⇒ Selection Procedure :-

I/p :- An array of n-integers , integer K

O/p :- Find K^{th} smallest element.

eg:-

A

50	20	30	70	60	19	17	80	90	11	36	99
1	2	3	4	5	6	7	8	9	10	11	12

K =

Apply partition :

k < m

20	30	19	17	11	36	50	70	60	80	90	99
1	2	3	4	5	6	7	8	9	10	11	12

k > m

K=m

m = 7 if(m = K)

return

Let K = 10

So, m = 70 60 80 90 99

↑ p

= (60) 70 (80 90 99)

8 9 10 11 12

K > m

move Right

m = 80 (90 99)
10 11 12

- m = 10 as k = m so return

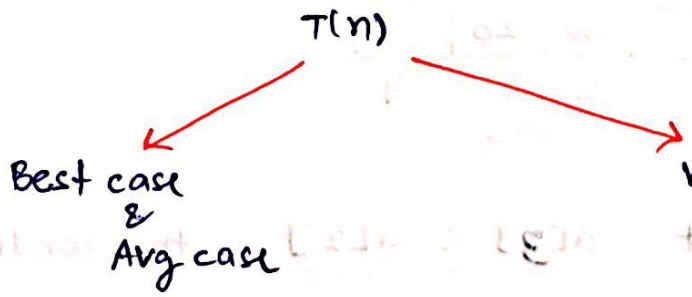
Algorithm:-

```
SP ( a, p, q, k)                                   —— T(n)
{
   if ( p==q)
      return a[p];
   else
   {
      m = partition ( a,p,q)                       —— O(n)
      if ( m==k)
         return ( a[K]);                           —— O(1)
      else if ( K < m)
         return SP( a, p, m-1, k);                 —— T( m-p)
      else
         return sp( a, m+1, q, k);                 —— T( q-m)
   }
}
```

— It is combination of Quicksort & Binary search.

Recurrence relation for time complexity :-

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ n + T(n-p) \text{ or } n + T(q-m) & \text{if } n > 1 \end{cases}$$



$T(n) = n + T(n/2)$ — Best partition

$= O(n)$

$T(n) = n + T(n-1)$ — Worst partition

$= O(n^2)$

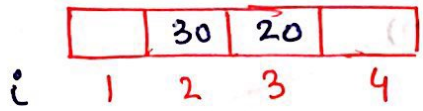
- Tail recursion is present, so we can convert into equivalent non-recursive program with ~~space~~ stack space $O(1)$, but time complexity is same.
- Here tail recursion is present in all cases (best, avg, worst).

⇒ Counting no. of Inversions :-

I/p :- An array of n elements

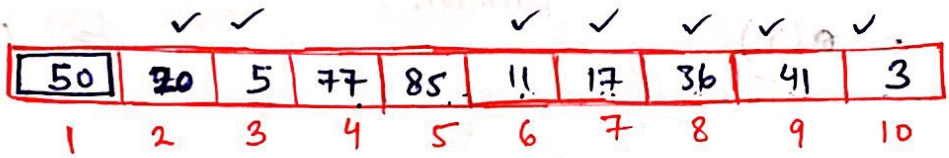
O/p :- Count Inversions

Inversion :



3>2 but a[3] < a[2], this condition is known as inversion.

eg:-



Inversion for 50 → 7

" 20 → 4

" 5 → 1

" 77 → 5

" 85 → 5

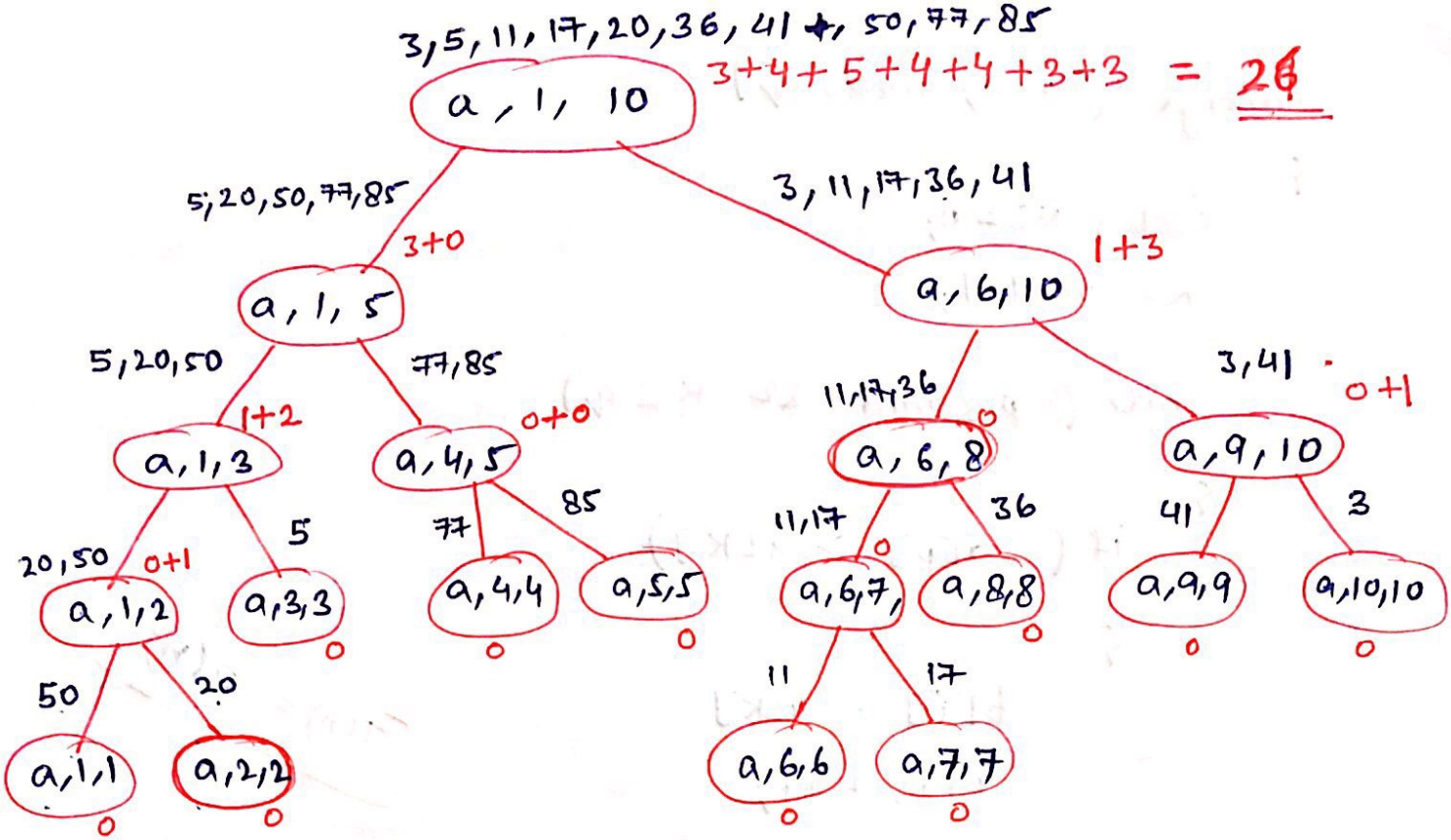
" 11 → 1

" 17 → 1

" 36 → 1

" 41 → 1

" 3 → 0



Algo:-

```
countInversion ( a, p, q )  ——— T(n)
{
    if ( p==q)
    {
        return 0;  ——— O(1)
    }
    else
    {
        mid = ⌊(p+q)/2⌋;  ——— O(1)
        NI1=countInversion ( a, p, mid-1);  ——— T(n/2)
        NI2=countInversion ( a, mid+1, q);  ——— T(n/2)
        NI += merge ( a, p, mid, q);  ——— NI = NI1+NI2;  ——— O(1)
                                          ——— O(n)
        return NI;
    }
}
```

```
merge (a, p, mid, q)
{
    i=1 ; NI=0;
    K= mid+1;

   while ( p ≤ mid && K ≤ q)
   {
      if( a[p] > a[K])
      {
           b[i] = a[K]
           i++; k++;
           NI += mid-p+1;
      }
      else
      {
           b[i] = a[p]
           i++; p++;
      }
   }
   copy remaining elements to B[]
}
```

$T(n) = O(n)$

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

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

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

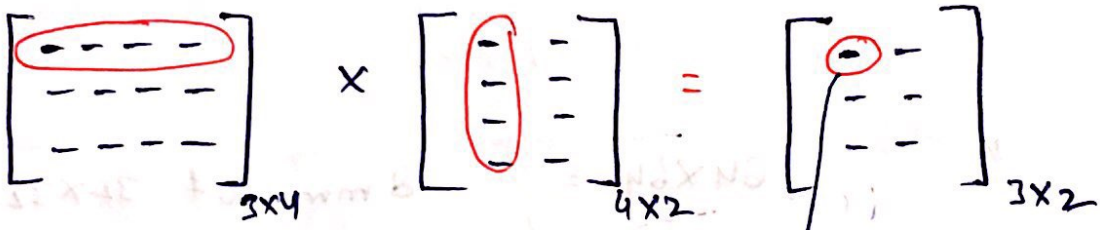
⇒ Strassen's Matrix Multiplication :-

Matrix - multiplication without divide & conquer :-

$[A] + [B] = [C]$ Sum $\quad T(n) = O(n^2)$

$[A] \times [B] = [C]$ Mul $\quad T(n) = O(n^3)$

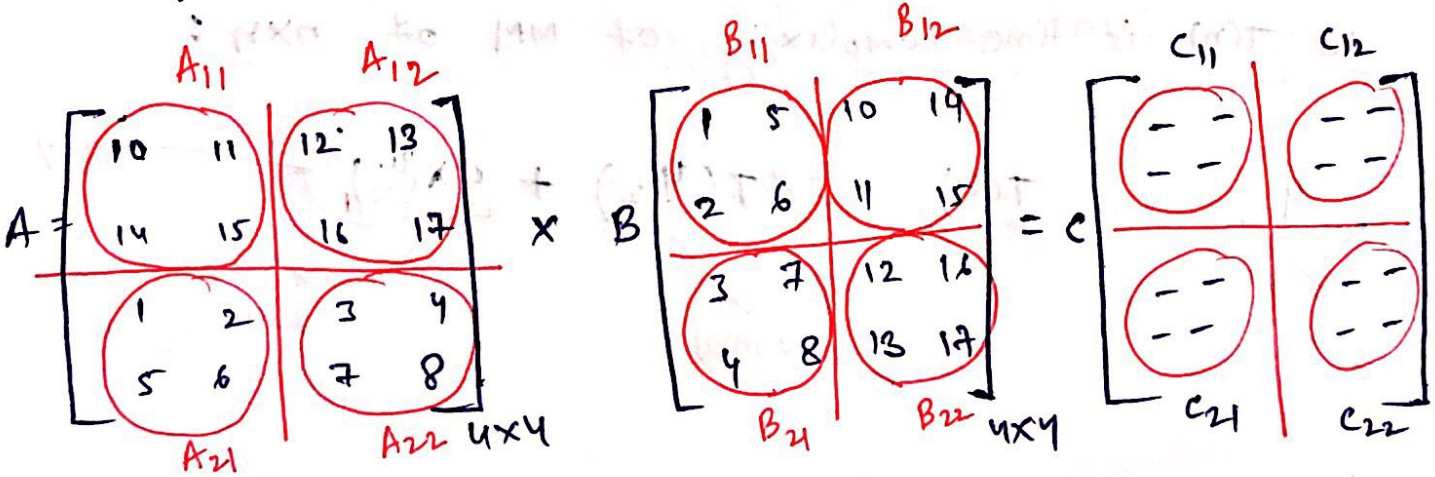
$$[A]_{3\times4} \times [B]_{4\times2} = [C]_{3\times2}$$



$$C_{11} = A_{11} \cdot B_{11} + A_{12} B_{21} + A_{13} B_{31} + A_{14} B_{41}$$

Matrix - multiplication with D & C :-

(i) Divide matrices by 2, and assume each group as single elements.



$$
\begin{aligned}
C_{11} &= A_{11}B_{11} + A_{12}B_{21} \\
C_{12} &= A_{11}B_{12} + A_{12}B_{22} \\
C_{13} &= A_{21}B_{11} + A_{22}B_{21} \\
C_{14} &= A_{21}B_{12} + A_{22}B_{22}
\end{aligned}
$$

8 multiplication (mm)
+
4 additions (mA)
+

1 matrix-mul (mm) of 4×4 = 8 mm of 2×2 + 4mA of 2×2

64×64 = 8 mm of 32×32 + 4mA of 32×32

n×n = 8mm of $n/2 \times n/2$ + 4mA of $n/2 \times n/2$

Let T(n) is time-complexity of MM of n×n :

n=4

$$T(4) = \underbrace{8T(4/2)}_{8mm} + \underbrace{4\times(4/2)^2}_{\text{matrix addition time}} \quad \rightarrow O(n^2)$$

So,

$$T(n) = \begin{cases} O(1) & \text{if } n \le 2 \\ 8T(n/2) + 4(n/2)^2 & \text{if } n > 2 \end{cases}$$

$T(n) = 8T(n/2) + n^2$

$T(n) = \theta(n^3)$

By applying master theorem

According to Strassen's :-

$T(n) = 7T(n/2) + 18(n/2)^2$

(mul ↓, addition ↑ increased)

$T(n) = 7T(n/2) + n^2$

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

Q. What is time complexity?

```
A(n)
{ if(n<=1)
    return 1;
  else
    return (n^2+n+1)
}
```

$T(n) = O(1)$

constant

no, recursion & no loop.

Q. Find time complexity $T(n)$?

```
A(n)
{ if(n ≤ 1)
    return (n²+n+1)          ——— O(1)
  else
    return ( A(n/2) + A(n/2) + n)
}
```

$$T(n) = \begin{cases} O(1) & n \le 1 \\ 2T(n/2) + c & n > 1 \end{cases}$$

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

$$= 2T(1) + (c + 2c + 4c + \cdots)$$

$$= 2O(1) + c(1 + 2 + 4 + \cdots)$$

K terms

$$= \frac{1(1 \cdot 2^{\log_2 n} - 1)}{2-1} \qquad K = \log n$$

$$T(n) = O(n)$$

Q. Find $T(n)$:

```
A(n)
{
  if(n≤1)
  {
    return 1
  }
  else
  {
    return ( A(√n) )
  }
}
```

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

$$= T(n^{1/2}) + c$$

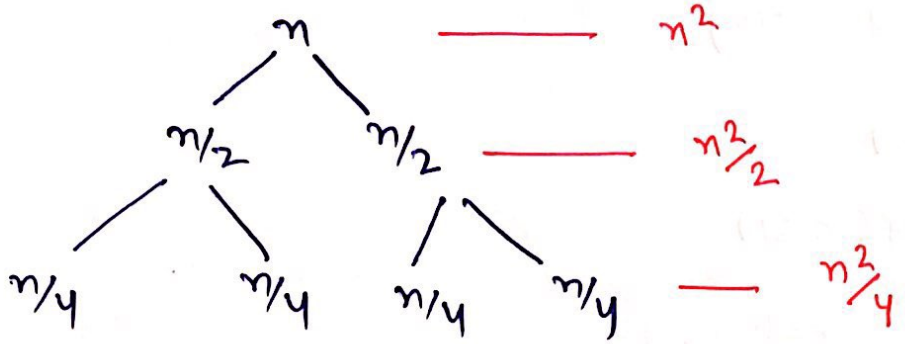
$T(n) = O(\log(\log n))$

Q. Find $T(n)$:

```
A(n)
{
  if(n ≤1)
    return n
  else
  {
    return ( 5A(n/2) + 3A(n/2) + Matrix Addition)
  }
}
```

$$T(n) = 2^8 T(n/2) + O(n^2)$$

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



$$T(n) = n^2 \left(1 + 1/2 + 1/4 + \cdots \right)^{\log n}$$

$\frac{a(1-r^n)}{1-r}$ ✓

$$T(n) = n^2 \left(\frac{1\left(1 - 1/2^{\log_2 n}\right)}{1 - 1/2} \right) = 2n^2 \left(1 - 1/n \right)$$

$$\boxed{T(n) = \Theta(n^2)}$$

Q.

```
A(n)
{
  if(n ≤ 1)
    return 1;
  else
    return (n * A(n-1))
}
```

Find recurrence relation for Time complexity, value & multiplication.

(i)

$$T(n) = \begin{cases} O(1) & n \le 1 \\ T(n-1) + c & n > 1 \end{cases}$$

(ii) $$V(n) = \begin{cases} 1 & n \le 1 \\ n * V(n-1) & n > 1 \end{cases}$$

(iii) $$M(n) = \begin{cases} 0 & n \le 1 \\ M(n-1) + 1 & n > 1 \end{cases}$$

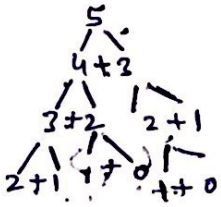
Q.
```
A(n)
{ if (n≤1)
      return n
  else
      return ( A(n-1) + A(n-2))
}
```

Find recurrence rel for Time, value, addition.

$$T(n) = \begin{cases} O(1) & n \le 1 \\ T(n-1) + T(n-2) + c & n > 1 \end{cases}$$

$$V(n) \begin{cases} n & n \le 1 \\ V(n-1) + V(n-2) & n > 1 \end{cases}$$

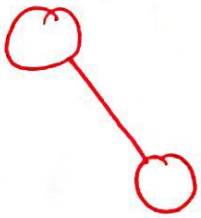
$$A(n) \begin{cases} 0 & n \le 1 \\ A(n-1) + A(n-2) + 1 & n > 1 \end{cases}$$



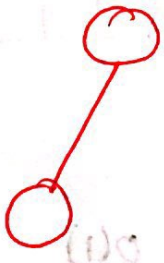
Binary Tree

⇒ [redacted]:-

Almost complete Binary tree:-



BT ✓
ACBT ✗

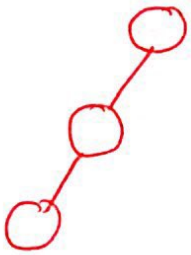


BT ✓
ACBT ✓

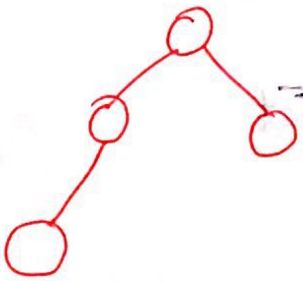
In the given binary tree at every node -

(i) After completion of left only go to right.

(ii) After completion of one level fully then go to next level.



ACBT X



ACBT ✓

- In a complete Binary tree of K levels:

$$\text{Total nodes} = 2^K - 1 \quad \text{(min or max)}$$

- In a complete / Almost complete Binary tree of n-nodes

$$\text{No. of Leaf nodes} = \left\lceil \frac{n}{2} \right\rceil$$

$$\text{Internal nodes} = \left\lfloor \frac{n}{2} \right\rfloor$$

- If complete BT contain n nodes and k-levels

$$n = 2^k - 1$$

$$K = \log_2(n+1)$$ No. of levels

Height = Level - 1 = K - 1

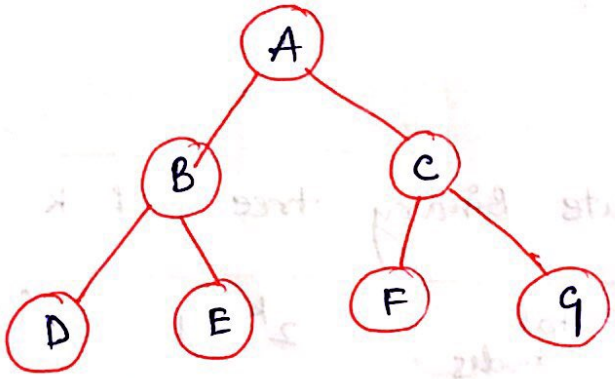
Representation of Binary trees :-

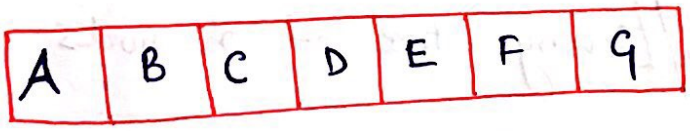
↳ Array

↳ Linked List

(i) Array Representation :-

eg

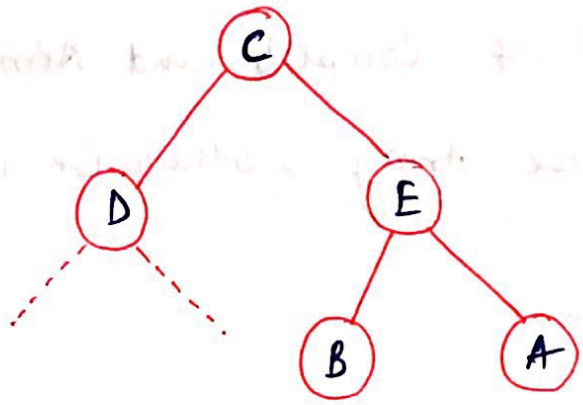


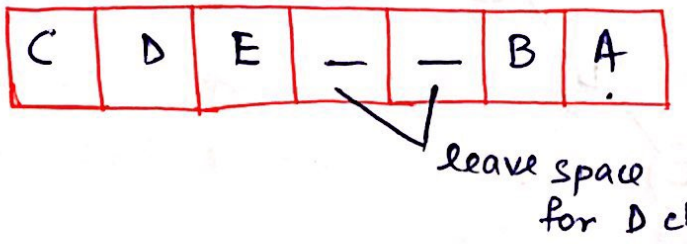


If node is stored in I^{th} place in array -

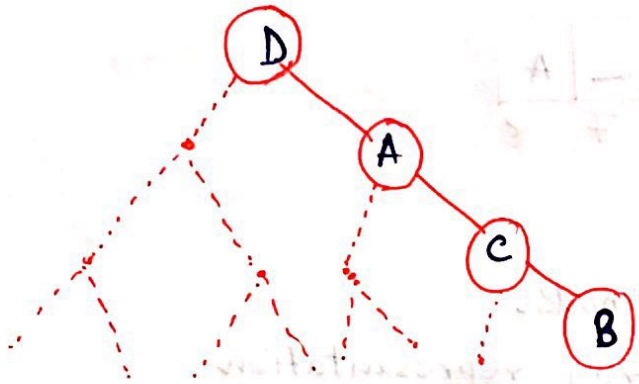
- parent (i) = $\lfloor i/2 \rfloor$
- Left - child (i) = $2i$
- Right - child (i) = $2i+1$

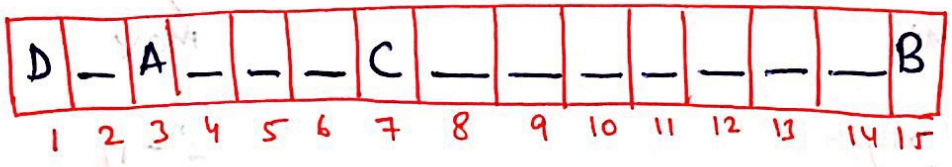
eg:-



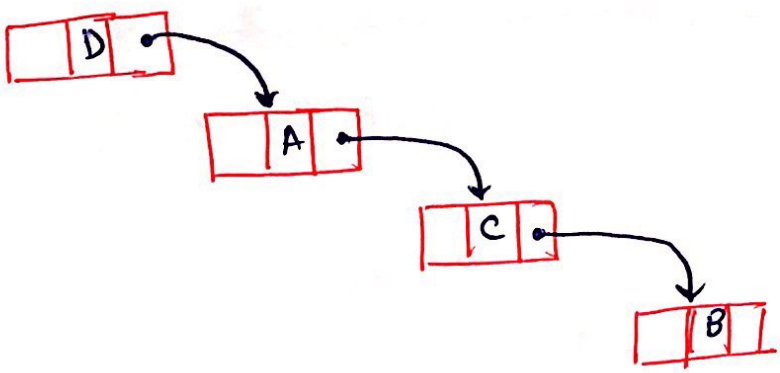


eg:-





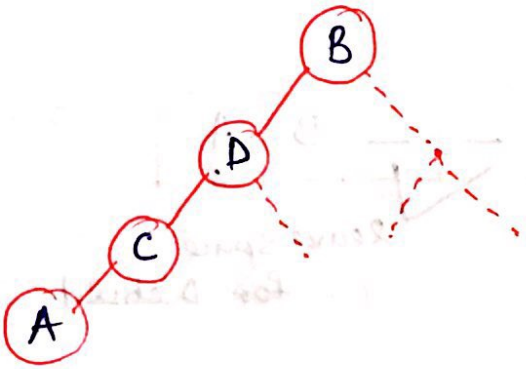
Space
= 15

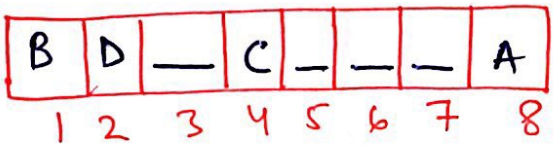


Space = 3X4
= 12

NOTE:- In case of Complete and Almost Complete Binary tree use Array, otherwise use Linked List.

eg:-





BT – n nodes

Array representation

Min (n)

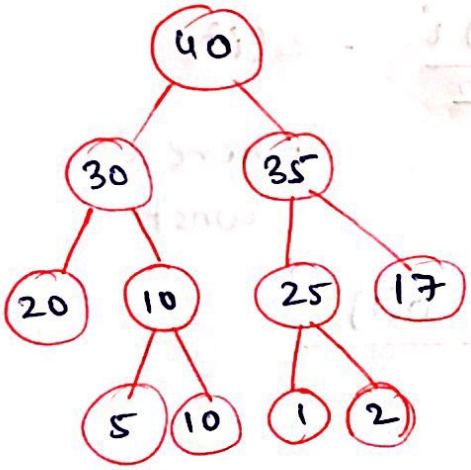
Max $(2^n - 1)$

⇒ Heap - tree :-

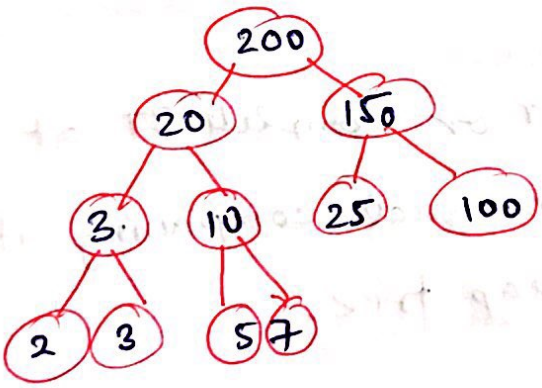
- Max Heap tree
- Min heap tree

= Max heap tree :-

In the given almost complete BT or complete BT at every node Root is maximum or equal comparing its children is known as max heap tree.



It is not Max heap tree as it is not ACBT. or CBT.



ACBT, so it is max heap tree

200	20	150	3	10	25	100	2	3	5	7

Max Heap:-

1st max — 1 — 0 — O(1)

2nd max — 2 — 1 — O(1)

3rd max — 3 — 2 — O(1)

⋮ ⋮

10th max — 10 — 9 — O(1)

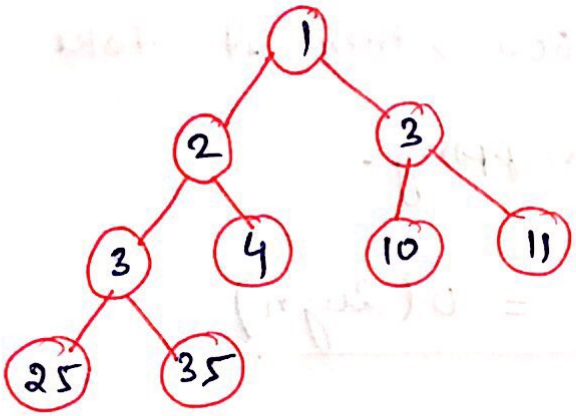
So, for i^{th} max — $\frac{(i-1)\,i}{2} = O(i^2)$

where i is const

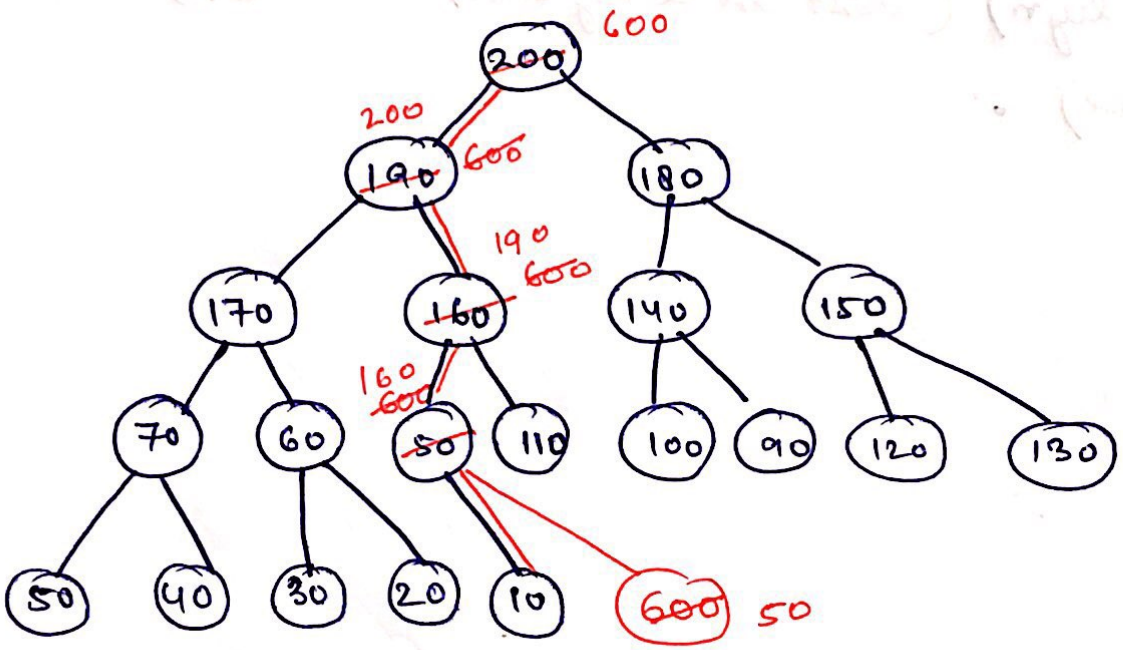
So, $T(n) = O(1)$

Min Heap tree:-

In the given almost complete BT or complete BT at every node Root is minimum or equal compairing its children is known as min heap tree.



Insertion in Max heap :- (Max heapify)



n = arraysize (21)

m = last element position (20)

m = m+1

= 21

a[m] = x

so, if we insert 600, then it take $\log n$ comparison and swapping.

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

NOTE:- Inserting a element into Minheap or Maxheap which already contain n elements will take $O(\log n)$ (worst case & Avg case) and $O(1)$ (Best case).