V, E, S)
  Step 1
  for each vertex v ∈ G do
      dist[v] = ∞
      dist[source] = 0
                                  } O(V)

  Step 2
  for i=1 to |V| -1
    for each edge (u,v) ∈ G
      Relax (u, v, w) -   TC - O(1)
                                  } O(V·E)

  Step 3
for each edge (u,v) ∈ G
  if( dist[u] + w(u,v) < dist(v))
    return "Graph contain -ve weight cycle"

  return distance
```

Total T.C = $O(V.E)$

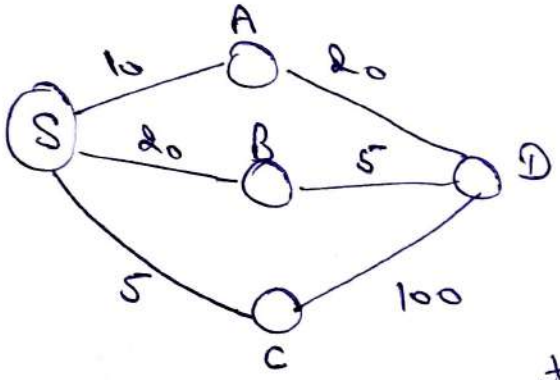
Lec-46

Introduction to dynamic Programming (DP)

→ Dynamic Programming gives optimal solution always.

→ But 'greedy' also gives optimal solution technique

So diff b/w them is



Greedy technique follow path S → D because initial cost = 5 is minimum but total cost = 105 is high

Dynamic Programming traverse through all paths from (S) to (D) and then examine which has minimum cost and then follow that path.

Dynamic Programming divide the problems into series of overlapping sub-problems.

Two features ① optimal Sub Structure
② overlapping Sub Problems

overlapping subproblems means repeative problems here repeative problems solve only once and store somwhere in memory and use when in need.

It reduced T.C and it is main motive of DP.

Lec-47

Knapsack failed using greedy approach

0/1 Knapsack

Knapsack means carry bag

0 → means obj not present

1 → means obj present

object	ob1	ob2	ob3
weight	2	4	5
Profit	20	25	60

Acc to knapsack, we we not put objects partially in bag, we will put complete object in bag

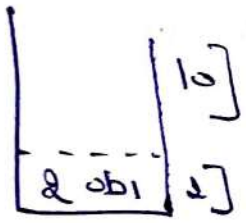
Suppose knapsack capacity (M) = 12

If we use greedy approach in knapsack

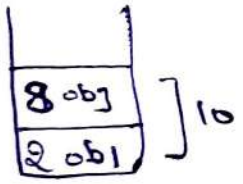
P/w = 10 6.2 7.5

Highest P/w = 10

∴ Put obj in bag first



then 7.5 so put obj in bag



Acc to capacity only 2 Kg of weight of left but obj2 has 4 so in knapsack we can't put partially

∴ Profit = obj1 + obj3
20 60 = 80

But it is not max Profit
Max Profit = 85

∴ Greedy approach fails in Knapsack

Lec-48

→ 0/1 Knapsack (KS) Recursive eq.

0/1 Knapsack (n, m) → n → total no of object
m → total weight of capacity of ~~object~~ of knapsack

Recursive eq is.

n — n is nth element

p_n — p_n is profit

w_n — w_n is weigh of nth element

eq when we put object in knapsack → 0/1KS(n-1, (m-w_n)) + p_n

(n-1) → means now 1 object reduced

(m-w_n) → w_n weight reduced from total capacity

p_n → p_n means profit obtained on that nth object

eq when we don't put object in KS

0/1KS(n-1, m)

when $w_n > m$

eq → $0/1\,KS(n-1, m)$

when $n = 0$ OR $m = 0$

output = 0

These all equation is recurrence relation

→ Now let's discuss about Recursion tree and its T.C

As we know "0/1 KS Recursive eq is

$$0/1\,KS(n,m) = \begin{cases} 0/1\,KS(n-1, m-w_n) + p_n \\ 0/1\,KS(n-1, m) \end{cases}$$

Let take an example

$0/1\,KS(4, 4)$

$n \to 4$

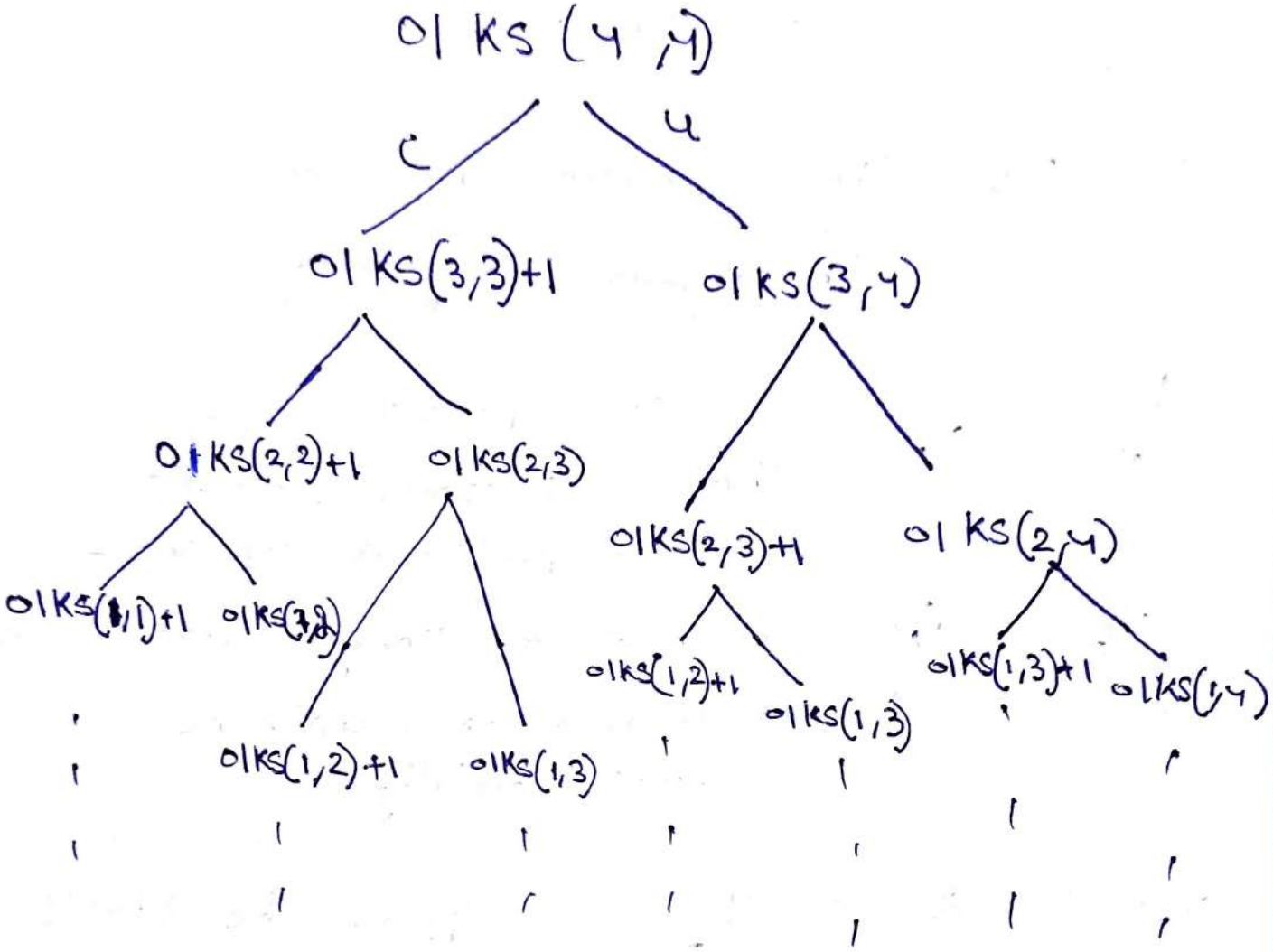
$m = 4$

total 4 object

Each object has its weight = 1 and profit = 1 on each object

Consider C → when we put object in KS

Unconsider U → when we don't put in KS



Now there are total 2^n no of fn call

$\therefore$ Acc to it T.C should be $O(2^n)$

But Acc to dynamic Programming when some repeative fn calls are there then, solve them once and store the result in a table.

So Unique fn call = $n \times m = n.m$

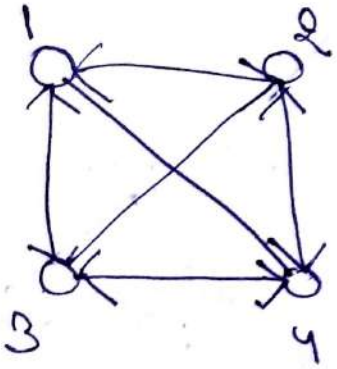
$\therefore$ T.C reduced from $O(2^n)$ to $O(n.m)$

with the help of dynamic Programming.

Repeative no of fn call = $(n+1)(m+1)$

Lec-49

Travelling Salesman Problem in Dynamic Programming



This problem says that a salesmen start from a particular and visit all nodes and finally 'came' back to the node from which he started initially

cost	1	2	3	4
1	0	10	15	20
2	5	0	25	10
3	15	30	0	5
4	15	10	20	0

→ Suppose he starts from 1

→ Now Acc to greedy technique his path will be like

$1 \xrightarrow{10} 2 \xrightarrow{10} 4 \xrightarrow{20} 3 \xrightarrow{15} 1$

Total cost = 55

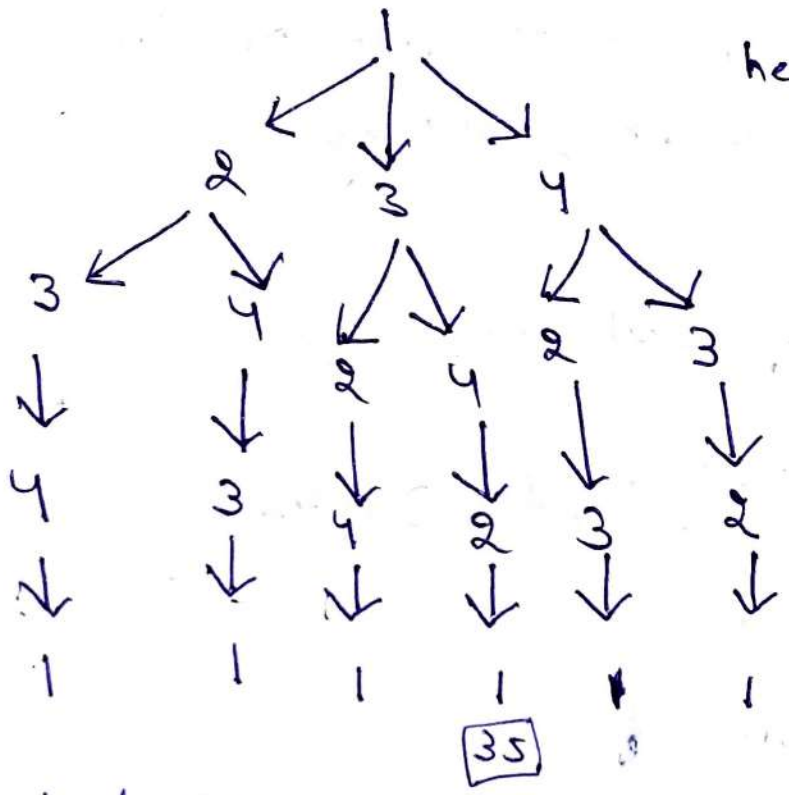
Acc to greedy technique minimum cost = 55

→ Now we will find cost Acc to Brute force method

This method says that try all possible ways to traverse the path

Path will be like

It is also NP complete problem / hemitolian cycle problem



Out of them
minimum cost = 35 Path is
$1 \to 3 \to 4 \to 2 \to 1$

∴ greedy technique fails here.

→ Now calculate T.C
from any Path like $1 \to \boxed{2 \to 3 \to 4} \to 1$

only $(n-1)$ th term changes
and create diff possible outcome

$\therefore$ $T.C = O(n-1\,!) = O(n!) = O(n^n)$

n^n is biggest TC

and dynamic Programmin is not so much successfull to reduced T.C as repeative path are not present so much.

Lec- 50

Sum of Subsets Problem in Dynamic Programming

=) Sum of Subset (SOS)

Set $S = (10, 20, 30, 40)$

$m = 50$

$m =$ Sum.

Here sum of subset problem says that we have to check that whether there are ~~f~~ subsets present or not whose sum $= m$.
We will answer in Yes/No.

Here we will brute force full method
It say pick out all possible subset and find out sum and check where is sum $= m$.

Total possible subsets $= 2^n$

Recursive equation of SOS

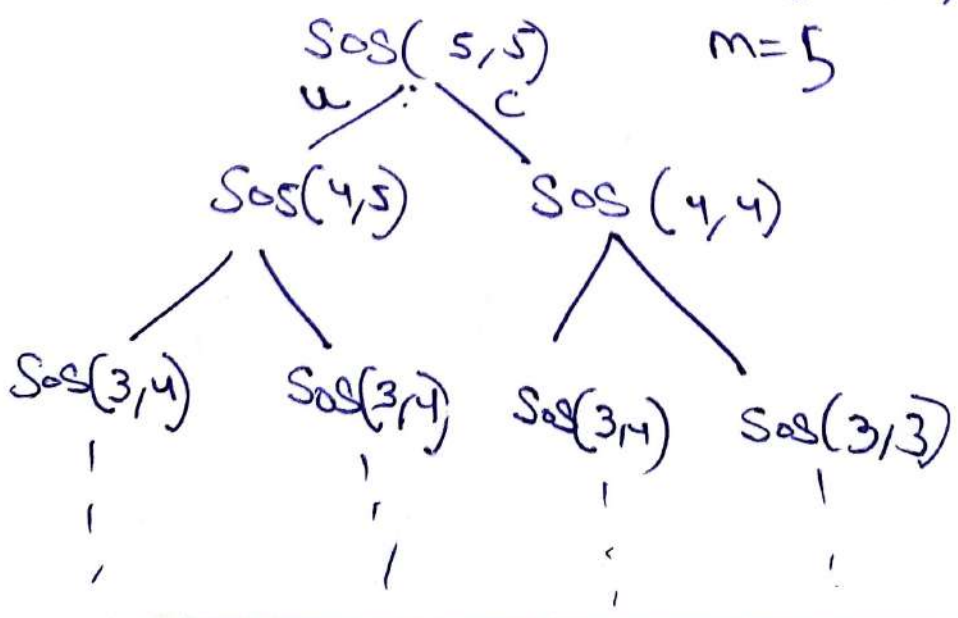
$$SOS(n,m) = \begin{cases} SOS(n-1, m) \rightarrow \text{not consider } n\text{th element} \\ \quad \text{OR} \\ SOS(n-1, m-w_n) \rightarrow \text{consider } n\text{th element} \\ \qquad w_n \rightarrow n\text{th weight} \\ SOS(n-1, m) \rightarrow w_n > m \\ n=0 \text{ AND } m=0 \rightarrow \text{termination condition} \\ n=0,\ m \neq \text{false} \\ n \neq 0,\ m=0 \ \cancel{\text{false}} \text{ True, don't consider any element} \end{cases}$$

→ Recursive Tree

Let us take example

$S = \{1, 1, 1, 1, 1\}$

$m = 5$



Total possible fn call = 2^n

Total depth of tree = $O(n)$

$\therefore$ TC = $O(2^n)$

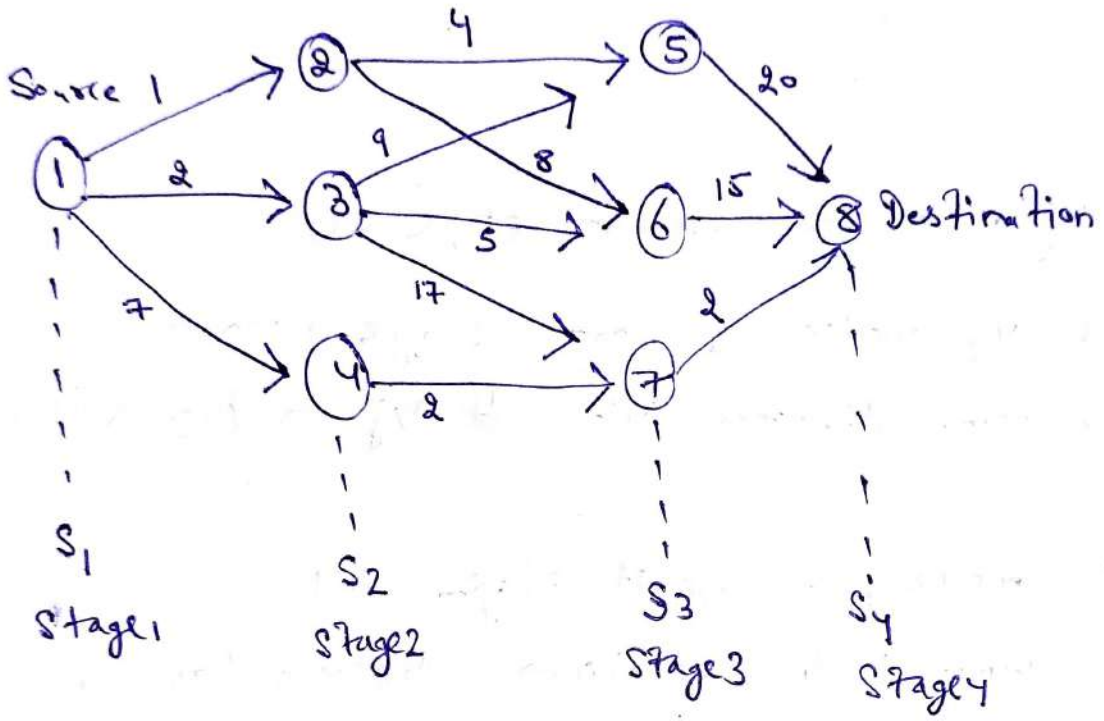
But there are also some repeative fn calls. $\therefore$ T.C $\downarrow$

Unique fn calls = $O(n.m)$

$\therefore$ T.C = $O(n.m)$ if m is less

Lec- 51

Multistage Graph in Dynamic Programming



Multistage Graph :- The graph having only 1 vertex at source, Destination stage and intermediat stage vertex should not connected directly to each other.

Now our problem is that from source to destination our cost should be minimum.

Acc to greedy approach

the Path = $1 \xrightarrow{1} 2 \xrightarrow{4} 5 \xrightarrow{20} 8$

cost = 25

Let do Acc to dynamic Programming

Recursive Equation :-

$$fun(S_i, V_j) = \min_{\forall} \left\{ fun(S_{i+1}, K) + C(V_j, K) \right.$$

$fun(S_i, V_j)$ means, we are at stage S_i, and V_j vertex of that stage, what is min. distance b/w ~~set~~ V_j to Destination

$fun(2,3)$ means, we are at stage 2 and what is the minimum distance between vertex 3 and destination

→ S_{i+1} is next stage to S_i

K is vertex of S_{i+1} stage

$C(V_j, K)$ is cost from vertex V_j to vertex K

Suppose $(S_i, V_j) = (1, 1)$

then $S_i = 1$

$V_j = 1$ and $K = 2$ OR 3 or 4

we will choose K forwhich $C(V_j, K) = \min$

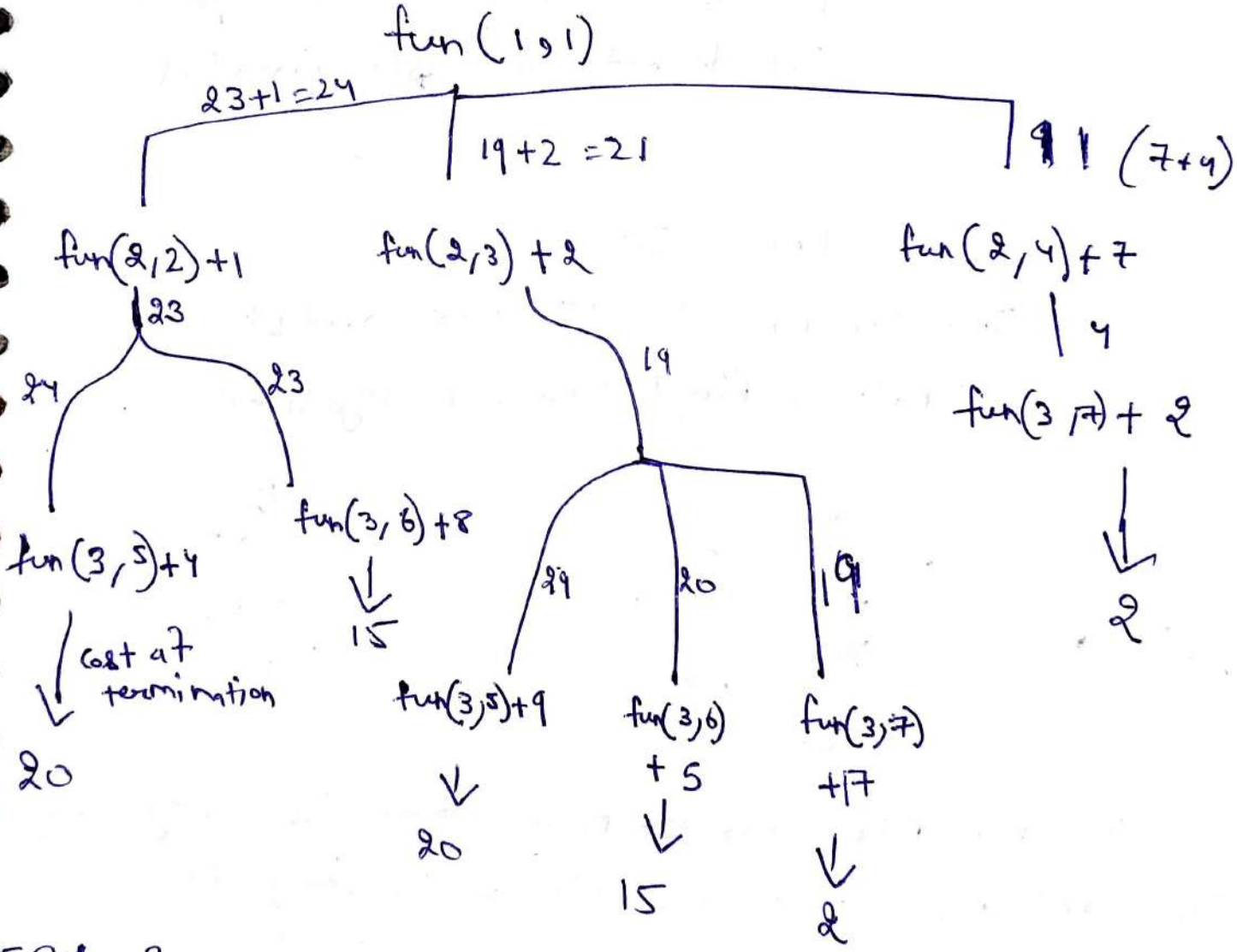
termination condition when we will

$at_j = S_{final} - 1$

$cost = C(v_j, D)$

$D \rightarrow$ Destination vertex

→ Lets solve example



out of 24, 23 → min → 23+1 = 24
out of 29, 20, 19 → min → 19+2 = 21
out of 7 → min → 7+4 = 11
} min is 11

$\therefore$ minimum cost = 11

Path is $① \xrightarrow{7} ④ \xrightarrow{2} ⑦ \xrightarrow{2} ⑧$

Time complexity = $O(V+E)$

$E > V$

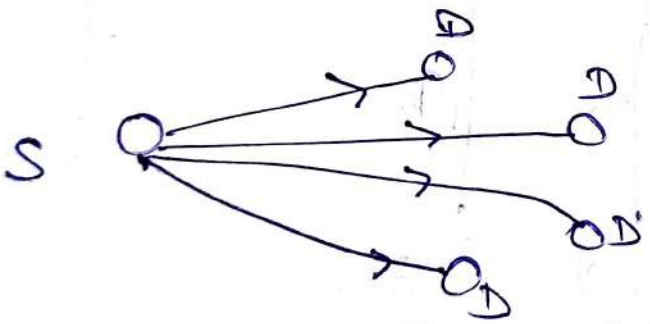
$\therefore O(E)$

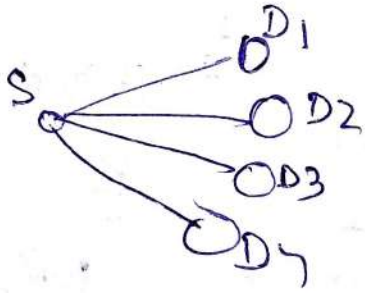
As we are traversing throught each vertex only once and some fn are repeative

$\therefore$ T.C do not go upto exponential

Lec - 52

Introduction to All Pair Shortest Path (Floyd Warshall Algorithm)





In All Pair Shortest Path there is only one source and left nodes are all Destination, Here we will from source to all destinations in shorest path. In all Pair we need ~~A to~~ shortest path from D_1 to all Destinations we need shorest Path

Acc to Dijkstra Algo

$T.C = V^3 \log V$

Acc to Bell Man ford

$T.C = V^4 \log V$

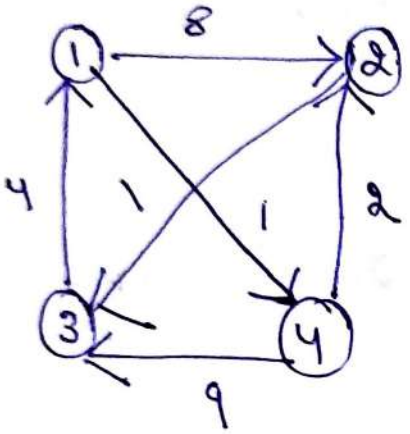
which is very high.

So with the help of dynamic Programming

T.C will reduced to V^3

Lec- 53

Floyd warshall working with Example (All Pair Shortest Path Algorithm)



we have given a directed graph and we will make distance matrix

→ Distance Matrix D_0

$$D_0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ 4 & \infty & 0 & \infty \\ \infty & 2 & 9 & 0 \end{bmatrix} \end{array}$$

If direct connected b/w two vertex → write distance
If not " " " " " → write ∞ "

→ D_1 Distance Matrix

In this matrix we can write distance b/w two vertex via (1) vertex

for Ex :- $2 \xrightarrow{x} 3$ It Possible, we can write $2 \xrightarrow{a} 1$ $1 \xrightarrow{b} 3$

In two cases distance = k and a+b
we will prefer minimum.

$$D_1 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 8 & \infty & 1 \\ 2 & \infty & 0 & 1 & \infty \\ 3 & 4 & 12 & 0 & 5 \\ 4 & \infty & 2 & 9 & 0 \end{array}$$

from 3 to 2 via 1 distance = 4 + 8 = 12
" " to 4 via 1 " = 4+1 = 5

→ D_2 Distance Matrix → Here we can choose path via vertex ①, ② both if Possible.

$$D_2 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 8 & 9 & 1 \\ 2 & \infty & 0 & 1 & \infty \\ 3 & 4 & 12 & 0 & 5 \\ 4 & \infty & 2 & 3 & 0 \end{array}$$

→ D_3 distance Matrix → means we can choose Path through vertex ① ② ③

$$D_3 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 8 & 9 & 1 \\ 2 & 5 & 0 & 1 & 6 \\ 3 & 4 & 12 & 0 & 5 \\ 4 & 7 & 2 & 3 & 0 \end{array}$$

D_4 matrix → we can choose minimum distance path via ① ② ③ ④

$$D_4 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & 4 & 1 \\ 2 & 5 & 0 & 1 & 6 \\ 3 & 4 & 7 & 0 & 5 \\ 4 & 7 & 2 & 3 & 0 \end{array}$$

Now D_4 is Final Result

In D_4 the shortest Path b/w any two nodes is available

Lec-54

→ Time complexity

$$A^k[i,j] = \min \left\{ \begin{array}{l} A^{k-1}[i,j] \\ A^{k-1}[i,j] + A^{k-1}[k,j] \end{array} \right\}$$

For Example

$$A^2[4,3] = \min \left[\begin{array}{ll} 9 & = 9 \\ 2+1 & 3 \end{array} \right]$$

$$= 3$$

In D_0 matrix we don't need this eq.

but D_1 we need it

In D_1 there are n^2 no of element

$\therefore$ for one matrix it take $O(n^2)$

And we do it upto D_4

ie $4n^2$

but in case of n no of vertex in a graph

$T.C = O(n\,n^2) = O(n^3)$

→ space complexity

here we need space for only 2 matrix

because initiall we make D_0, which require space $= O(n^2)$

Now with the help of D_0, we make D_1 which require (n^2)

Now To make D_2 we need D_1 and space we can use of D_0 by removing element from it

$\therefore$ Space comple $= O(2n^2)$

$= O(n^2)$

Lec - 55

Hashing :- Storing and Retrieving data in O(1) time.

→ we are having some ~~on~~ terms given related to hashing

① Search Key (24, 52, 91, 67, 48, 83)

=) It is defined as keys or values given which needs to be store in hash table and delete, or search from hash table.

② Hash Table :- It is like data structure which stors value given in search keys.

③ Hash function (Kmod10, Kmodn, Mid square, Folding methods)

=) These function helps to be done all operation in O(1) time.

operations like insertion, deletion, searching.

Most used fn = Kmod10

→ K mod 10

In insertion to insert 24

24 mod 10 = 4

∴ Hash value = 4

It means at index 4 in table insert 24

To delete 24 24 modeto = 4

delete value from index 4

→ Mid Square means

Suppose Key = 123

So Squar of mid = $2^2 = 4$

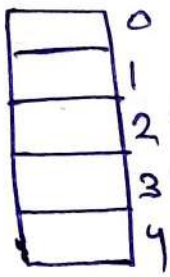
means insert 123 at 4 index

→ Folding Method

Suppose Key = 123456

$\therefore$ 123
456
579

means insert 123456 at index 579



Hash table

Lec - 56

Collision in hashing

→ Let see what is collision in hashing

Kmodn ⇒ we will make hash table from index 0 to n-1

K mod 5

∴ Hash table

⇓

0	24
1	19
2	32
3	
4	
5	

Keys → 24, 19, 32, 44

So 24 mod 5 = 0

∴ Put 24 at 0 index

19 mod 6 = 1

32 mod 6 = 2

44 mod 6 = also 2

So at index 2 44 also should be insert but 32 is already there ∴ There happens a collision b/w 32 and 44

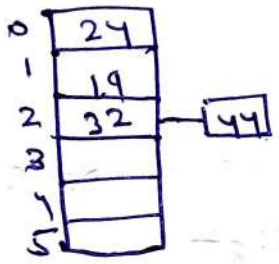
→ Collision Resolution Technique

- Chaining (open hashing)
- Open Addressing (Closed Hashing)
  1. Linear Probing
  2. Quadratic //
  3. Double //

→ Chaining (open Hashing)

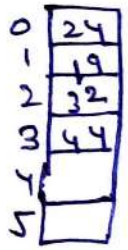
- Here we will add key as a linklist in a open chain

for ex:- 44



→ Open Addressing

① Linear Probing → we will insert key in next Available space



② Quadratic Probing →

Here we will use $h + i^2 \bmod n$

h → hashvalue

i → Probing value

suppose we key = 30

$\therefore 30 \bmod 6 = 0$

$\therefore h = 0, i = 1$

$\therefore 0 + 1^2 \bmod 6 = 1$

but at index 1 no empty space

Now $i = 2$

$0 + 2^2 \bmod 6 = 4$

So insert 30 at index 4

Lec- 57

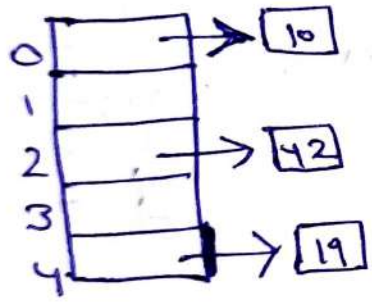
Chaining in Hashing

Keys :- 42, 19, 10, 12

K mod 5

Here we will connect keys as a node in the form of link list.

In case of collision upcoming node will be insert at the beginning.

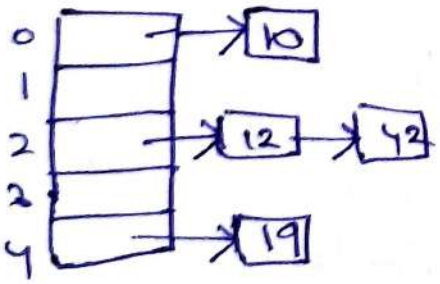


42 mod 5 = 2

19 mod 5 = 4

10 mod 5 = 0

12 mod 5 = 2



→ Advantages of Chaining

Deletion is easy

Insert is " in O(1) time

To delete 12, free 12 and index 2 will store address of 42

→ Disadvantag

Searching is taking O(n) in worst case.

It Take extra space

→ Load factor $\alpha = \frac{\text{no of Element}}{\text{no of slots}}$

$\Rightarrow \frac{4}{5}$

Lec-58

Linear Probing in Hashing

$h(k) = k \bmod 10$

Here if collision occurs, insert key in next upcoming empty slot

And use also a function

$h'(k, i) = (h(k) + i) \bmod 10$

Keys:- 43, 135, 72, 23, 99, 19, 82

0	19
1	
2	72
3	43
4	23
5	135
6	82
7	
8	
9	99

- $43 \bmod 10 = 3$
- $135 \bmod 10 = 5$
- $72 \bmod 10 = 2$

- $99 \bmod 10 = 9$
- $19 \bmod 10 = 9$

next empty slot = 0

- $82 \bmod 10 = 2$

next empty slot = 6

Probe no of 23 = 2
" " " 19 = 2
" " of 82 = 5

- $23 \bmod 10 = 3$

but At 3 already 43 present
∴ collision occurs
next empty slot is 4
∴ insert 23 at 4

OR

$h'(k, i) = (h(k) + i) \bmod 10$
$= (3 + 1) \bmod 10$
$= 4$

i = probe no. means how many times we tried to insert in table

i is different for diff keys.

Advantage → No Extra space require

Disadvantage → Searching time ⇒ O(n) in worst case

Best, Average case = O(1)

→ Deletion difficult

For ex :- 43 is deleted

and then 82 had to be deleted

So 82 mod 10 = 2, but 72 is there

and at 3, slot is empty,

Empty indicates 82 is not present

if 82 is present, it should be present in 3

∴ our searching for deletion stops.

→ Primary Clustering :- It ↑ tendency to inserting element at a particular position.

→ Secondary Clustering :- When 2 or more element follow the same probe sequence.

Lec- 59

Ques on Hashing | Linear Probing

Q) The Keys 1, 3, 12, 4, 25, 6, 18, 20, 8 are inserted into empty hash table of length 10 using open addressing with hash functions $H(i) = i^2 \bmod 10$ and linear Probing. What is the resultant hash table and find the max. probe value?

=) $H(i) = i^2 \bmod 10$

(P.V)

for 1 → $1^2 \bmod 10 = 1$, Probe value = 1

for 3 → $3^2 \bmod 10 = 9$, P.V = 1

for 12 → $12^2 \bmod 10 = 4$, P.V = 1

for 4 → $4^2 \bmod 10 = 6$, P.V = 1

for 25 → $25^2 \bmod 10 = 5$, P.V = 1

for 6 → $6^2 \bmod 10 = 6$, But at 6, 4 is Present P.V = 1, Now next empty is 7, ∴ PV = 1+1 = 2

Insert 6 at 7

for 18 → $18^2 \bmod 10 = 4$, But at 4, 12 is Present P.V = 1 next empty slot at 8 ∴ Insert 18 at 8 and P.V = 1+1+1+1+1 = 5

for 20 ⇒ $20^2 \bmod 10 = 0$, PV = 1

for 8 ⇒ $8^2 \bmod 10 = 4$, next empty slot at 2, P.V = 9

0	20
1	1
2	8
3	
4	12
5	25
6	4
7	6
8	18
9	3

Max Probe value = 9

Lec- 60

Quadratic Hashing

Advantage :- No Extra space
Primary clustering ~~[illegible]~~ resolved

Disadvantages :-

→ Searching Time → $O(n)$
→ Secondary clustering
→ No Guarantee of finding slot for inserting key.

Lec - 61

Double Hashing

Collision Resolution Technique

$h_1(k) = k \bmod 11$

$h_2(k) = 8 - (k \bmod 8)$

$\rightarrow [h_1(k) + i\, h_2(k)] \bmod 11$ — (a)

Keys → 20, 34, 45, 70, 56

Here we use 2 hash functions.

- for 20 → $20 \bmod 11 = 9$
- for 34 → $34 \bmod 11 = 1$
- for 45 → $45 \bmod 11 = 1$

but 1 has already 34

∴ Now use $h_2(k)$

$h_2(k) = 8 - (45 \bmod 8)$

$= 8 - 5$

$h_2(k) = 3$

$[h_1(k) + i\, h_2(k)] \bmod 11 = [1 + 1 \cdot 3] \bmod 11 = 4 \bmod 11 = 4$

So insert 45 at 4

- for 70 → $70 \bmod 11 = 4$

but ~~to~~ 4 has already 45

$h_2(k) = 8 - (70 \bmod 8) = 8 - 6 = 2$

0	
1	34
2	
3	56
4	45
5	
6	70
7	
8	
9	20
10	

Acc to (a)

$(4 + 1 \cdot 2) \bmod 11 = 6 \bmod 11 = 6$

- for 56

$h_1(k) = 56 \bmod 11 = 1$

$h_2(k) = 8 - (56 \bmod 8) = 8$

Acc to (a)

$[1 + 1 \cdot 8] \bmod 11 = 9$

but 9 is not empty

$\therefore$ Now $j = 2$

So $[1 + 2 \cdot 8] \bmod 11 = 17 \bmod 11 = 6$

but 6 is not empty

Now $i = 3$

So $[1 + 3 \cdot 8] \bmod 11 = 25 \bmod 11 = 3$

So insert 56 at 3

Advantages :-

No Extra space

No Primary clustering

No Secondary "

Disadvantage

- Searching time is $O(n)$

Lec-63

Radix Sort

Given Keys 904, 46, 5, 74, 62, 1

It is non comparison based algo

It is stable.

	Step1		Step2		Step3		Step4 final
904	904		001		001		001
46	046		062		904		005
5	005	→	904	→	005	→	046
74	074		074		046		062
62	062		005		062		074
1	001		046		074		904
							sorted array

Working :-

In given keys 904 is largest and has 3 no of digit.

→ step 1 → Make all no 3 digits

step2 → Arrange all no acc to unit place of all no. It should relatively stable.

step 3 → Sort them acc to tens place

step4 → Sort them acc to hundred place and then we get sorted array

Lec-64

Counting Sort

Non comparison based algo.

→ Given Input size is 'n'

→ Given Input range is 'k'

Suppose Range is (1-5)
means all elements of input array must belong to (1-5)

Suppose input array is 2, 1, 2, 3, 1, 2, 4

Range is 1-5

So make extra array for 1 to 5
occurrence array

1	~~1~~ 2
2	~~2~~ 3
3	1
4	1
5	0

Extra array is occurrence array in which we define which elements of input array is occurred how many times.

Now write 1, 2, 3, 4, 5 acc to their occurrence in a serial

⇒ 1, 1, 2, 2, 2, 3, 4

So finally we got sorted array

Time complexity = $O(n+k)$

Space " = $O(k)$

lec - 65

Bucket Sort

- Non comparison algo.

→ works on floating-point numbers between range 0.0 to 1.0

→ Inputs should be uniformly and independently distributed across [0,1] to get a running time of O(n)

(*) Avg or Bestcase

Input Array A → 0.79, 0.13, 0.64, 0.39, 0.20, 0.89, 0.53, 0.42, 0.06, 0.94

0	0.06
1	0.13
2	0.20
3	0.39
4	0.42
5	0.55
6	0.64
7	0.79
8	0.89
9	0.94

Working of BUCKET SORT

1) let B [0 -- n-1] be a new array
2) n = A.length
3) for i=0 to n-1
4) make B[i] an empty list
5) for i = 1 to n
6) insert A[i] into list B[⌊nA[i]⌋]
7) for i = 0 to n-1
8) Sort list B[i] with insertion Sort
9) concatenate the lists together B[0], B[1] - - B(n-1) in order

T.C = O(n)

0.06, 0.13, 0.20, 0.39, 0.42, 0.55, 0.64, 0.79, 0.89, 0.94

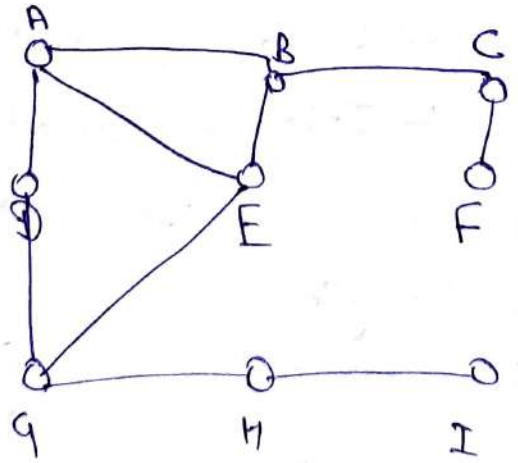
Worst case

If elements are like 0.79, 0.78, 0.795, 0.79

then apply (8) insertion sort internally

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

$= O(n^2)$



into memory.

(5) Adaptive → Already sorted data takes less time

(6) Recursive / Non Recursive SA → Recursive, if it uses recursion otherwise not.

Lecture - 50

Bubble sort Algorithm Explanation.

0	1	2	3	4	5
7	11	9	2	17	4

We have to sort it in ascending order

our result after sorting =

0	1	2	3	4	5
2	4	7	9	11	17

Here we have to do some Passes

Ist Pass → There are some steps in which we have to compare element

For, in first step we compare 0, 1 element

ex If 0th element < 1 th element then ok; if not we swap the element.

	0	1	2	3	4	5	
Ist Pass	7	11	9	2	17	4	→ 0, 1 compare
	7	9	11	2	17	4	→ 1, 2 "
	7	9	2	11	17	4	→ 2, 3 "
	7	9	2	11	17	4	→ 3, 4 compare
	7	9	2	11	4	17	→ 4, 5 "

अब Ist Pass का motive है कि हमने 17 जो सबसे बड़ा no the उसको ठिकाने लगा दिया, सबसे last में

1st Pass

Now same steps have to follow in 2nd Pass, Here we compare element from idex 0 to 4 because 1 element has sorted i.e 17.

2nd Pass → 7 9 2 11 4 17 → 0,1 compare
7 2 9 11 4 17 → 1,2
7 2 9 11 4 17 → 2,3
7 2 9 4 11 17 → 3,4

2nd Pass completed

3rd Pass result 2 4 9 11 17

4rt Pass result → 2 4 7 9 11 17

5th Pass result → 2 4 7 9 11 17

Now we have sort our array.

If n is no of element in array or n is length of array then no of Passes = n−1

In first Pass → 5 comparison (n−1)
5 Possible swap

In 2nd Pass → 4 comparison (n−2)
4 Possible swap

In 3rd Pass → 3 comp. (n−3)
3 swap

In 4th Pass → 2 comp.
2 swap

In 5th Pass → 1 comparison
1 swap.

Sum of
(Total no of comparison

$\Rightarrow 1 + 2 + 3 + \cdots n-1 = \frac{n(n-1)}{2} = O(n^2)$

It is stable algorithm

By default if algorithm is not adaptive, it can be made adaptive

Why it is called bubble sort?

⇒ भारी element को एक एक करके end में फेंकों और हल्के element bubble up होते रहते है Right to left.

Lecture - 51

Bubble Sort code in C

```
#include<stdio.h>
void printarray(int *A, int n)
{
    for(int i=0; i<n; i++)
    {
        printf("%d \t", A[i]);
    }
    printf("\n");
}
void bubblesort(int *A, int n)
{
    int temp;
    for(int i=0; i<n-1; i++) // for pass
{
    printf("working on pass no %d\n", i+1);
    for(int j=0; j<n-1-i; j++) // for comparison
                                  in each pass
    {
        If(A[j]>A[j+1])
        {
            temp=A[j];
            A[j]=A[j+1];
            A[j+1]=temp;
        }
    }
}
}
```

```
int main ()
{
    int A[] = {2, 3, 4, 1, 5};
    printf("Array before sorting");
    printarray[A, n];
    bubblesort [A, n];
    printf(" Array after sorting");
    printtArray [A, n];
    return 0;
}
```

Output

Array before sorting

2 3 4 1 5

working on Pass no 1,2,3,4 in four lines

Array after sorting

1 2 3 4 5

Code for Adaptivity means already sorted array.

we write same print array fn.

and made changes in sort fn.

```
Void adap bubblesort (int *A , int n)
{ int temp;
  int sorted = 0;

for (int i=0; i<n-1 ; i++)
{       sorted = 1; - printf(" workon Pass %d", i+1);
   for (int j=0; j<n-1-i; j++)
   {
      If ( A[j] > A [j+1])
      {temp = A[j];
       A[j] = A[j+1];
       A[j+1] = temp;
       sorted = 0;
      }
   }
   If (sorted)
    return;
}
}

int main ()
{int A[] = { 1, 2, 3, 4, 5};
   int n = 5;
   printf (" Befor sorted");
   printarray (A, n);
   printf (" After sorted");
    printarray (A ,n) ;
   return 0;
}
```

output

Befor sorted

1 2 3 4 5

working on pass 1

After sorted

1 2 3 4 5

Lecture - 52

Insertion sort Algorithm

0	1	2	3	4
7	12	3	4	1

Length = 5

Sorted | Non sorted

Ist task 7 | 12 3 4 1

अब हमे इस line को एक एक करके आगे बढाना है।

Ist Pass ⇒

7	12	3	4	1

इस पास मे हमे 12 अगर 7 से बडा को वही रहेगा, और नही तो swapping हो जाएगी।

Result

Sorted		Non-S		
7	12	3	4	1

2nd Pass ⇒ 7 12 3 | 4 1

पहले 3, 12 से compare होगा

After comparision 7 3 12 | 4 1

अब 3, 7 से compare होगा

Result ⇒

Sorted			N-S	
3	7	12	4	1

4 7 12

3rd Pass

3 7 12 | 4 | 1

Sorted | Nons

Result ⇒

3	4	7	12	1

4th Pass Result

1	3	4	7	12

1, सबसे compare होकर 3 वाली जगह पर insert होता रहेगा।

and finally we get sorted array

Pass	Pass Swap	Pass comparison
Ist	1	1
2nd	2	2
3rd	3	3
4th	4	4

Total ⇒ n-1 Pass

Total Pass, Comp Pass = $1+2+3 \ldots (n-1)$

$\Rightarrow \frac{n(n-1)}{2}$

W.C T.C = $O(n^2)$

Best case → $O(n)$

Stable → Yes

Adaptive → Yes

Intermediate result in not useful

Lecture no - 53

Insertion Sort code

```
#include<stdio.h>
void printarray (int *A, int n)
{
    for(int i=0; i<n; i++)
     {  printf("%d", A[i]);
     }
     printf("  ");

}

void insertsort (int *A, int n)
{
    int key, j;
   for(int i=1; i<=n-1; i++) //for Passes
   {    key = A[i];
        j = i-1;
    while (j>=0 && A[j]>key)//for each pass
    {
        A[j+1] = A[j];
        j--;
    }
    A[j+1] = key;
   }
}
```

Lecture - 54

Selection Sort Algorithm

We have array ⇒

0	1	2	3	4
8	0	7	1	3

Length = 5

we have to sort in ascending order by Selection sort Algorithm.

Ist Pass

we assume 0th index element = minimum

0	1	2	3	4
8	0	7	1	3

↑ min

Now we compare ~~with~~ 0, 1 element, now minimum is 1th element

8	0	7	1	3

↑ min

Now we compare 0 with 2nd, 3rd, 4th element but 0 is minimum

So minimum = 0

Now we swap Now minimum and previously assumed minimum

we get

0	8	7	1	3

N.S

अब 0 index element has no role

2nd Pass

Now in N.S Part

we assume 1 th element = minimum

0	8	7	1	3

↑ min

Now we compare it with 2nd element and found now minimum = 7

0	8	7	1	3

min

and now we compare 7 with 3rd element and found now minimum = 1

0	8	7	1	3

min

Now we compare 1 with 4th element but minimum = 1

∴ finally minimum = 1

and in same way we exchange the element

and get

0	1	7	8	3

N.S

3rd Pass result =

0	1	3	8	7

N.S

4th Pass result =

0	1	3	7	8

we don't need 5th Pass

Total no of Pass = $n-1$

n = length of array

Pass	Poss comp
1	4
2	3
3	2
4	1

W.C TC = $O(n^2)$ ca

Best case TC = $O(n^2)$ $O(1)$ swaps

Average case TC = $O(n^2)$ $O(n)$ swaps

worst case TC = $O(1)$ auxillary

Total Poss. comparison = $1 + 2 + 3 + \cdots (n-1)$

$= \frac{n(n-1)}{2} = O(n^2)$

Maximum swaps = $n-1$

It is not stable It is inplace

It is not adaptive

Sorting in minimum no of swaps

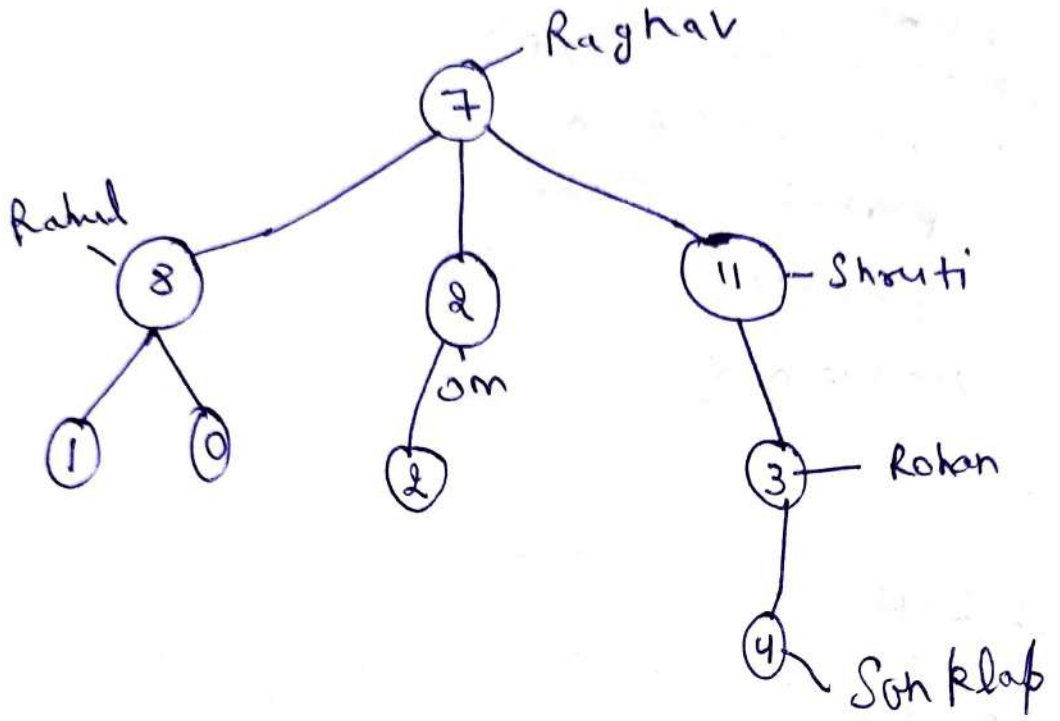
Lecture-55

Selection Sort Code

```
#include<stdio.h>
void printarray (int * A, int n)
{
    for (int i=0; i<n; i++)
    {
        printf("%d", A[i]);
    }
    printf("\n");
}
```

Lecture - 61

Introduction to Trees



Terminology

1. Root ⇒ Topmost Node
   ie Raghav is Topmost Node

2. Parent ⇒ Node which connects to the child
   ie Raghav is parent of Shruti
   Shruti is " " Rohan

3. Child ⇒ Node which is connected by another node is its child
   ie Shruti is child of Raghav
   ie Rohan is child of Shruti

4. Leaf/ External node ⇒ Nodes with no children
   Terminal
   ie Sankalp is External Node

(5) Internal code ⇒ Nodes with atleast one child.

(6) Depth; No of edges from root to the node

ie Depth of raghav = 0
Depth of Shruti = 1
" " Rohan = 2

(7) Height = No of edges from node Deepest leaf!

Height of Raghav = 3
" " Shruti = 2

(8) Sibling = Nodes belonging to the same parent

Rahul, Om, Shruti are sibling

(9) Ancestor → Parent or Parents of Parent

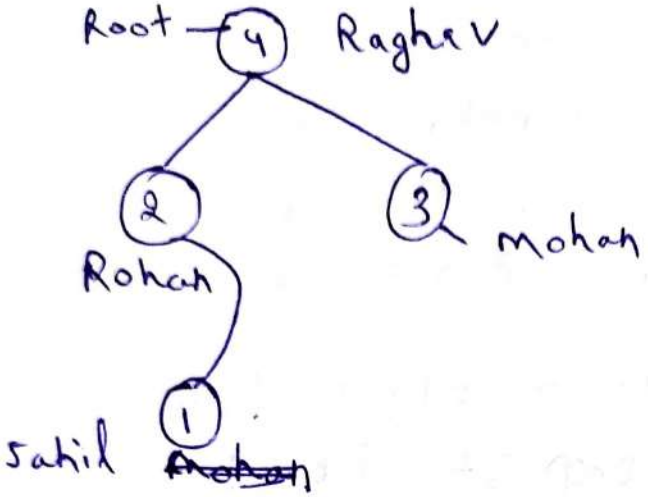
(10) Descendants → child or Childs of child

(11) Degree → no of immediate Child to node

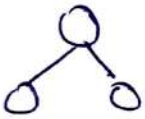
Degree of Raghav = 3

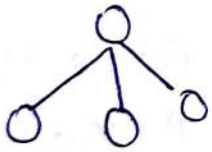
Binary Tree

Binary tree is a tree which has atmost two children. or has degree <=2



Raghav degree = 2
Mohan = 0
Rohan = 1

 = Binary = Binary

 = Not Binare (because Degree = 3 of Root)

→ Tree is made up of nodes and edges!

→ n nodes = n-1 edges

→ Degree = no of direct children (for a node)

→ Degree of a tree is the highest degree of a node among all the nodes present in the tree.

→ Binary Tree = Tree of degree <=2

Lecture - 03

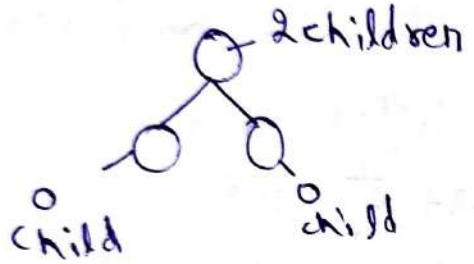
Types of binary tree

Tree = Non linear Data structures
Array, stack, queue, Linked List ⇒ Linear

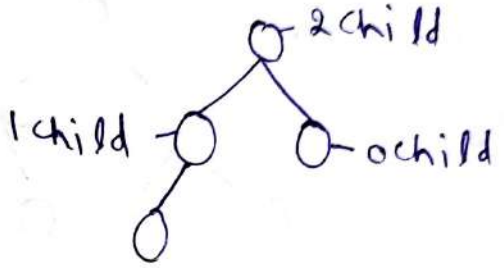
Type of binary tree

① Full or Strict Binary tree

⇒ All nodes have either 0 or 2 children



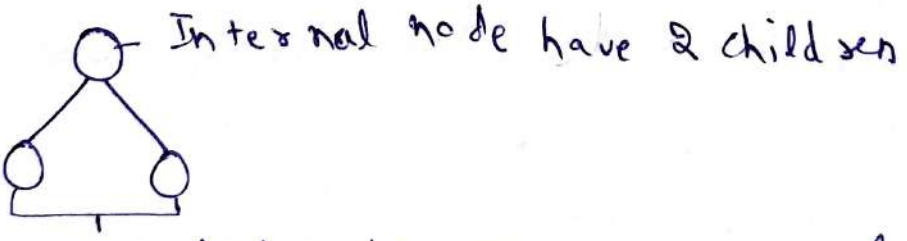
Full or Strict binary tree



Not Full or strict binary tree

② Perfect binary Tree

Internal nodes have 2 children and all leaf nodes are on same level



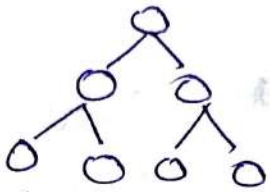
both leaf nodes are on same level

Every level has maximum no of nodes

(3) Complete binary tree

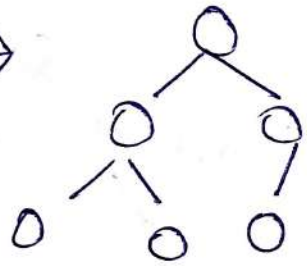
⇒ All levels are completely filled except possible the last level + last level must have its keys as left as possible.

Completebinary tree →



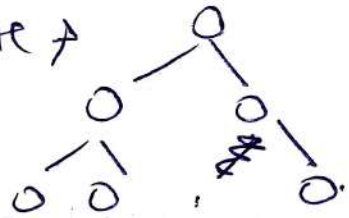
All level are filled

Complete binary tree →



All level not filled but जो filled है वो left में है और last level में जो, fill left में है।

Not complete binary tree →



(4) Degenerate tree

⇒ Every Parent node has exactly one child

Ex

Degenerate

left skewed

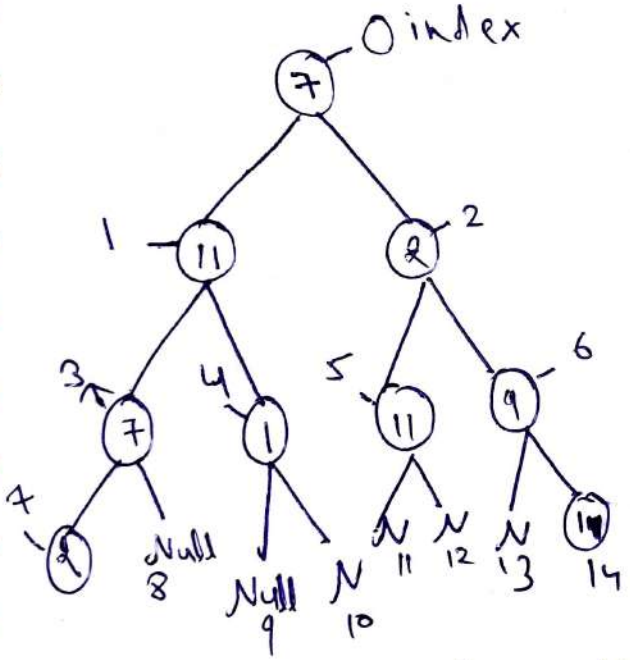
Right squed

(6) Balanced BT (AVL)

Lecture - 64

Representation of a binary Tree

(1) Array Representation



we need array of size 15

0	1	2	3	4
7	11	2	7	1

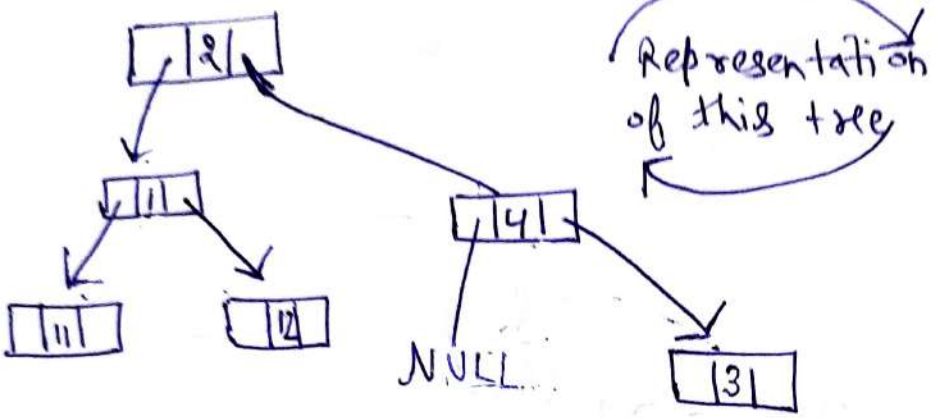
- -

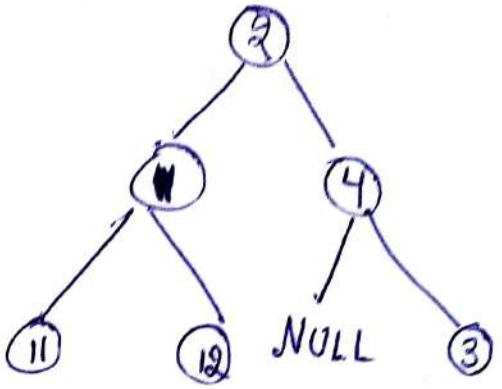
but there is disadvantag of this representation

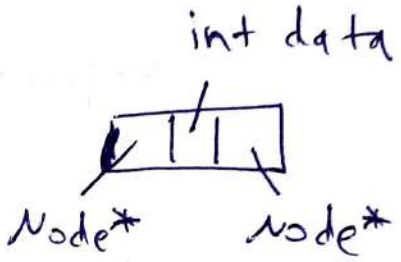
If we want to add 2 more node and store 2 element, we can't do it because our array size does not permit this

(2) Linked Representation

⇒ To avoid that disadvantage we have this








```
struct Node{
  int data;
  Struct Node * left
   "      "  * right
```