Algorithms & Data Structure

Kiran Waghmare

-Analysis of Algorithms

Algorithm
- -Design
- -Domain Knowledge
- -Language
- -Hardware, OS
- -Analysis

Priori Analysis
- -Algorithms
- -Independent of PL
- -Independent of H/W
- -Time & Space

Program
- -Implementation
- -Programmer
- -Programming Languages
- -H/W and OS
- -Testing

Posterior Analysis
- - Program
- -Dependent on PL
- -Dependent on H/w
- -Time

```
Algorithm Swap(a,b)
{
    temp = a;
    a = b;
    b = temp;
}
```

Time

temp = a; → 1
a = b; → 1
b = temp; → 1

$f(n) = \boxed{3}$ → O(1)

Space

a → 1
b → 1
temp → 1

$S(n) = \boxed{3}$ words → O(1)

```
x = 5*a+6*b---> 1
x = 5*a+6*b
x = 5*a+6*b
x = 5*a+6*b
x = 5*a+6*b
----------------------------
f(n) =5 ---> O(1)
```

Frequency Count Method

n=5   i=0 1 2 3 4 5

	0	1	2	3	4
A	8	3	9	7	2

Time   Space

```
Algorithm Sum(A,n)
{
      S=0;                        1
      for(i=0;i<n;i++)            n+1
      {
            S=S+A[i]              n
      }
      retun S;                    1
}
```

A --->n
n ---->1
S ---->1
i ---->1

$f(n)=2n+3$ $S(n)=n+3$

$O(n)$ $O(n)$

Space

A- n^2

B- n^2

C- n^2

n,i,j -1+1+1=3

$S(n) = 3n^2+3$

$O(n^2)$

A=

nxn= n^2

3x3 = 9

Time

```
Algorithm Add(A,B,n)
{
    for(i=0;i<n;i++)              n+1
    {
        for(j=0;j<n;j++)          n+1
        {
            C[i,j]=A[i,j]+B[i,j]   n
        }
    }
}
```

n+1 / n-1

n * (n+1) = n^2

n * n = n^2+n

$2n^2+2n+1$

$O(1)$ $O(n)$ $O(n^2)$ $O(n^3)$

f(n)=n+1+n^2+1+n^2

=2n^2+2n+1

$= 2n^2+2n+1 \Rightarrow O(n^2)$

```
Algorithm Multiply(A,B,n)
{
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            C[i,j]=0;
            for(k=0;k<n;k++)
            {
            C[i,j]= C[i,j]+A[i,k]*B[k,j]
            }
        }
    }
}
```

Time

3×3 9 → 25 n=5

$n+1$

$n(n+1)$

$n \cdot n$

$n \cdot n(n+1)$

$n \cdot n \cdot n$

$f(n) = 2n^3 + 3n^2 + 2n + 1$

$O(n^3)$

Space

A= n^2
B= n^2
C= n^2
n,i,j,k=4

$S(n) = 3n^2 + 4$

$O(n^2)$

Method 2

1 2 3 4 5 6 ....n=n(n+1)/2

f(n) = n(n+1)/2

=O(n^2)

```
for(i=0;i<n;i++)
{
for(j=0;j<i;j++)
{
stmt;
}
}
```

$n+1$

$n(n+1)$

$n(n)$

$f(n) = n^2$

$O(n^2)$

Method 1

```
for(i=1;p<=n;i++)
{
    p=p+i;
}
```


Assume, $P > n$ ——— (1)

$P = \frac{k(k+1)}{2}$

$\frac{k(k+1)}{2} > n$

$\frac{k^2+k}{2} > n$

$k^2 > n$

$k > \sqrt{n}$

$O(\sqrt{n})$

Summary

```
for(i=0;i<n;i++)          ——  O(n)
for(i=0;i<n;i=i+2)        ——  n/2 → O(n)
for(i=n;i>1;i--)          ——  O(n)
for(i=1;i<n;i=i*2)        ——  O(log2 n)
for(i=1;i<n;i=i*3)        ——  O(log3 n)
for(i=n;i>1;i=i/2)        ——  O(log2 n)
```

`for(i=0;i<n;i++)`	$O(n)$
`for(i=0;i<n;i=i+2)`	$\frac{n}{2} \Rightarrow O(n)$
`for(i=n;i>1;i--)`	$O(n)$
`for(i=1;i<n;i=i*2)`	$O(\log_2 n)$
`for(i=1;i<n;i=i*3)`	$O(\log_3 n)$
`for(i=n;i>1;i=i/2)`	$O(\log_2 n)$

Types of Time Functions

O(1) → Constant

O(log n) → Logrithmic

O(n) → Linear

O(n^2) → Quadratic

O(n^3) → Cubic

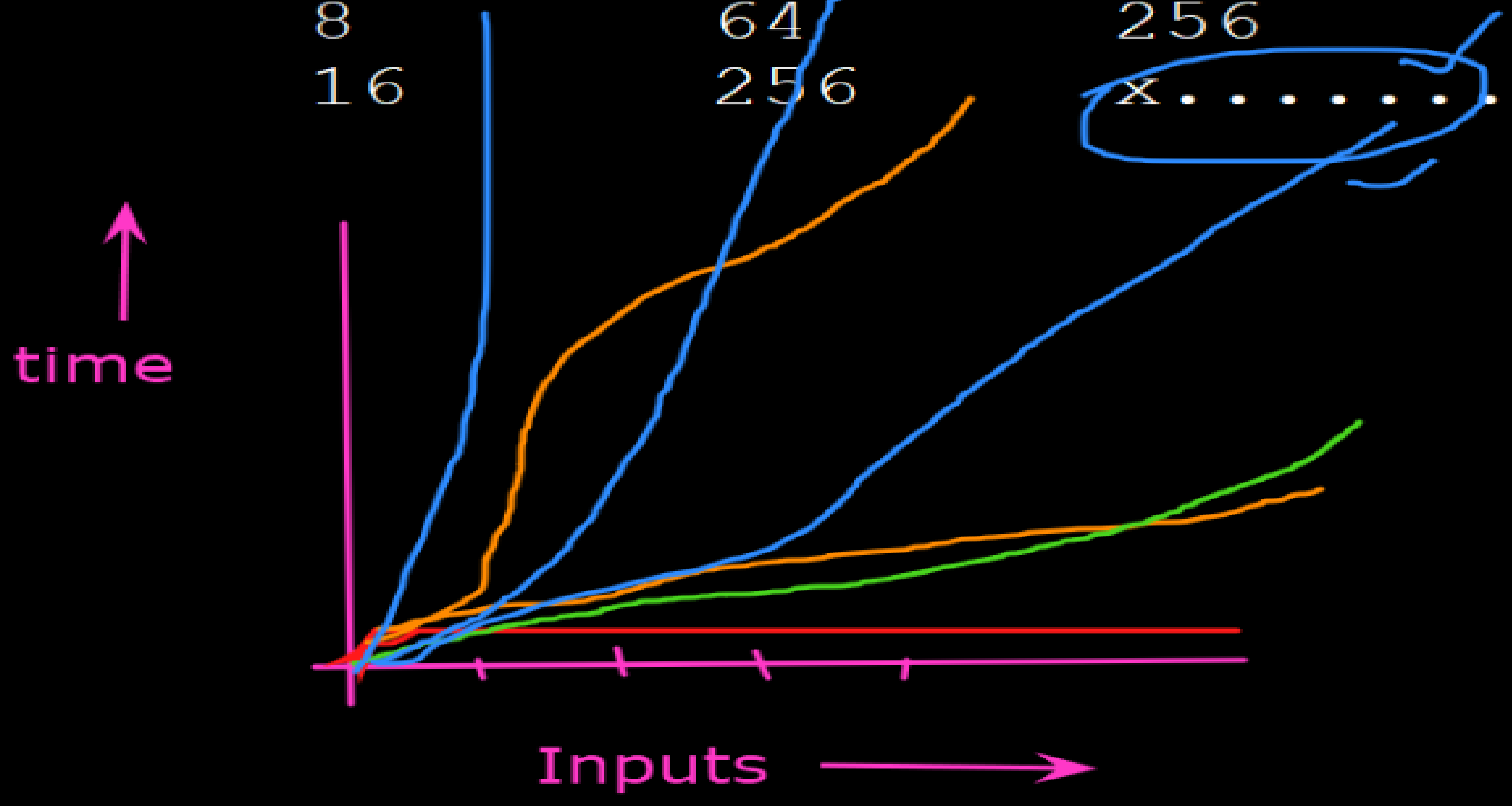
O(2^n) → Exponential

O(3^n)

O(n^n)

1<logn<sqrt(n)<nlogn<n^2<n^3<----<2^n<3^n

log n	n	n^2	2^n
1	2	4	4
2	4	16	16
3	8	64	256
4	16	256	x.......



$\log 4$

$\log 2^2$

$2(\log 2)$

$2(1) = 2$

Asymptotic Notations

The commonly used asymptotic notations used for calculating the running time complexity of an algorithm is given below:

- Big oh Notation (?)
- Omega Notation (Ω)
- Theta Notation (θ)

Asymptotic Notations

O → Big-oh ⇒ Upper bound ⇒ <u>Max</u>

Ω → Big-omega ⇒ Lower bound ⇒ <u>Min</u>

Θ → Theta ⇒ Averange bound ⇒ Range

```
Asymptotic Notations
-------------------------
Big-Oh:
---------
O(f(n)) = { g(n) : there exists c > 0 and n0
such that f(n) ≤ c.g(n) for all n > n0. }


f(n)= 2n+3    c=5, g(n)=n
f(n) <= c*g(n)
2n+3 <= 7*n
Let,n=1
5 <= 10*1
```

O(n) => Upperbound

```
1<logn<sqrt(n)<nlogn<n<n^2<n^3<....
```

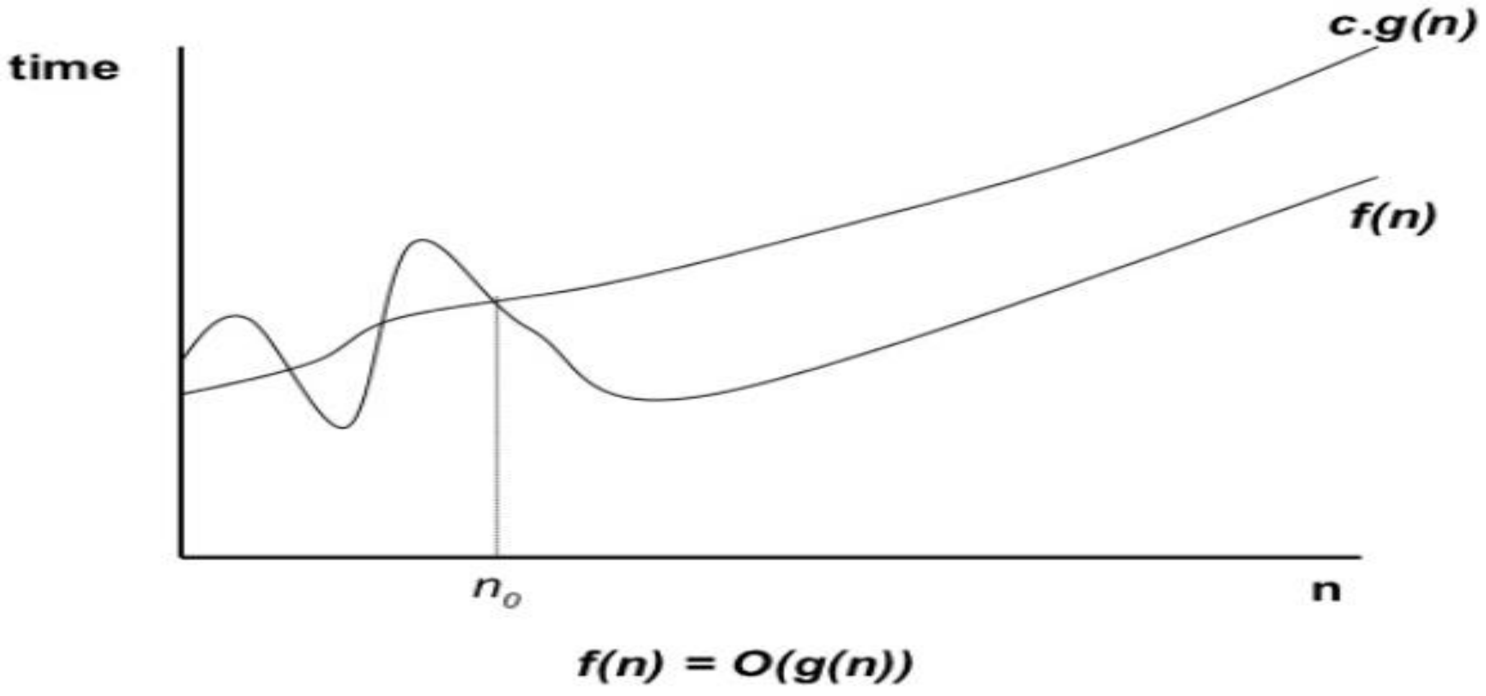
Vaild

Lowerbound

Upperbound

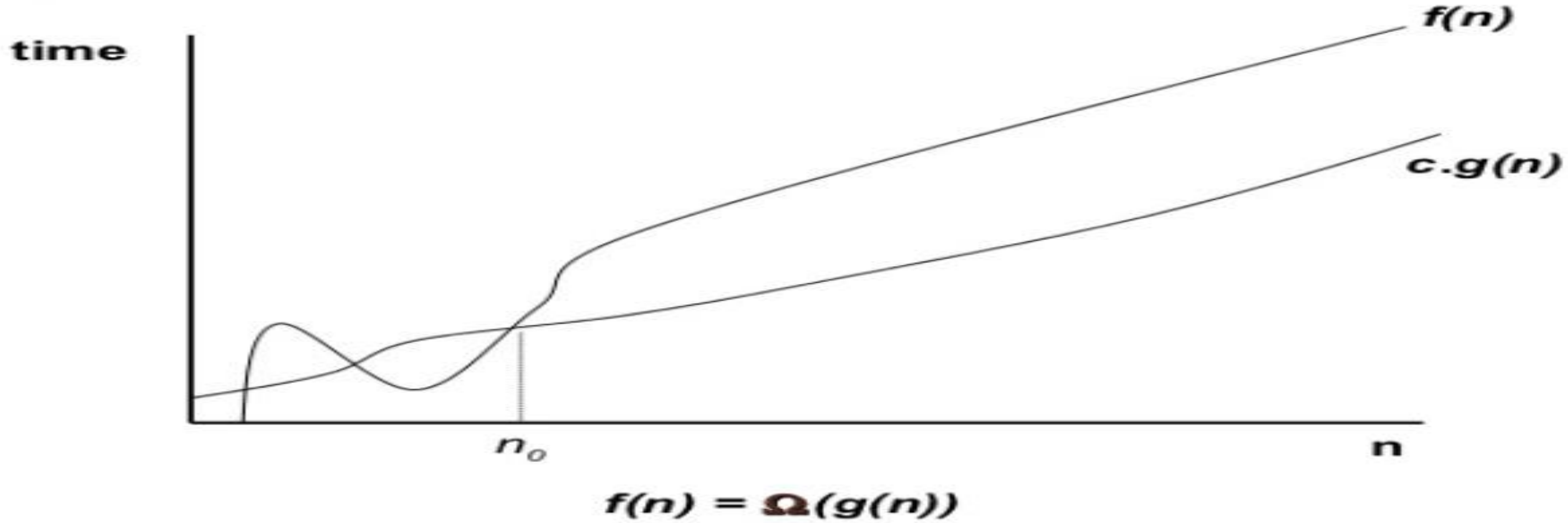
Tight bound

Big-O notation
(Upper Bound – Worst Case)



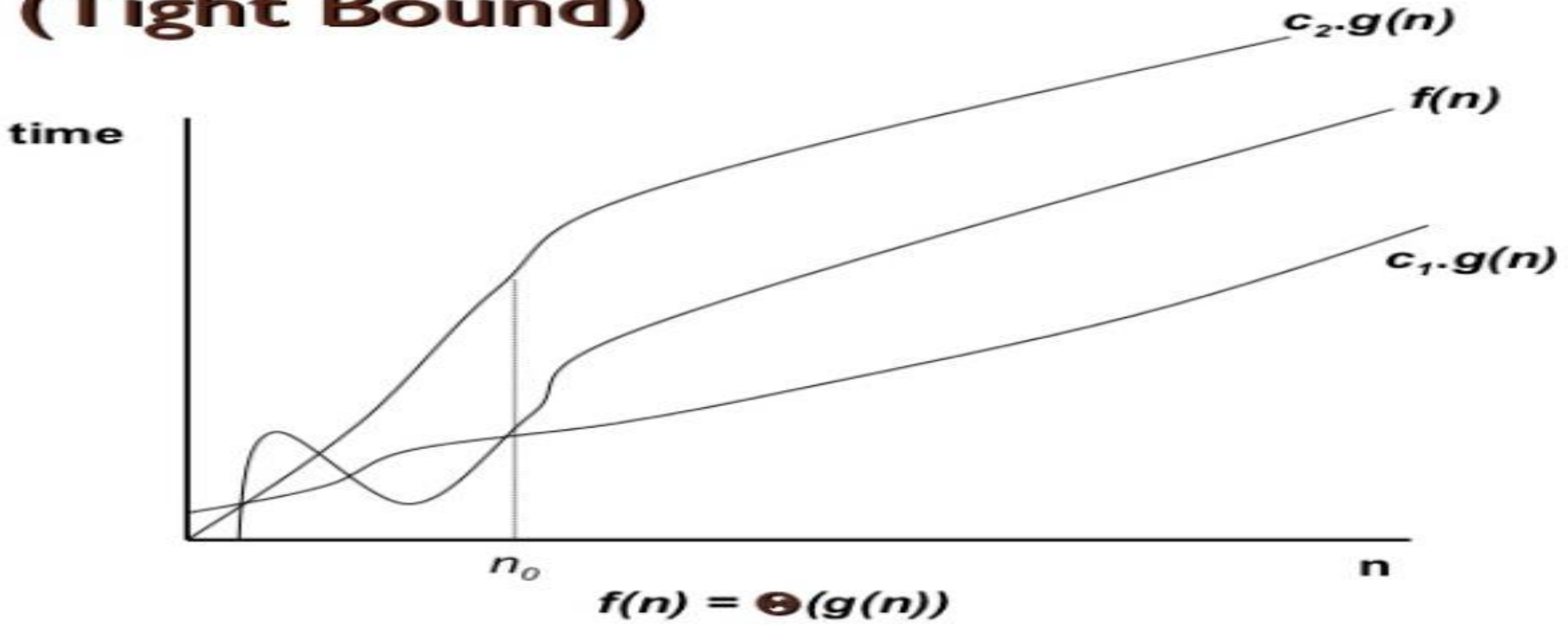
O(f(n)) = { g(n) : there exists c > 0 and n0 such that f(n) ≤ c.g(n) for all n > n0. }

Ω-notation
(Lower Bound – Best Case)



Ω(f(n)) ≥ { g(n) : there exists c > 0 and n0 such that g(n) ≤ c.f(n) for all n > n0. }

Θ notation (Theta) (Tight Bound)



θ(f(n)) = { g(n) if and only if g(n) = O(f(n)) and g(n) = Ω(f(n)) for all n > n0. }

Best, Worst and Average C

Linear Search:

0 1 2 3 4 5 6 7 8 9

A={8,5,9,3,14,17,5,19,4,5} n=10

Key= 8 no of Comparison= 1 true → BEST case O(1)

key= 17

Key 20 no of Comparison = 10 false → WORST case O(n)

Best Case:

Worst Case: Avg =(1+2+3....n)/n

=(n(n+1)/2)/n

O(n)

Binary Search Tree:

Binary Search Tree:

BST: 20,10,30,5,15,25,40

Key :20

BEST CASE: Root= O(1)

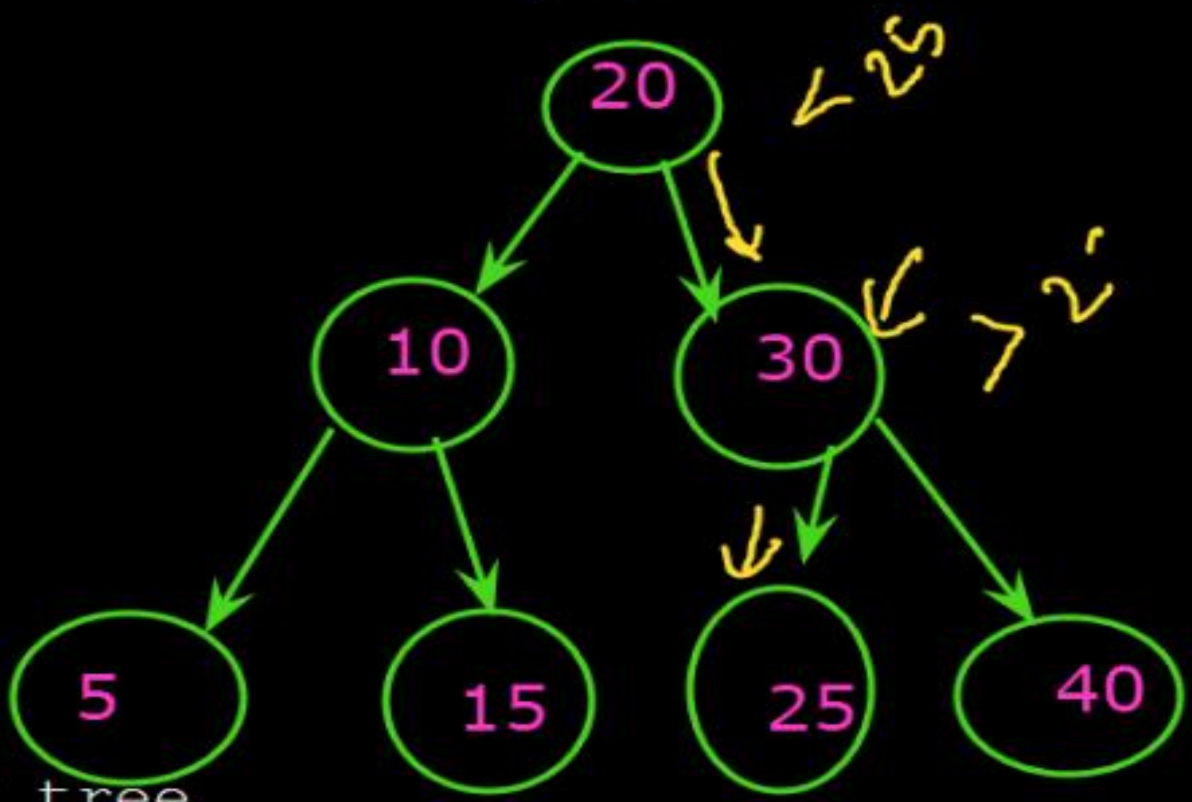
No of Comp=1

Key:25

Worst case: O(log n)= h

(Leaf node)

No of comp = 3 = Height of tree



BST : 45, 40, 30, 35, 20, 15, 5

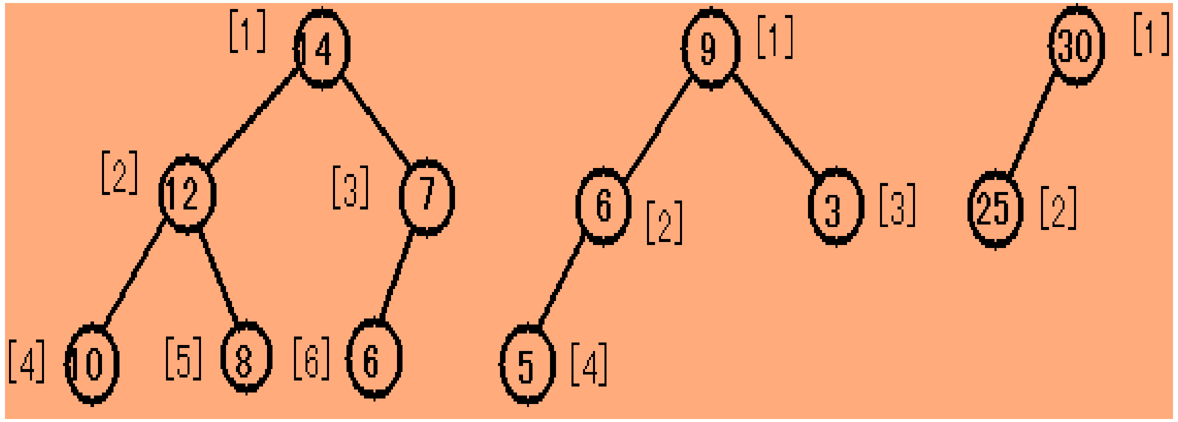
Heap

Module I

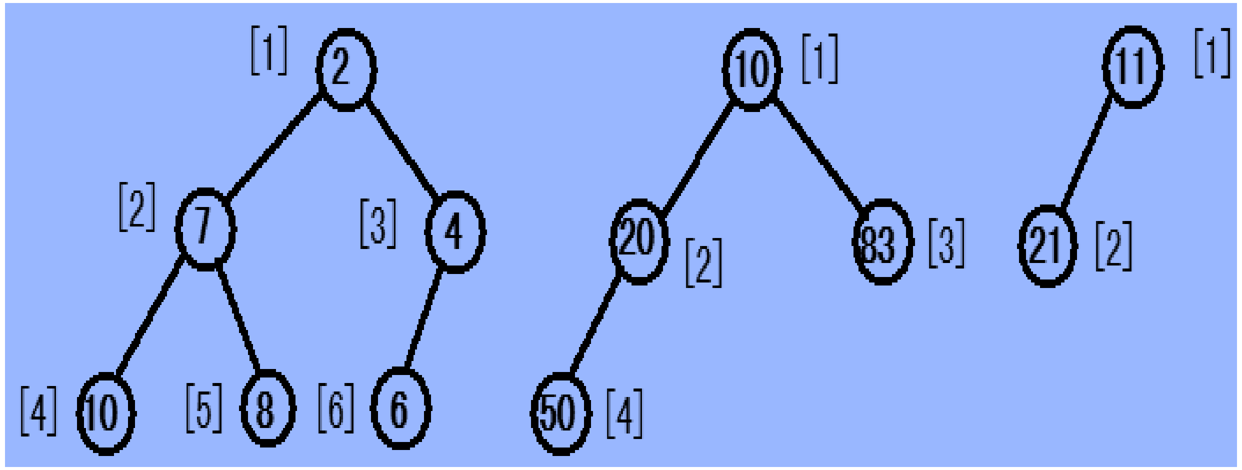
Kiran Waghmare

Heap

- **Example:**
  - Max-Heap

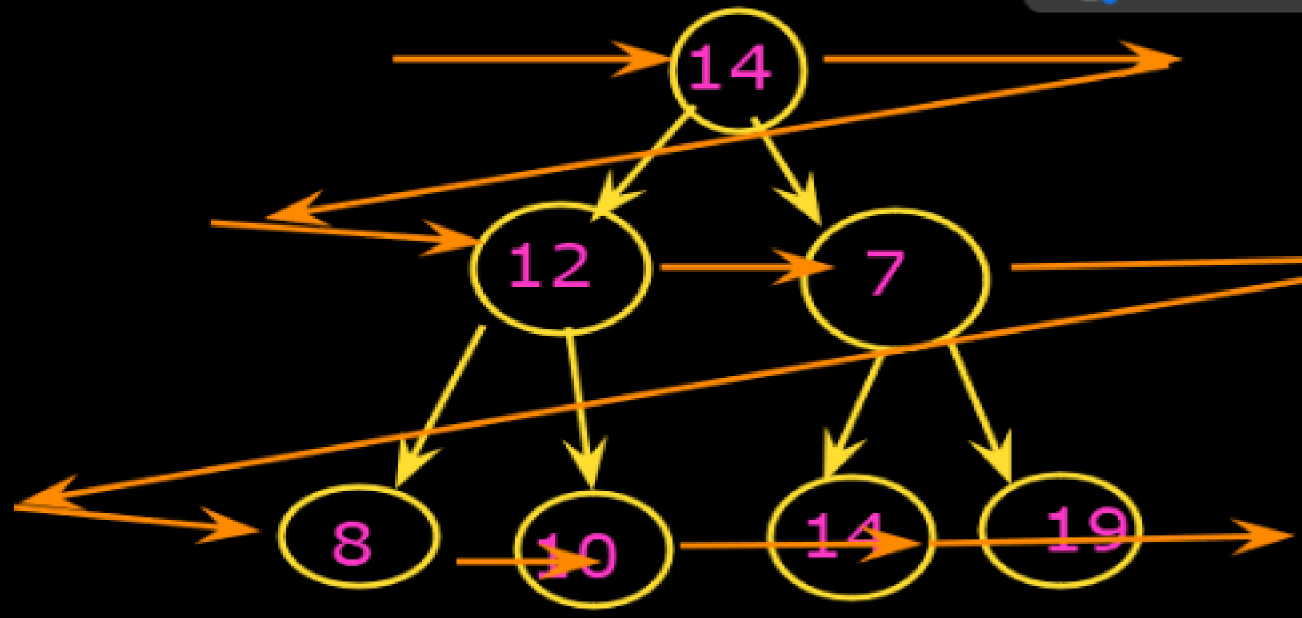


  - Min-Heap



Heap:

-Binary tree
14,12,7,8,10



Node(i)
Parent(i)=i/2
LC(i)=2*i
RC(i)=2*i+1

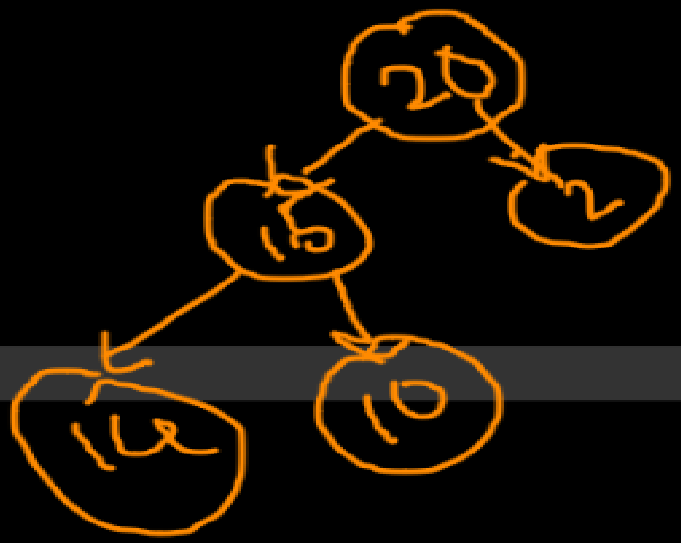
14	12	7	8	10	14	19
1	2	3	4	5	6	7

-Types of Heap
- -Min Heap
- -Max Heap

Heap:

-Binary tree
14,12,7,8,1

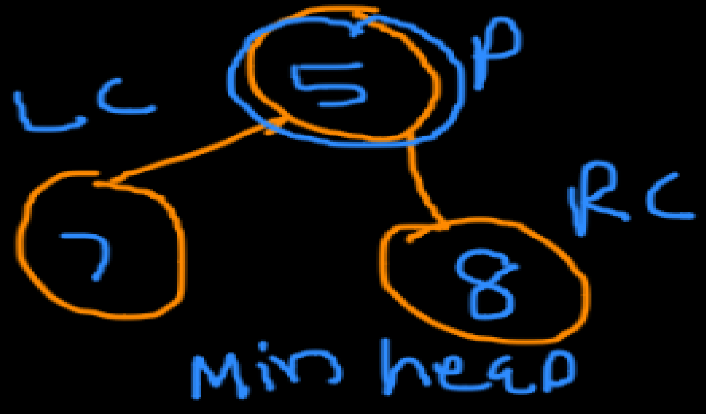
-Types of Heap
Heap: 5 14 23 32 41 87 90 50 64 53

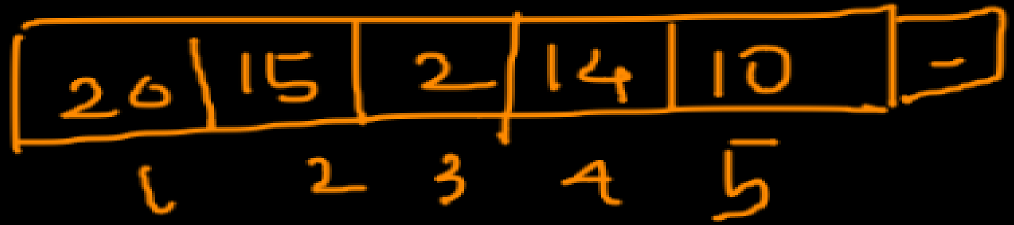


Ex: 20 15 2 14 10

-Min Heap
-Max Heap

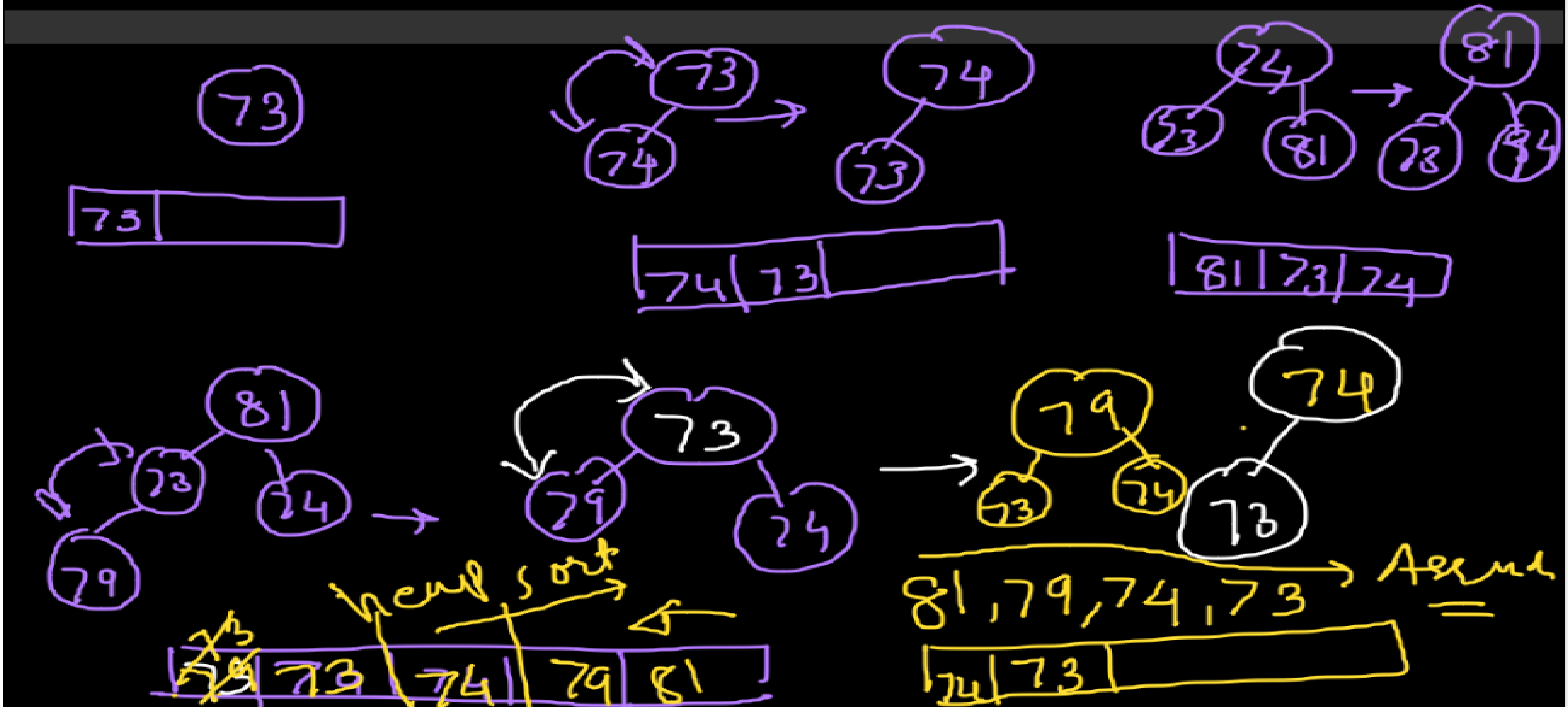
-BST





Ex: 20 15 2 14 10
Ex: 73, 74,81,79

Who can see what you share here? Recording



Who can see what you share here

```
void heapify(int arr[], int n, int i)
{
    int largest = i;
    int l = 2 * i + 1;
    int r = 2 * i + 2;

    if (l < n && arr[l] > arr[largest])
        largest = l;

    if (r < n && arr[r] > arr[largest])
        largest = r;

    if (largest != i) {
        int swap = arr[i];
        arr[i] = arr[largest];
        arr[largest] = swap;

        heapify(arr, n, largest);
    }
}
```

Example:- The fig. shows steps of heap-sort for list (2 3 7 1 8 5 6)