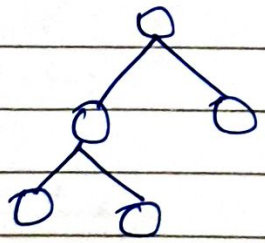
Heap Sort:

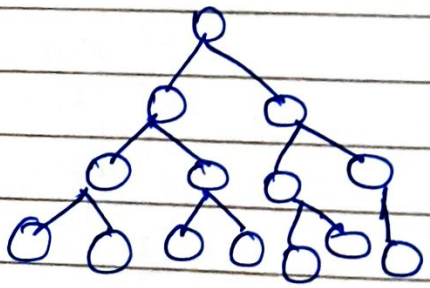
Heap: It is a sorting algo. which uses heap data structure to sort the elements either in ascending or descend. order.

- Heap is a complete binary tree with the following properties:

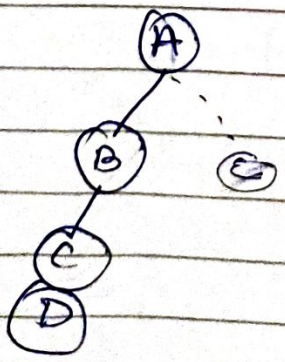
1. Shape Property : Complete Binary tree.

A ~~Complete~~ binary tree of height 'h' is said to be complete binary tree if: upto the height (h-1) it is full with nodes and the height (h) the nodes present must be from left towards ~~big~~ right.





left skewed tree:



In array:

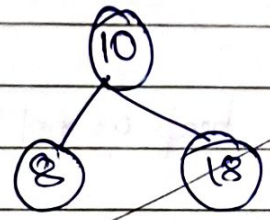
A	B	X	C	X	X	X	D

∴ Array wastages lot of space

So, linklist is used (generally)

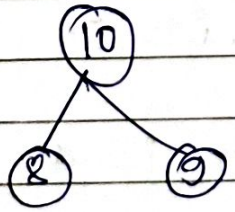
2. Order Property:

min heap / max heap { The left child has lesser value than the parent & the right has the higher value than the parent.

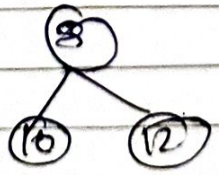


max heap

max heap { The parent has higher value than its child node.

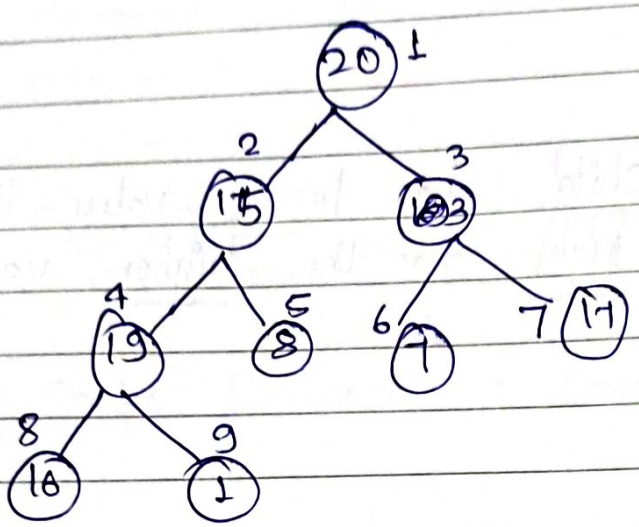


min heap { The parent has lesser value than the child node.



Q. Create a max heap from given elements.

20, 15, 3, 19, 8, 7, 17, 16, 1.



max heap or not?

for index i=9 → ✓ (1)

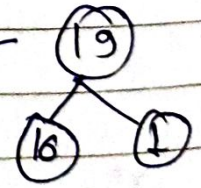
for index i=8 → ✓ (16)

for index: i=7 → ✓ (17)

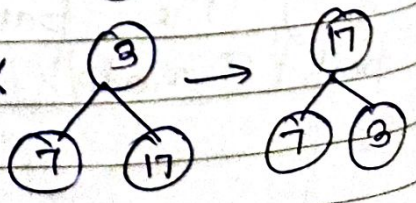
for index i=6 ✓ (7)

for index i=5 ✓ (8)

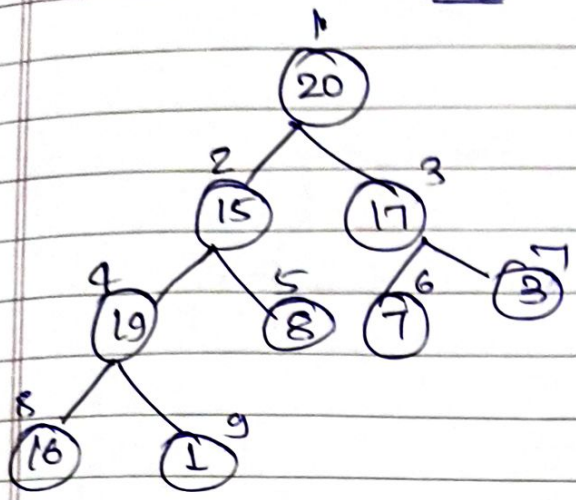
for index i=4 ✓



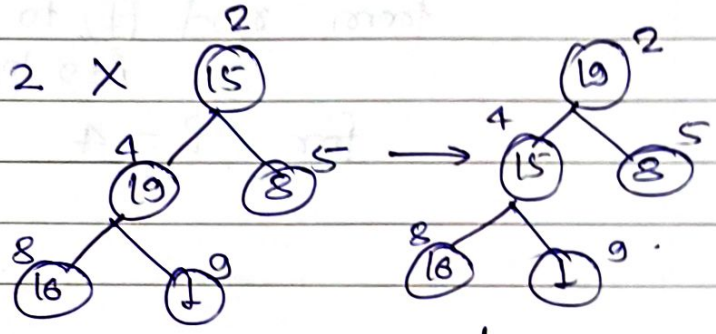
for index i=3 ✗



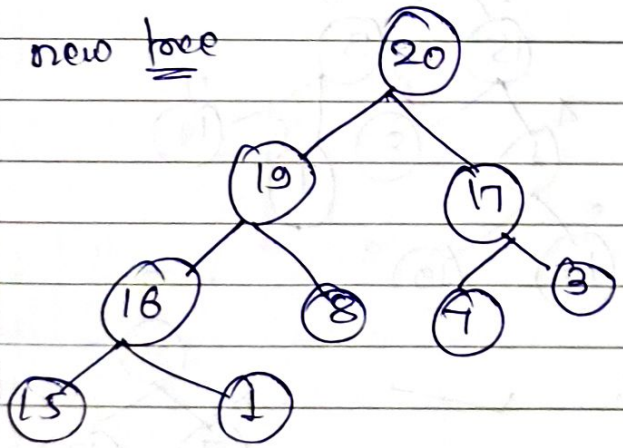
So, $i = 3$ tree.

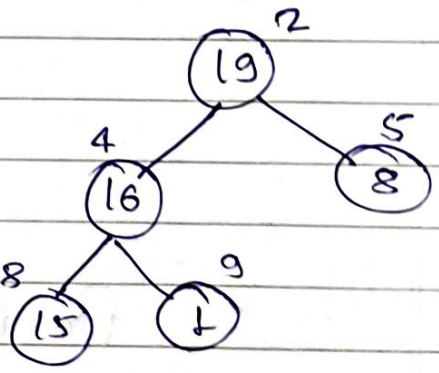


for index $i = 2$ ✗

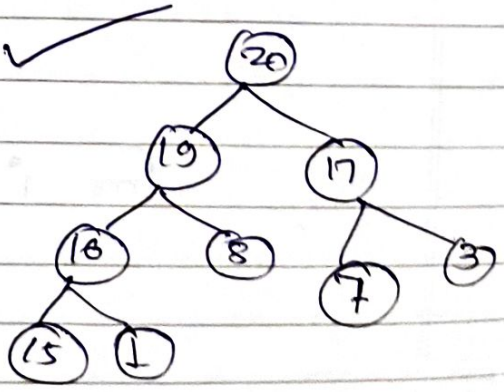


new tree



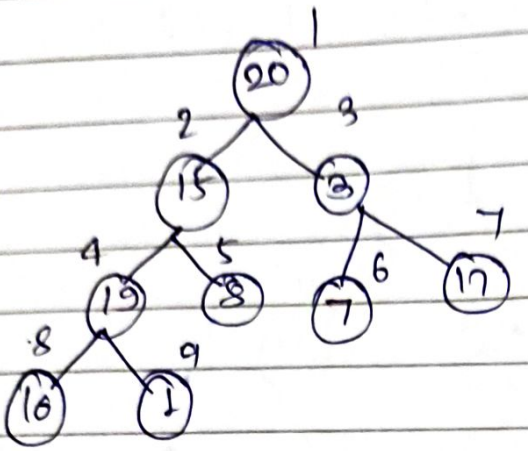


∴ for index $i = 1$: ✓



⟶ max heap

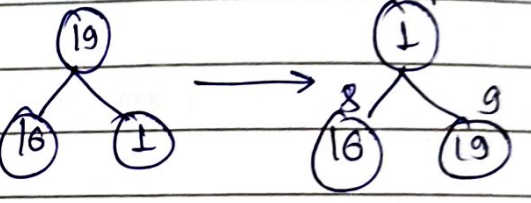
min heap



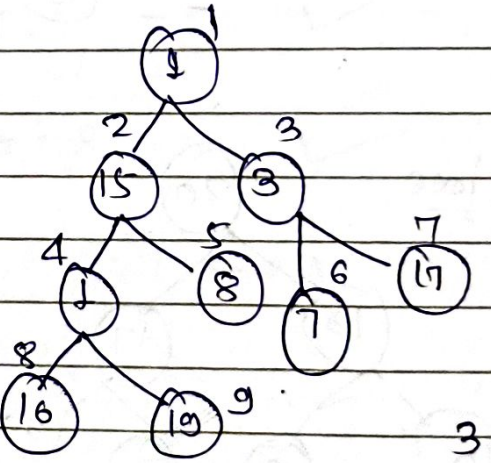
from node 1 to 5 it satisfies

i=9 to i=5

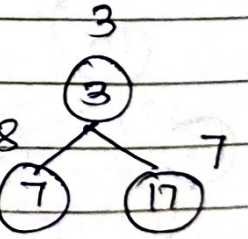
for i = 4 X



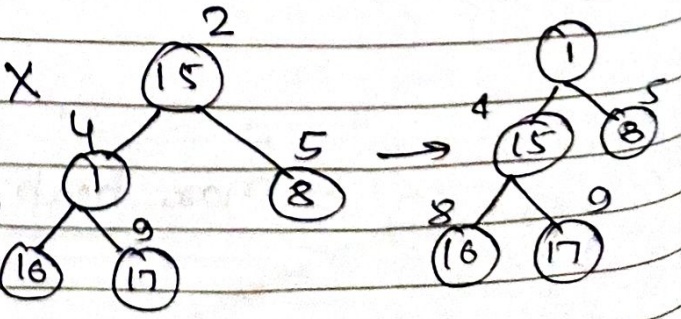
free!



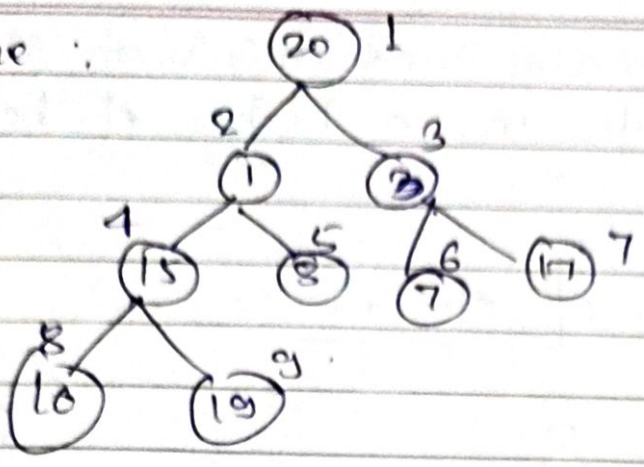
for i = 3 : ✓



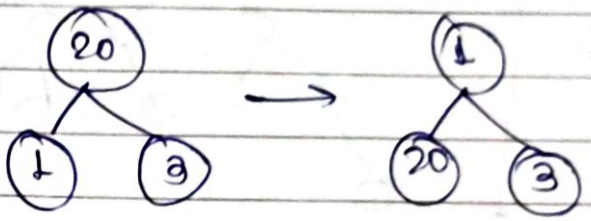
for i = 2 X



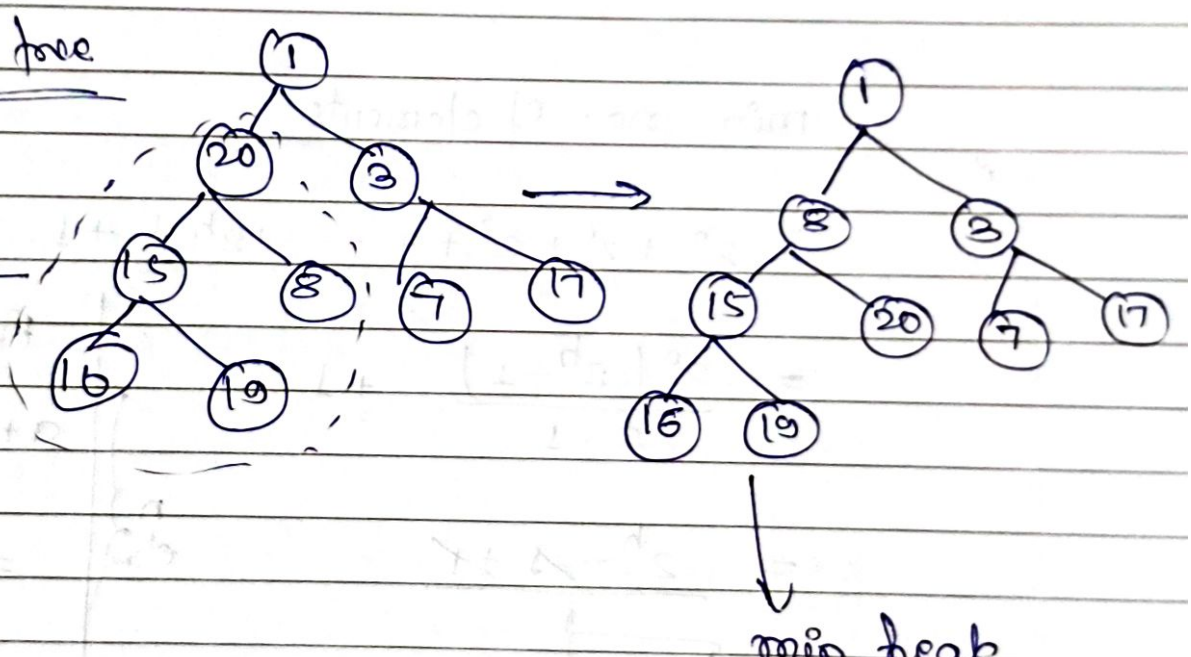
new tree :



for $i = 1$ X



new tree

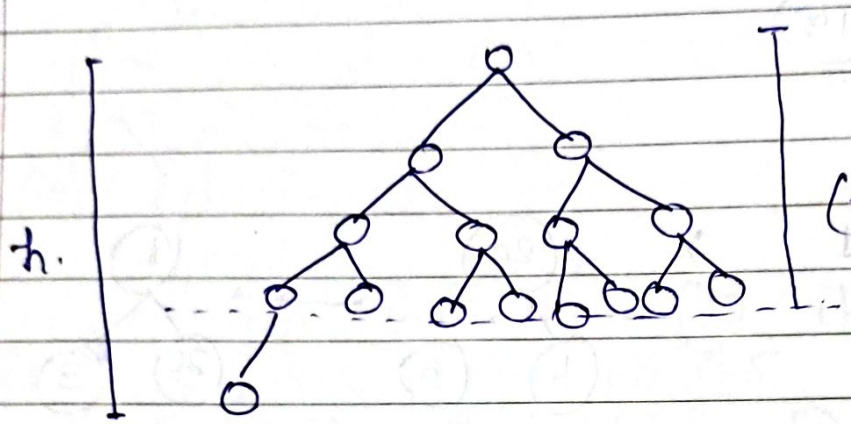


min heap

Q. Find the maximum and minimum number of elements present in a heap of height h?

Soln:

Min. no. of elements:



levels

0 ⇒ 2^0 nodes
1 → 2^1 nodes
2 → 2^2 nodes
3
4 ⋮ 2^{h-1} nodes

min no. of elements

$= 2^0 + 2^1 + 2^2 + + \cdots + 2^{h-1} + 1.$

$= \frac{2^0(2^h - 1)}{2-1} + 1$

$= \frac{2^h - 1 + 1}{1}$

$= \boxed{2^h}.$

∴ minimum $= 2^4 = 16$

Finite G.P.

$a + ar + ar^2 + \cdots + ar^{n-1}$

n elements

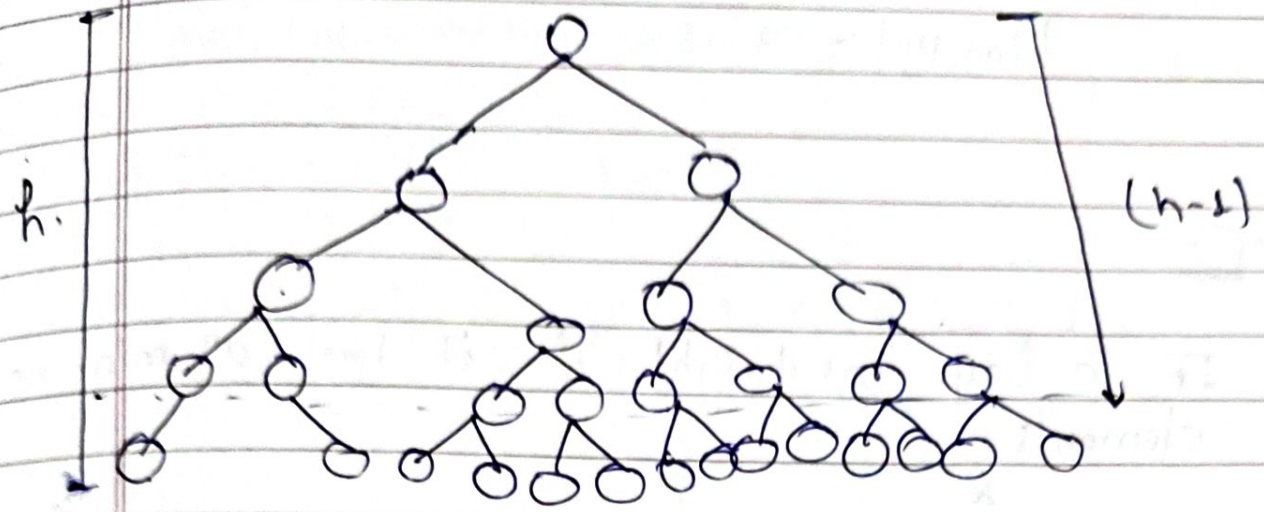
$= \frac{a(r^n - 1)}{(r-1)}$

$a + ar + ar^2 + \cdots + ar^n$

n+1 elements

$= \frac{a(r^{n+1} + 1)}{r-1}$

Maximum number of elements:



Max. no. of element

$= 2^0 + 2^1 + 2^2 + \cdots\cdots + 2^h$

$= 2^0 (2^{h+1} - 1)$

$= \boxed{2^{h+1} - 1}$

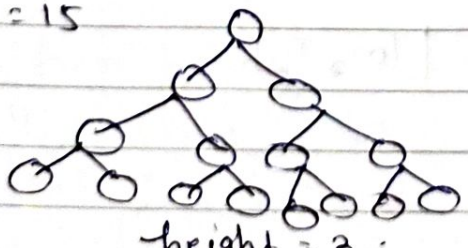
$2^{4+1} - 1$

$= 2^5 - 1 = 32 - 1 = 31$

Q. Prove that an n-element heap has height $\lfloor \log_2 n \rfloor$. where $n = 15$.

Show that 15 element heap has height

$n = 15$



$\lfloor \log_2 15 \rfloor = 3.$

$n = 10$

$\lfloor \log_2 10 \rfloor = 3.$

Proof:

If a heap has height 'h' it has 2^h min. no. elements

&

$2^{h+1} - 1$ maximum number of elements

- Let 'n' be the number of nodes of a heap with height h.

~~$2^h \leq n \leq 2^{h+1} - 1$~~

$$\boxed{2^h \leq n \leq 2^{h+1} - 1}$$

$$\Rightarrow 2^h \leq n < 2^{h+1}$$

($\because 5 \leq 6-1$ True, $\because 5 < 6$)

Taking $\log_2$ in the above inequality.

$$\log 2^h \leq \log_2 n \leq \log_2 2^{h+1}$$

$$\Rightarrow h \log_2 2 \leq \log_2 n < (h+1) \log_2 2$$

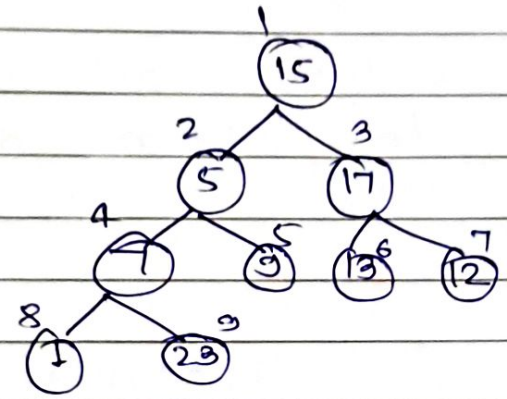
$$\Rightarrow h \leq \log_2 n < h+1$$

$\because 5 \leq 5.2 < 6$
$\therefore 5 = \lfloor 5.2 \rfloor$

$\therefore h = \lfloor \log_2 n \rfloor$

Q. Show that with the array representation for sorting an n-element heap, the leaves are the nodes indexed by $\lfloor \frac{n}{2} \rfloor + 1, \lfloor \frac{n}{2} \rfloor + 2, \ldots, n$.

1	2	3	4	5	6	7	8	9
15	5	17	7	9	13	12	1	23

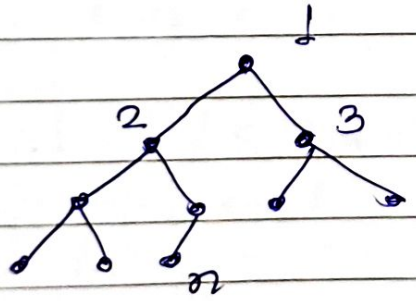


$n = 9$

$\lfloor \frac{9}{2} \rfloor + 1 = 4 + 1 = 5$

$\lfloor \frac{9}{2} \rfloor + 2 = 4 + 2 = 6$

Proof:



$n - \lfloor \frac{n}{2} \rfloor = 9 - \lfloor \frac{9}{2} \rfloor$

$9 - 5$

$= 4 = \lfloor \frac{9}{2} \rfloor$

$= \lfloor \frac{n}{2} \rfloor$.

In a heap (complete binary tree) having 'n' number of nodes, the index of parent of the n^{th} node (last node) is $\lfloor \frac{n}{2} \rfloor$.

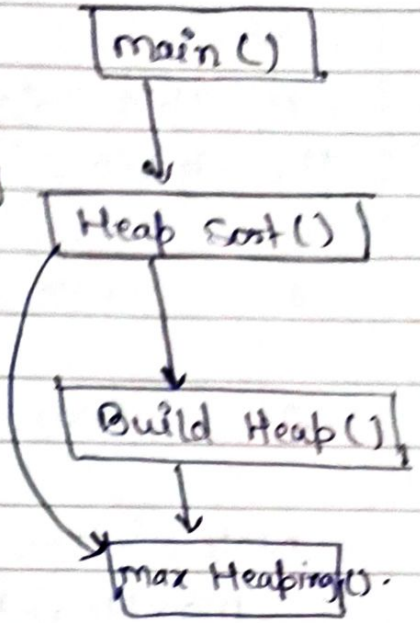
parent of last leaf node, must be the last non-leaf node.

∵ $\lfloor \frac{n}{2} \rfloor$ is the last non-leaf node, the nodes after that are all the leaf nodes and are indexed by $\lfloor \frac{n}{2} \rfloor + 1 \rightarrow \lfloor \frac{n}{2} \rfloor + 2, \ldots$

$\ldots - n$.

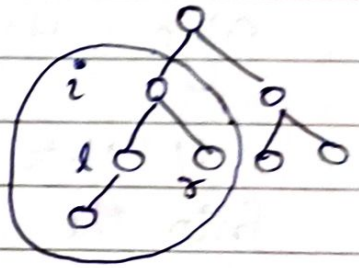
Heap Sort

Arrange the elements either in ascending or descending order in an array.



Algo (Maxheap (A, i))

// Purpose :- this algorithum creates a maxheap at index i.




```
1.   l ← LCHild (i)
2.   r ← RChild (i)
3.   max ← i
4.   if ( l ≤ Heapsize (A) && A[l] > A[i] )
5.       max ← l
6.   if ( r ≤ Heapsize (A) && A[r] > A[max] )
7.       max ← i
8.   if ( i ≠ max )
     {
9.       Swap ( A[i], A[max] )
10.      maxHeapify ( A, max )
     }.
```

Algo BuildHeap (A)

// This algorithum builds a maxheap for the given array A.

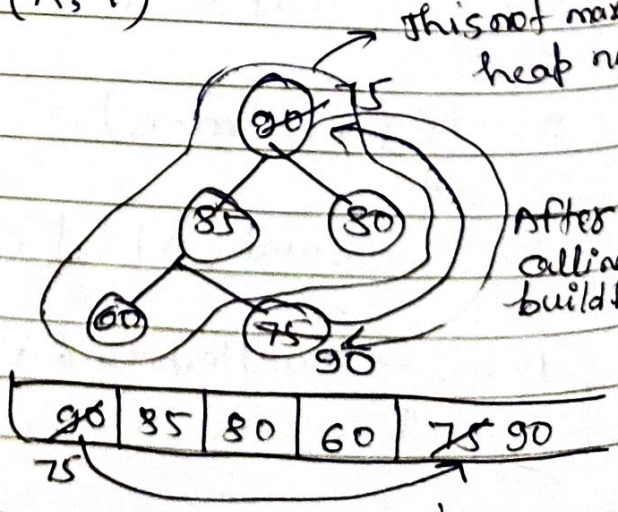
1. for $i \leftarrow \left\lfloor \frac{\text{Heapsize}(A)}{2} \right\rfloor$ down to 2.
2. maxHeaping (A, i)

Note: The Buildheap algorithum calls the maxHeaping () algo. for all the non-leaf nodes.

Algo Heapsort (A)

// this algo sorts the array elements 'A' into the ascending order.

1. Build Heap (A)
2. for i ← heapsize(A) down to 1
   {
3. Swap (A[i], A[heapsize(A)])
4. Heap Size (A) - -
5. max Heapify (A, 1)
   }

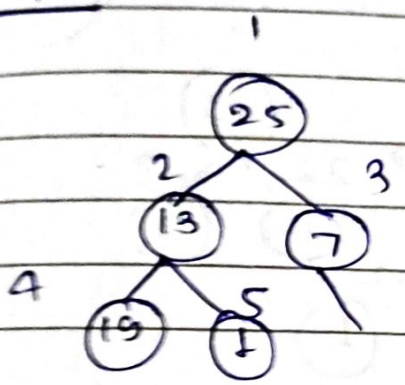


Q. For the given Array A =

1	2	3	4	5
25	13	7	19	1

Show, the steps neatly to sort the array elements in ascending order using heapsort.

Soln

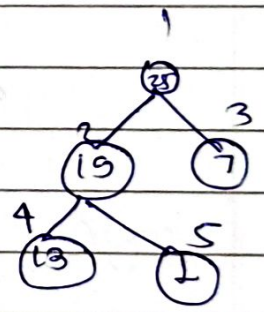


Heapsort

1. Build Heap ( )

1.1) $i = \lfloor \frac{5}{2} \rfloor = 2$

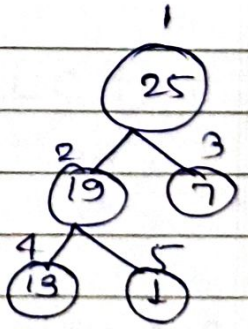
max Heapify (A, 2)



1.2. $i = \lfloor \frac{5}{2} \rfloor - 1$

$= 2 - 1 = 1$

max heapify (A, 1)



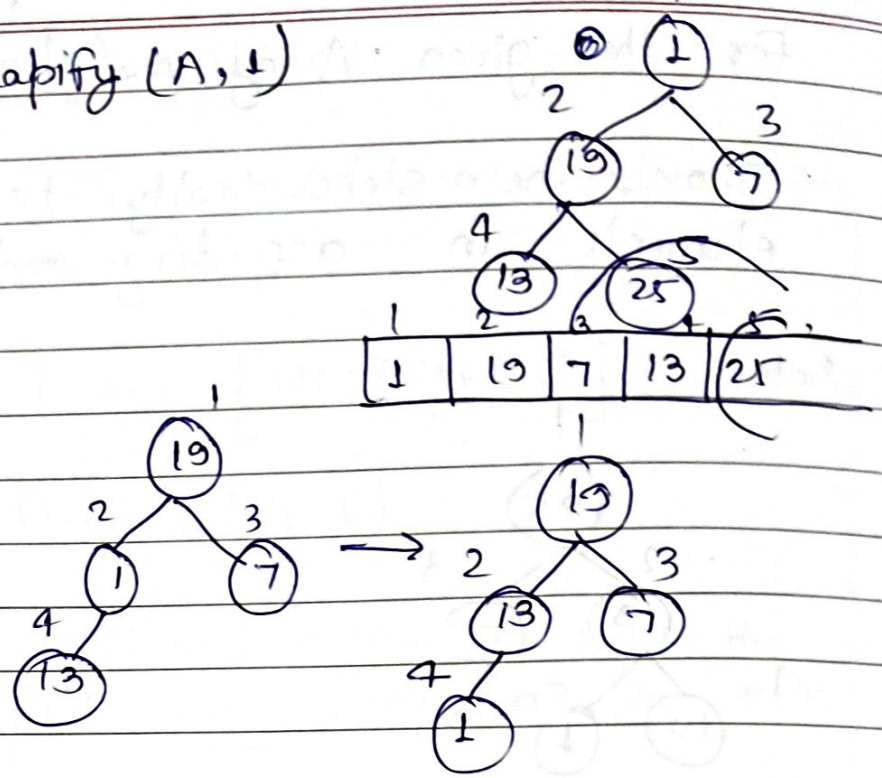
2.

2.1.) $i = 5$ heapsize(A) = 5

Swap (A[1], A[5])

heap size (A) = 4

maxHeapify (A, 1)



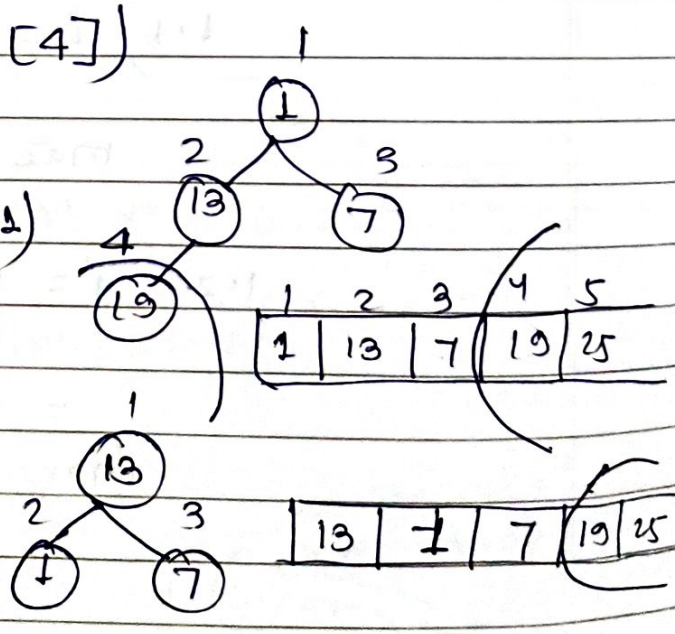
2.2

i = 4, heapsize(A) = 4

Swap (A[1], A[4])

heapsize(A) = 3

Max Heapify (A, 1)



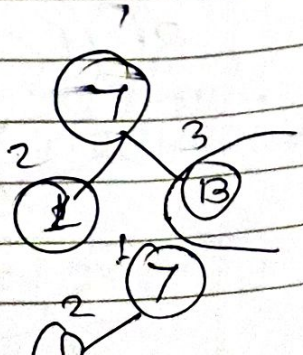
2.3

i = 3, heapsize(A) = 3

Swap (A[1], A[3])

heap size (A) = 2

max heapify (A, 1) →



Running time = $n \log_2 n$

Array:

1	7	13	19	25

2.4.

$i = 2$, heapsize(A) = 2

Swap (A[1], A[2])

heap size = 1

maxheapify (A, 1)

1	2	3	4	5
1	7	13	19	25

- Heap sort

- Greedy Algo.
  - Knapsack Problem.
  - Minimum Spanning Tree (Prim's Algo.)

Dynamic Programming:

- Longest Common Sequence (LCS)
- Quick Sort
- Introduction to NP Problem.

Greedy Technique:

- Greedy technique is a problem solving technique used for optimization problem.

- Optimization means either to maximize or minimize the value of the objective function.

- Though greedy technique is a well known optimization problem solving technique, but sometime it is unable to give the global optima.

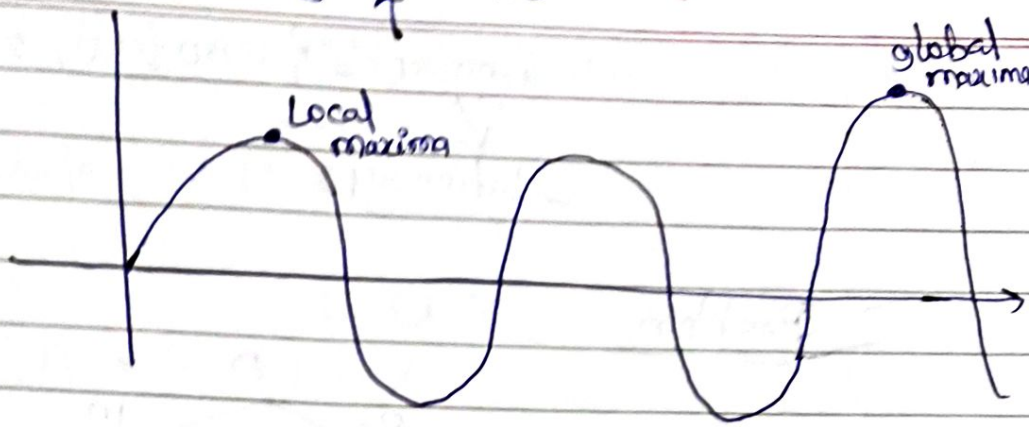
Example:

To find the minimum number of currency notes from the given denominators.

S = 500, 500 - - - -
200, 200, - - - - -
100, 100, - - - -
50, 50, - - -
20, 20, - - - - - -
10, 10, - - - - -
5, 5, - - - - -
2, 2, - - - - - -
1, 1, - - - - - - -

select RS 289/- from the given deno. so, that number of currency notes should be minimum.

to Solve this we use ~~gto~~ population based Algo.



~~Notes selected~~ value ~~so far~~.

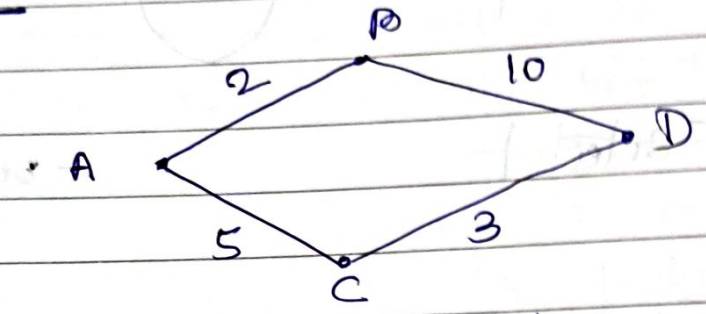
Notes selected		Value so far	Rejected/ selected
		0	
500	→ 0+500 = 500 > 289	0	Rejected.
200	→ 0+200 = 200 < 289	200	Selected.
200	→ 200+200 = 400 > 289	200	Rejected.
100	→ 200+100 = 300 > 289	200	Rejected.
50	→ 200+50 = 250 < 289	250	Selected.
50	→ 250+50 = 300 > 289	250	Rejected
20	→ 250+20 = 270 < 289	270	Selected
20	→ 270+20 = 290 > 289	270	Rejected
10	→ 270+10 = 280 < 289	280	Selected
10	→ 280+10 = 290 > 289	280	Rejected.
5	→ 280+5 = 285 < 289	285	selected
5	→ 285+5 = 290 > 289	285	Rejected
2	→ 285+2 = 287 < 289	287	Selected
2	→ 287+2 = 289	289	Selected.

gtob

∴ Answer = { 200, 50, 20, 10, 5, 2, 2 }.

↓
Solution set.

Problem



Goal
A → D.

To go from A → D. (Two paths possible).

Greedy choses this one. ✓
(i) A → B → D (2+10 = 12)
(ii) A → C → D. (5+3 = 8)

↳ here (Greedy technique fails).

↳ solve this we use Dijkstra's Algo.

↓
It is greedy technique with backward travel

---

* **Generalised Algorithum of Greedy technique :-**

Algo Greedy (A)

// Here A is the Candidate set & 'S' be the solution set.

S ← ϕ (initially)
while (! Solution (S))
{

```
x ← best Candidate element in the set A.
if (Feasible (S ∪ x))        ← Union
{
    S ← S ∪ X
    A ← A - X
}
if ( ! Solution (A))
    display ("the solution set not found").
else
    return S ;
}.
```

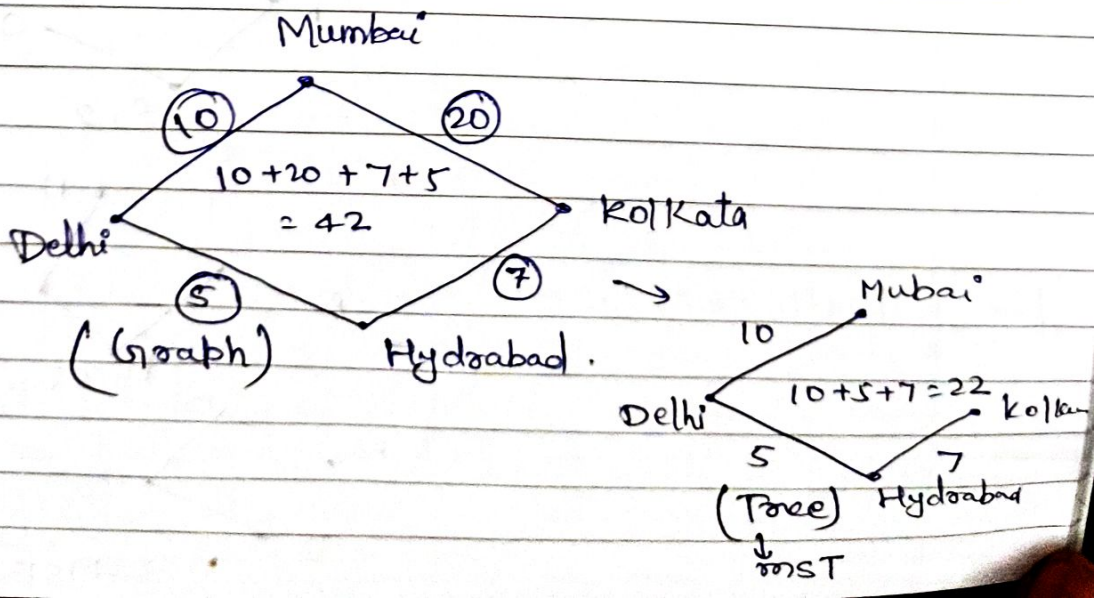
* Minimum Spanning Tree (MST)

Tree | Graph

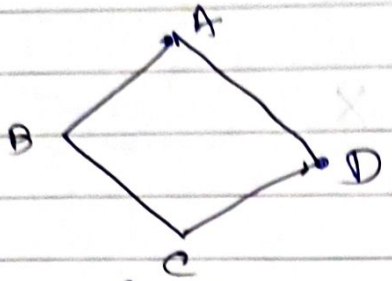
↓

is a cyclic graph

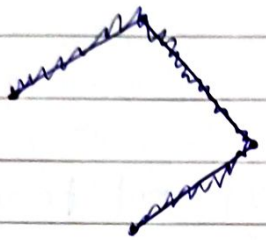
Every Tree is graph

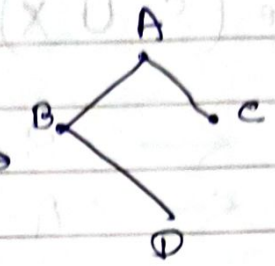


Spanning Tree is a tree generated from a graph with all the vertices of graph

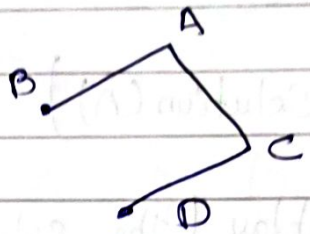


Graph.

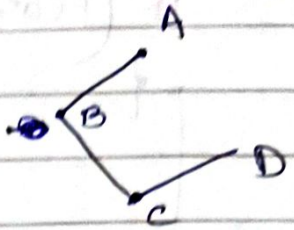




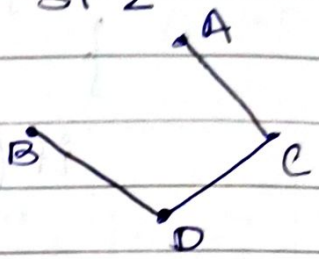
ST 1



ST 3

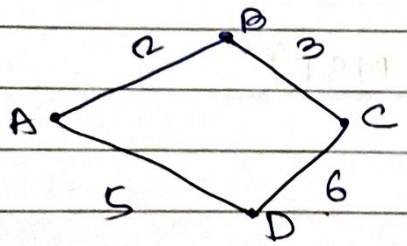


ST 2

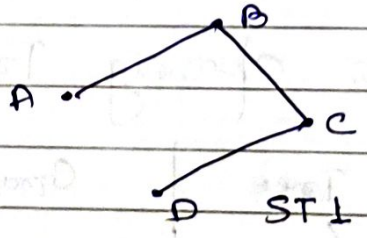


ST4.

- Minimum Spanning Tree : MST is spanning tree with minimum ~~value~~ total value.

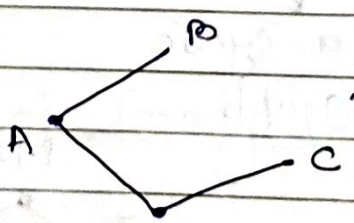


(Graph)



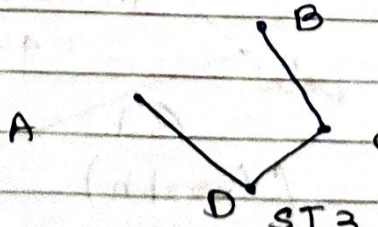
= 2+3+6 = 11

ST 1



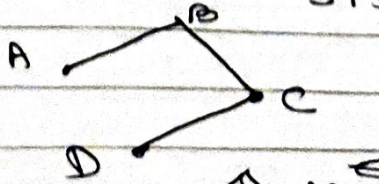
2+5+6 = 13

ST 2



5+6+3 = 14

ST 3



2+3+5 = 10

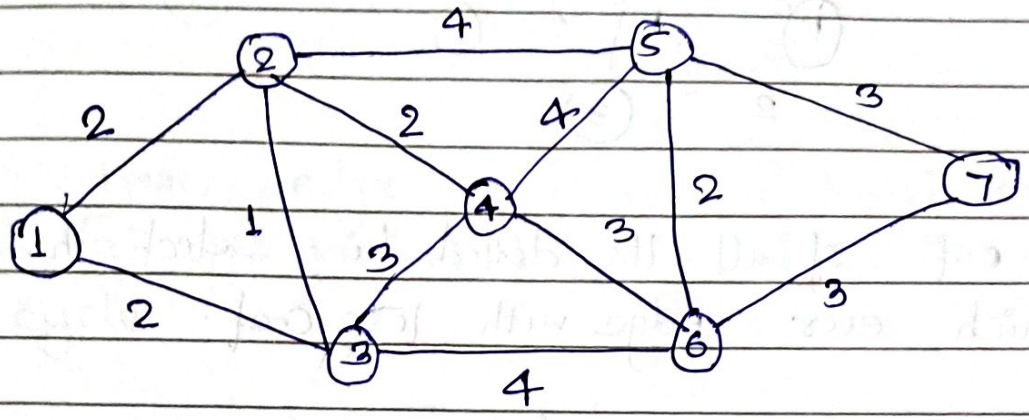
← minimum.

↖ MST

Minimum Spanning Tree:

* Prim's Algorithum:

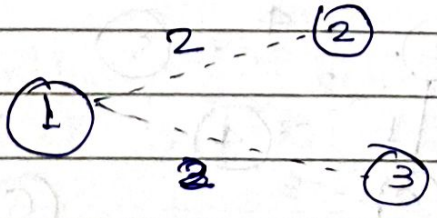
Using Prim's algorithum find the MST of the following graph.



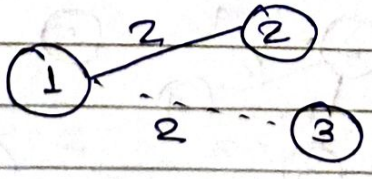
Ans: Step-1

Arbitrarily select one vertex, let vertex '1' is selected.

Draw the dotted lines from (1) to all the vertices.

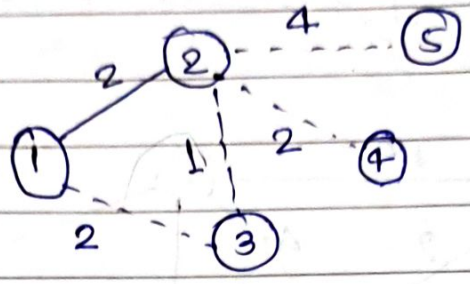


Select, the vertices which over with the less cost.

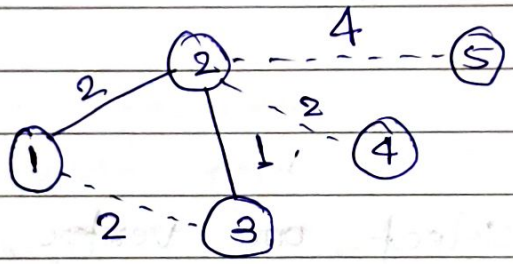


Step : 2

Since, recently '2' is selected, draw dotted line of all adjacent of (2).

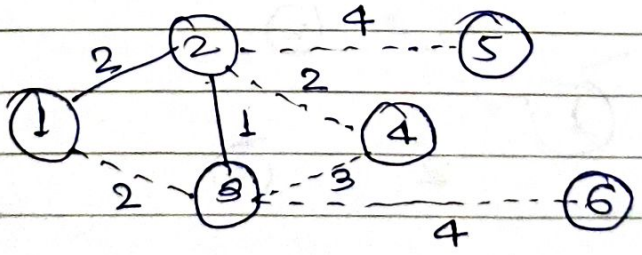


- out of all the dotted line select the vertex which ever edge with less cost.

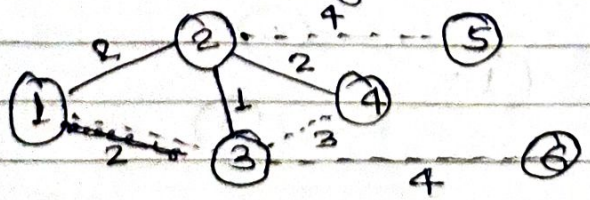


Step : 3

∵ recently selected vertex is (3) Draw dotted lines to the adjacent vertices of (3)

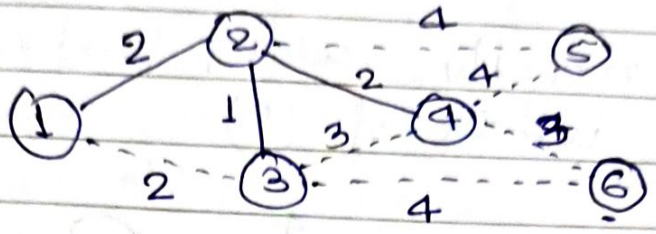


select, the ~~vertex~~ edge to (4) with cost 2.



Step 4:

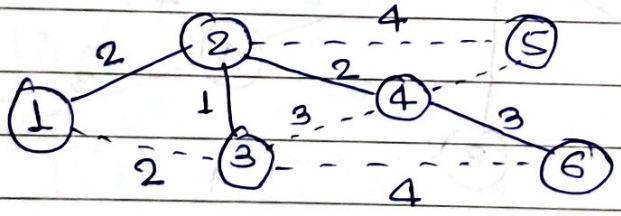
∵ recently selected vertex is (4) Draw dotted lines to the adjacent vertices of (4)



Here, edge $\overline{13}$ is the lowest one with cost (2), it can't be selected as it generates a cycle.

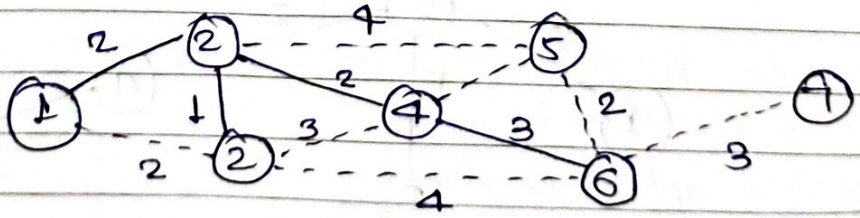
edge $\overline{34}$ with cost 3. not possible (cycle created).

Select $\overline{46}$ with cost 3.

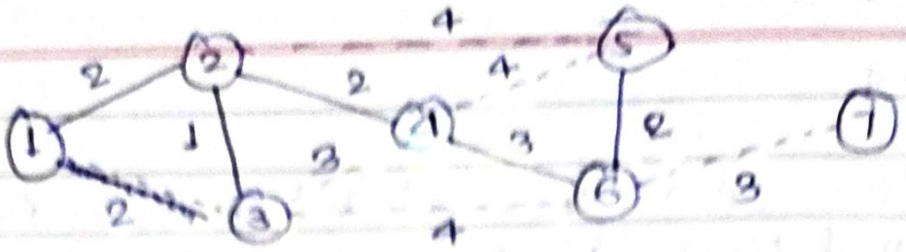


Step 5:

∵ recently selected vertex is (6) Draw dotted lines to the adjacent vertices of (6)

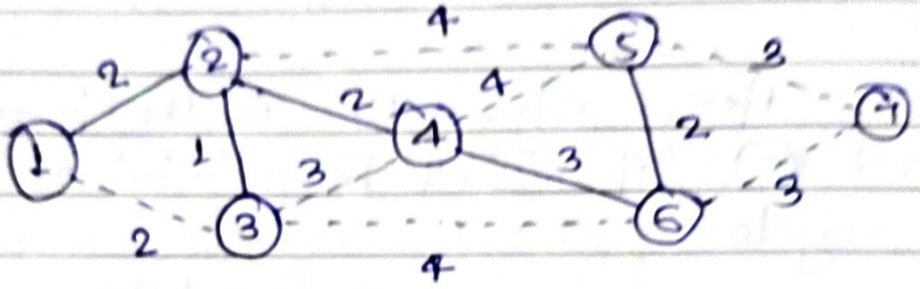


Here edge $\overline{65}$ is selected with cost 2.

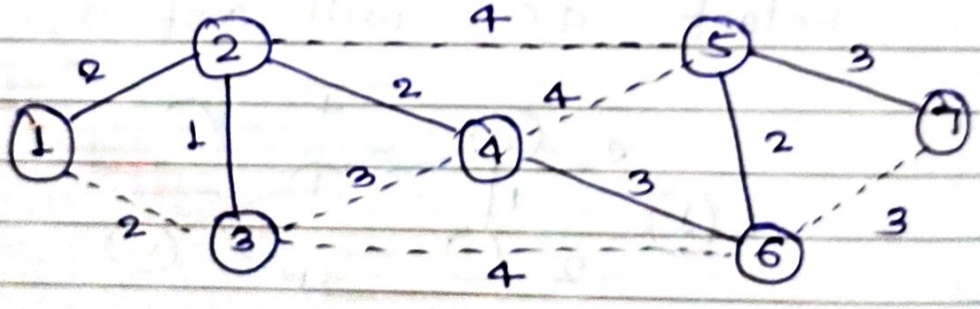


Step 6:

∵ we recently added vertex is (5) draw adjacent vertices of (5)

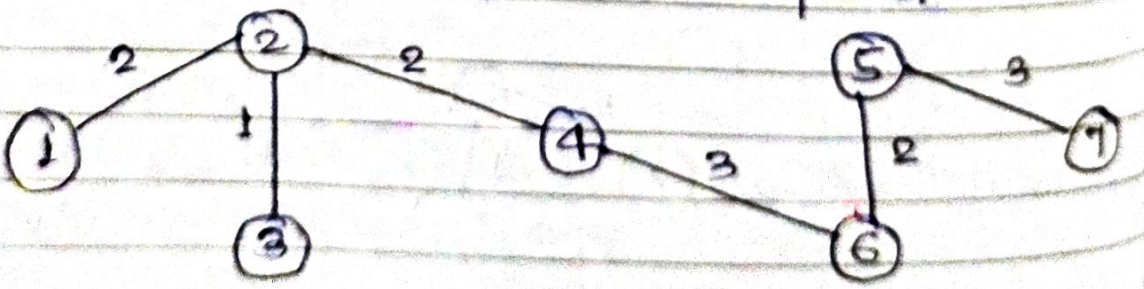


select edge $\overline{57}$ ~~free~~ with ~~value~~ 3.
Cost



Step - 7

Since all the vertex of the given graph is included in the tree, the required MST is:



Cost of MST = 2+1+2+3+2+3

= 13.

* Dynamic Programming :-

Dynamic Programming is a problem solving technique is also applied for the optimization problems. In optimization problem, there may be many possible solution. Each solution has a vertex, and we wish to find a solution with the optimal (minimum or maximum) value. We call such a solution an optimal solution to the problem.

The following are the steps to solve a DP problem:-

1. Characterize the structure of an optimal sol^n
2. Recursively define the value of an optimal solution.
3. Compute the value of an optimal sol^n in a bottom up fashion.
4. Construct an optimal sol^n from the computed information.

→ There are two methods with which DP problems are solved :

1. Memorization. (Top down approach, Recursive but less efficient)
2. Tabulation (Bottom up approach, efficient).

To find the factorial of a number :

Memo-ization

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

It is memorizing the value not storing it.

0	1	2	3	4	5
1	1	2	6	24	120

Dynamic Programming :-

— Memorization technique :-

Fibbonacci Series :-

```
Algo Fibonaci(n)
{
    if (n==0)
        return 0
    if (n==1)
        return 1;
    else
        return (fibonaci(n-1) + fibonacci(n-2))
}
```

F(5)
= F(4) + F(3)
= f(3) + f(2) + f(2) + f(1))
= f(2) + f(1) + f(1) + f(0) + f(1) + F(0) + 1
= f(1) + f(0) + 1 + 1 0 + 1 + 0 + 1
= 1 + 0 + 1 + 1 + 1 + 1
= 5.

— Tabulization method :-

```
Algo fibonaci(n)
{
    Create an array T[n]
        T[0] = 0
        T[1] = 1
        for(i= 2; i<n ; i++)
```

```
    T[i] = T[i-1] + T[i-2]
    return T[n];
}.
```

$T[2] = T[1] + T[0]$

$T[3] = T[2] + T[1]$

* LCM (Longest Common Subsequence) :

Let $X = \langle x_1, x_2, x_3, \ldots x_m \rangle$ be a string and $Y = \langle y_1, y_2, y_3, \ldots, y_n \rangle$ be another "

where m and n may or may not be equal to 'n'.

find a string $Z = \langle z_1, z_2, \ldots z_k \rangle$ where Z is the longest Common subsequence (LCS) of X & Y.

Ex:

Let $X = \langle A, B, C, B, D, A, B \rangle$

$Y = \langle B, D, C, A, B, A \rangle$

1. $Z = \langle B, C, A \rangle$ is said to be Common subsequence of X and Y, but it is not the longest one. so, here Z is said to be the Common sequence of X and Y but not the LCS of X & Y.

2. $Z = \langle B, C, B, A \rangle$ is Common subsequence of length 4 also known as the LCS of X and Y. Because, we Can't able

to find any subsequence of length 5 or more.

S. $Z = \langle B, D, A, B \rangle$ is also another LCS (x,y)

Problem:

Let $X = \langle A, B, C, B, D, A, B \rangle$ len(x) = 7

$Y = \langle B, D, C, A, B, A \rangle$ len(y) = 6

find LCS (x, y).

↑ ↖ ←

x_i		y_j	0	1 B	2 D	3 C	4 A	5 B	6 A
			0	0	0	0	0	0	0
1	A		0	↑0	↑0	↑0	↖1	↑1	↖1
2	B		0	↖1	←1	←1	↑1	↖2	←2
3	C		0	↑1	↑1	↖2	←2	↑2	↑2
4	B		0	↖1	↑1	↑2	↑2	↖3	←3
5	D		0	↑1	↖2	↑2	↑2	↑3	↑3
6	A		0	↑1	↑2	↑2	↑3	↑3	↖4
7	B		0	↖1	↑2	↑2	↑3	↖4	↑4

c(8,7)

8×7

c[i,j] = length of LCS($x_i - y_j$)

c[0,0] = x = ϕ

y = ϕ

length (LCS (x, y)) = 0

From, the path we will only select the boxes which have diagonal arrow. (↖)

28-08-24.

c[i,j] { x = 'A' / y = 'B' }

∴ LCS = Z = <BCBA>

len(Z) = 4.

Algo LCS (x, y, m, n)

// Here x, and y are the given strings and m and n are the length of x and y, respectively.

{ create two matrices c[m][n] (numeric matrix) and b[m][n] (character matrix) ←, ↑, ↖

```
for(i=0 to m)
  do
    c[i][0] = 0
    // the first column is zero(0)

for( i ← 0 to n)
  do
    c[0][i] = 0
    // the first row is 0.

for i ← 1 to m
  do
    for j ← 1 to n
    do
      if (x[i] == y[j])
      { c[i][j] = c[i-1][j-1] + 1
        b[i][j] = '↖'
      }
```

```
else if (c[i-1][j] ≥ c[i][j-1])
    {
        c[i][j] = c[i-1][j]
        b[i][j] = '↑'
    }
else
    {
        c[i][j] = c[i][j-1]
        b[i][j] = '←'
    }
    return c and b
}
```

* Algo PrintLCS (b, x, i, j)

// This algorithum print the LCS and this algo is called after the LCS algorithum.

{

1. if (i == 0 // (or) j == 0)
2. return.
3. if b[i][j] = '↖'.
4. print_LCS (b, x, i-1, j-1)
5. Display x[i]
6. else if
   b[i][j] = '↑'
7. Print_LCS (b, x, i-1, j)

PL := Print

8. else
9. Print_LCS(b, X, i, j-1)
}.

Tracing Algo:

0	1	2	3	4	5	6
A	B	C	B	D	A	B
			X			

PL(b, x, 7, 6)
1. b[7][6] = '↑'
PL(b, X, i-1, j)
PL(b, X, 6, 6)

2. b[6][6] = '↖'
PL(b, x, 5, 5)
display x[6]

3. b[5][5] = '↑'
PL(b, x, 4, 5)

4: b[4][5] = '↖'
PL(b, x, 3, 4)
display (x[4])

5: b[3][4] = '←'
PL(b, X, 3, 3)

6: b[3][3] = '↖'
PL(b, X, 2, 2)
display (x[3])

7: b[2][2] = '←'
PL(b, x, 2, 1)

8: b[2][1] = ↖

PL (b, x, 1, 0)

display (x[2])

9: Since (j == 0)

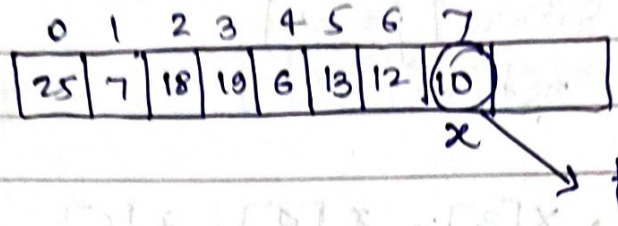
return.

x[2], x[3], x[4], x[6]

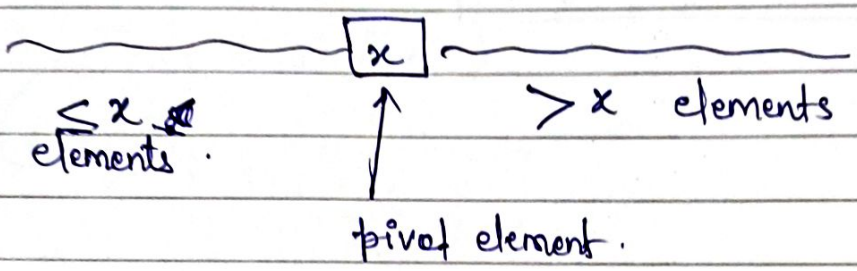
B, C, B, A

Quick Sort:

Divide & Conqur rule



Iteration-1



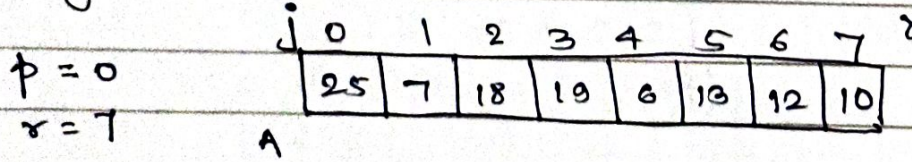
- Partition Algo:

1. Algo partition (A, P, r)

   // A is the array.
   P is the first index of array.
   q is the last index of array.

2. Algo Quicksort (A, P, r)

1. Algo Partition



p = 0
r = 7

Step : 1

$x = A[r] - A[7] = 10$

$j = 0$

$i = j - 1$

Since $(A[0] > x)$ (10 > 10)

no operation ;

Step 2 :

$j = 1$

Since $(A[1] \leq x)$ (7 ≤ 10)

{

$i = i + 1 = -1 + 1 = 0$

swap $(A[i], A[j])$ (0, 1)

0	1	2	3	4	5	6	7
7	25	18	19	6	13	12	10

Step 3 :

$j = 2$

Since $(A[2] > x)$ (18 > 10)

no operation.

Setep 4 :

$j = 3$

Since $(A[3] > x)$ (19 > 10)

no operation.

Step 5 :

$j = 4$.

Since (A[4] ≤ x)
(6) (10)

{

i = i+1

0 = 0+1 = 1

Swap (A[i], A[j])
1 4

}

0	1	2	3	4	5	6	7
7	6	18	19	25	13	12	10

Step-6

j = 5

Since (A[5] > x)
(13) (10)

no operation.

Step-7:

j = 6

Since (A[6] > x)
(12) (10)

no operation.

Step 8:

i = 1 → i = i+1 = 2

Swap (A[i+1] , A[r])
2 7

7	6	10	19	25	13	12	18

> x

x ≥

pivot element.

- Algo Partition (A, P, r)

// This algo partition the array A [P.....r] into two subproblems and placed x = A[r] into the correct position, so, that LHS (x) ≤ x and RHS (x) > x , Also, it returns the index of 'i' where x being placed.

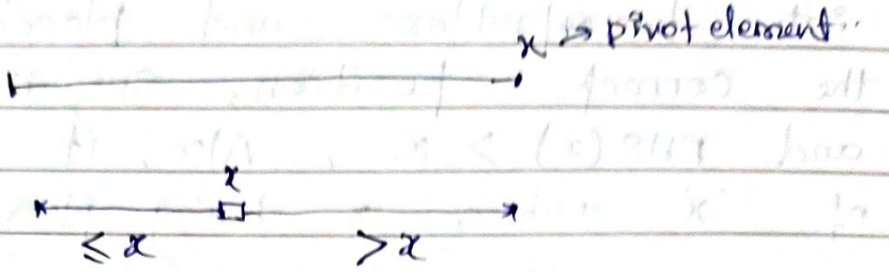
```
{
    x = A[r]      // pivot element

    i = p-1
    for j = 0 to (r-1)
      do
          if (A[j] ≤ x)
          {
              i = i+1
              swap (A[i], A[j])
          }
    swap (A[i+1], A[r])
      return (i+1)
    }.
```

- Algo Quicksort (A, P, r)

```
{
    if (p<r)
    {
        q = partition (A, P, r)
        Quicksort (A, P, q-1)
        Quicksort (A, q+1, r)
    }
}.
```

- Running time of Quick Sort:

Partition



→ Generalize recurrance of any D&C Problem.

$$T(n) = aT\left(\frac{n}{b}\right) + D(n) + C(n)$$

$$\boxed{T(n) = 2T\left(\frac{n}{2}\right) + n} + 0$$

↳ for Quick Sort

by Solving we get

$$\boxed{T(n) = \Theta(n \log_2 n)}$$

↳ Best Case Running of Quicksort (if the array is unsorted)

Case * If array is already sorted:

1	2	3	4	5	6	7
2	4	6	8	10	12	15

(n-1) size — pivot element

$$\begin{cases} T(n) = T(n-1) + n \\ T(1) = 0 \end{cases}$$

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

$= T(n-2) + (n-1) + n$

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

⋮

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

$= T(1) + 2 + 3 + \cdots + n$

$= 0 + 2 + \cdots + n$

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

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

$= \boxed{\Theta(n^2)}$ → if array is sorted. (worst case)

$T(n-1) = T(n-2) + T(n-1)$

In Quick Sort:

- Worst case : If the array is already sorted.
  → $\Theta(n^2)$.

- Best case : If the array is unsorted
  ↳ $\Theta(n \log_2 n)$

Prim's Algorithum:

1. [Initialize a tree T]
Let 'U' be the arbitarary vertex of 'G'
and 'T' = 'U'.

2. [The tree is updated]
Let 'e' be the edge of minimum weight joining a vertex of T and vertex is not in T, then T = T + e

3. [This step determines wheather a MST has been constructed]
If 'T' contains all the vertices of the given graph, display T, else goto step 2.

Classification of Problem:

P, NP, NP Complete, NP hard.

- **P Problems:**

↓

polynomial (n^k)

P problems are such category of problem which can be solved in polynomial time by deterministic machine.
(sequential processor) slow

- **NP Problems:**

↓

Non polynomial

NP problems are such category of problem which can be solved in polynomial time by non-deterministic machine.
(parallel processing) Fast.

or.

NP problems are ~~~~ verifiable in polynomial time by a deterministic machine.

(verifiable or decision problem should return yes/no in answer) (only)

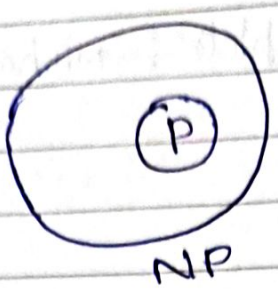
ex: Weather in the class there strength is 50?

yes/no.

How many students are in class (x)

Date
Page

→ relationship b/w P and NP.



- P is subset of NP
- so, all the P problems are NP Problems.

- NP Complete :

A problem 'L' is said to be NP Complete if there are two conditions.

if (1) $L \in NP$

(2) $L \leq_p L'$

where $L' \in NPC$

$\leq_p$ polynomial time reducable

→ Polynomial time reducable ($\leq_p$)

$L_1 \leq_p L_2$ if there exist a polynomial time algorithum which the problem L_1 can be converted to L_2.

ex:

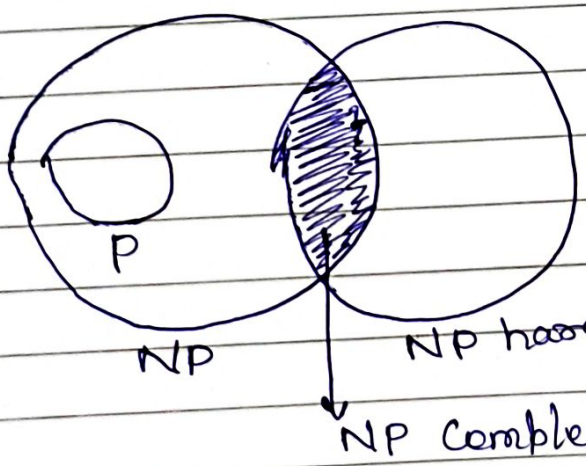
0/1 Knapsack Problem $\leq_p$ Clique Problem

- **NP hard**

A problem 'L' is said to be NP hard if

$$L \leq_P L' \text{ where } L' \in NPC$$

→ Relationship b/w all problems;



All P problem are NP ✓

All NPC problem are NP hard. ✓

Some NP hard problems are NPC. ✓

All NP Hard problems are NPC X.

All NPC problem are NP. ✓