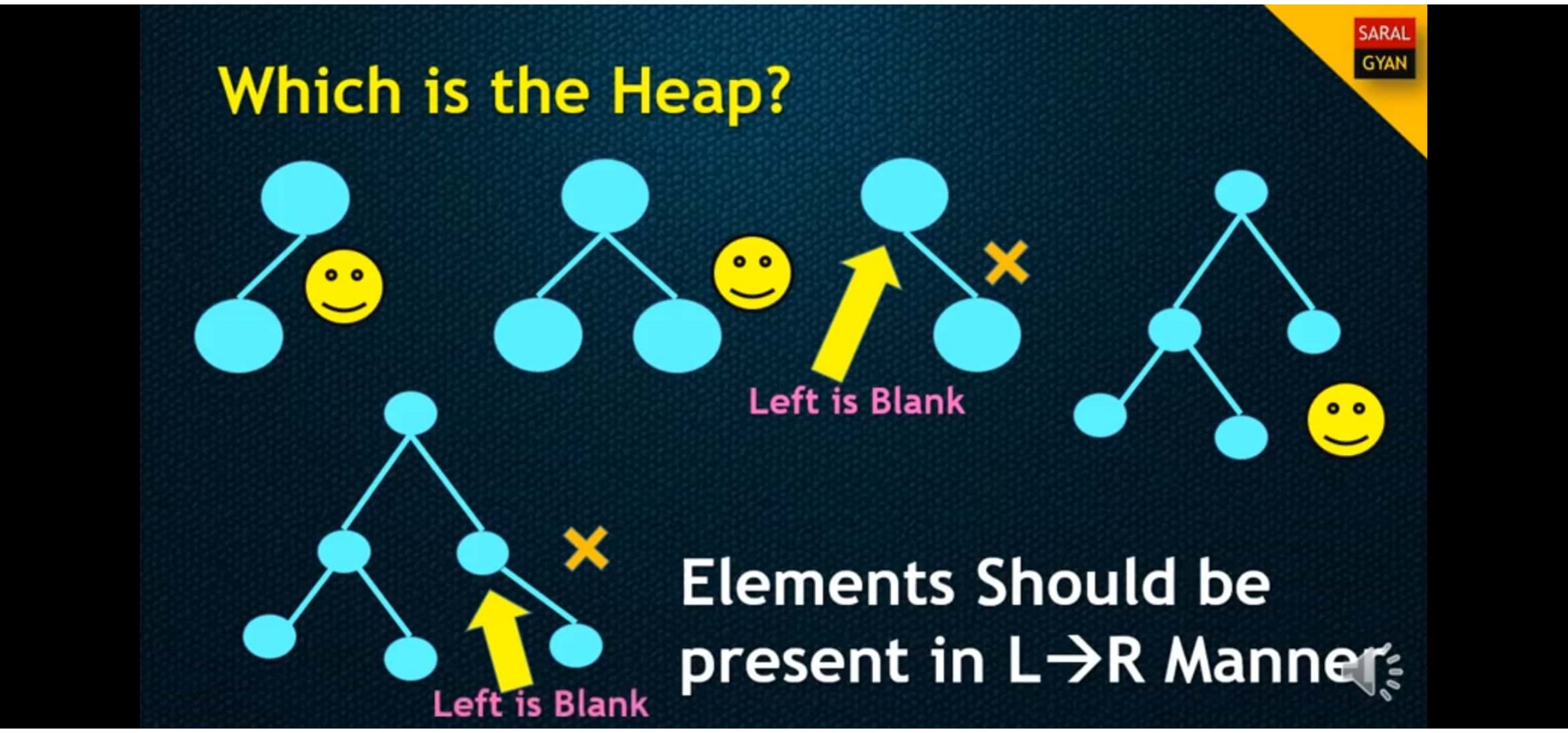
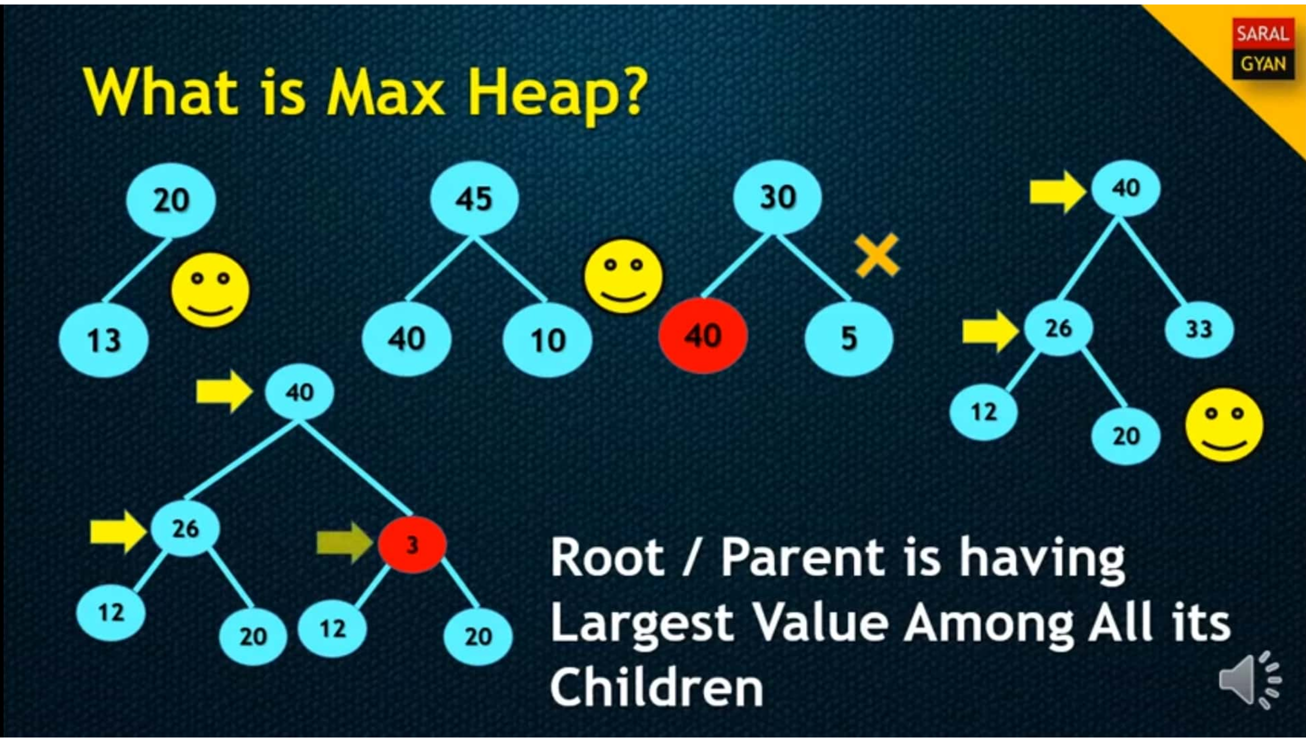
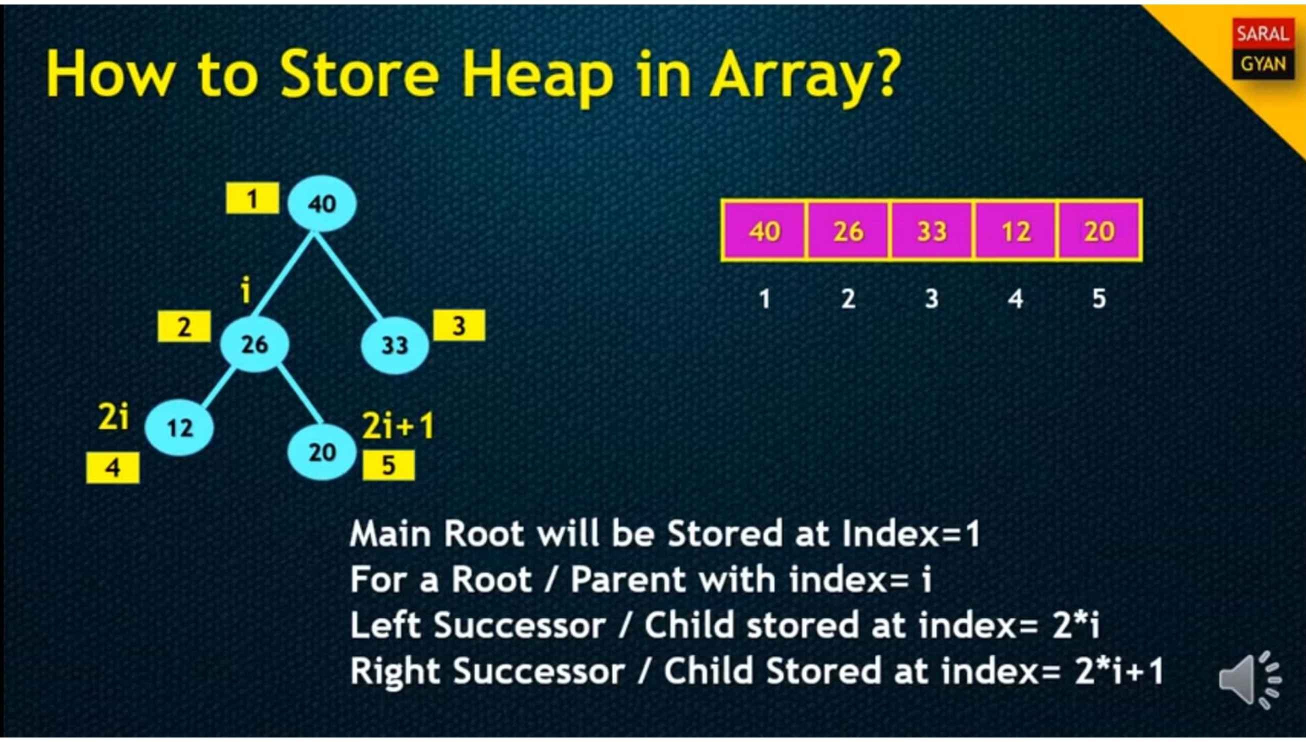
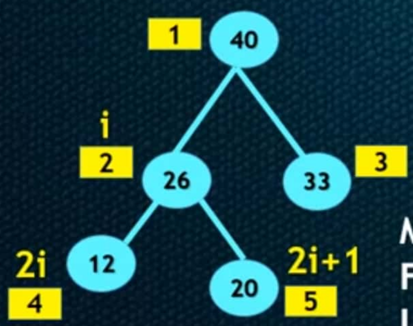

SARAL
GYAN
Which is the Heap?
Left is Blank
Left is Blank
Elements Should be
present in L→R Manner


SARAL
GYAN
What is Max Heap?
20
13
45
40
10
30
40
5
40
26
33
12
20
40
26
3
12
20
12
20
Root / Parent is having Largest Value Among All its Children


SARAL
GYAN
How to Store Heap in Array?
1
40
i
2
26
33
3
2i
12
20
2i+1
4
5
40 26 33 12 20
1 2 3 4 5
Main Root will be Stored at Index=1
For a Root / Parent with index= i
Left Successor / Child stored at index= 2*i
Right Successor / Child Stored at index= 2*i+1

Important Point

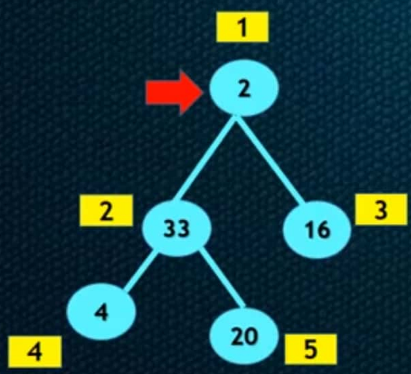


40	26	33	12	20
1	2	3	4	5

Main Root will be Stored at Index=1
For a Root / Parent with index= i
Left Successor / Child stored at index= 2*i
Right Successor / Child Stored at index= 2*i+1

If a Child is at index=i, then its parent will be at index= Lower bound (i/2) e.g. child=5, parent =LB(5/2)➔ 2

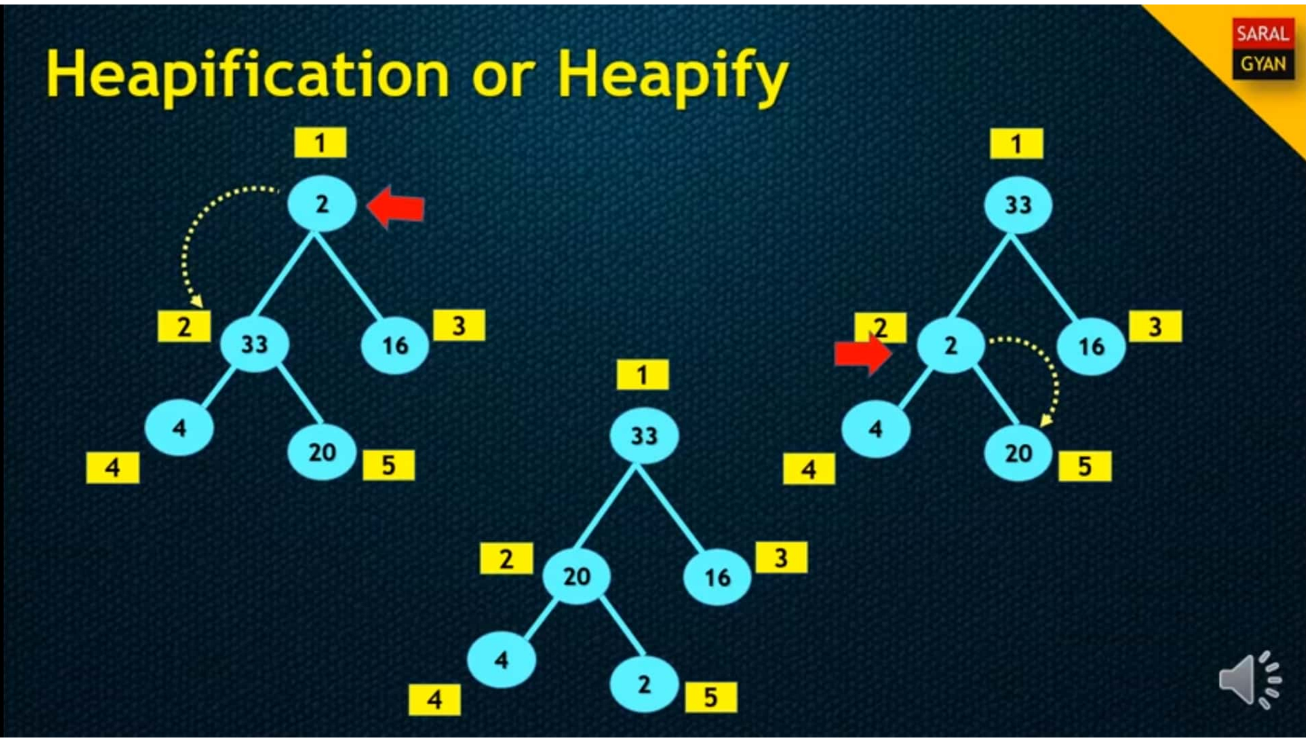
Heapification or Heapify



Step1: For a given Non-Leaf Node, Test Max Heap Property, if not satisfied

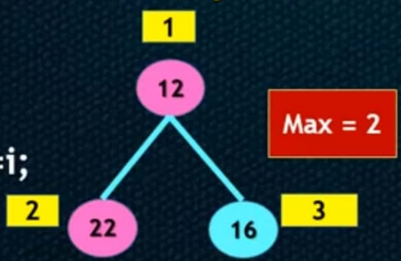
Step2: Swap the Node with the Larger Node Among its Children

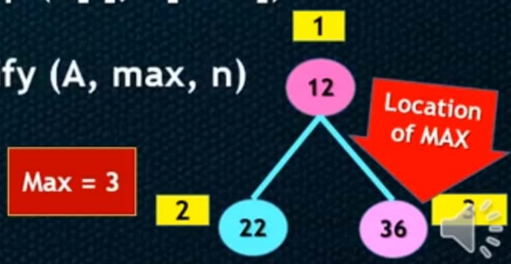
Step3: Repeat the Procedure for Child, Non-Leaf Node, till leaf node

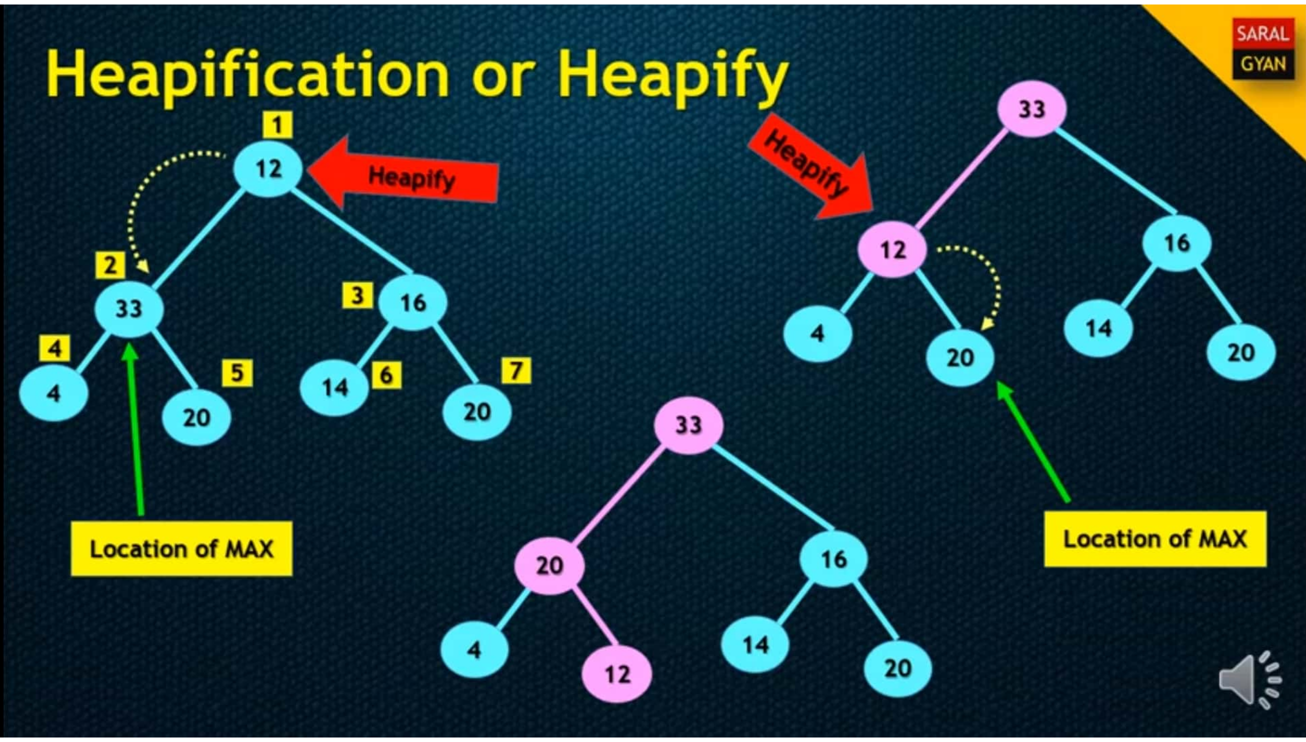

Heapification or Heapify
SARAL
GYAN
1
2
2
33
3
16
4
4
5
20
1
33
2
20
3
16
4
4
5
2
1
33
2
2
3
16
4
4
5
20

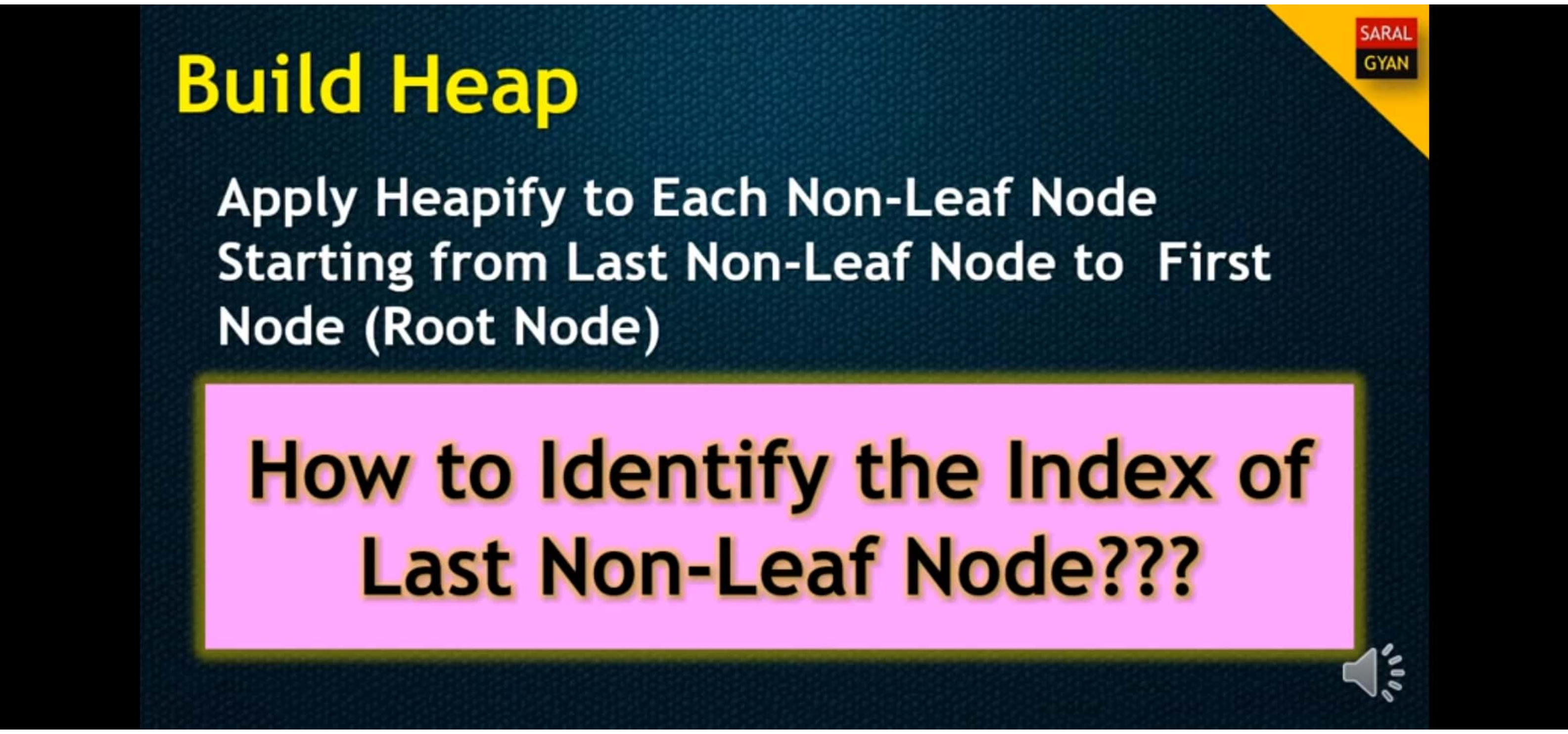
Heapification or Heapify Algorithm

```
Heapify (A, i, n)
{
  left=2i;
  right=2i+1;
  if (left<n and A[left]>A[i])
  {
    max=left; //only index
  }
  else
  {
  max=i;
  }
If(right<n and A[right]>A[max])
{
  max=right;
}
 if (i != max)
 {
   Swap (A[i], A[max])
 }
 Heapify (A, max, n)
```






Heapification or Heapify
SARAL
GYAN
Heapify
Heapify
1
12
2
33
3
16
4
4
5
20
14
6
7
20
Location of MAX
33
12
16
4
20
14
20
Location of MAX
33
20
16
4
12
14
20

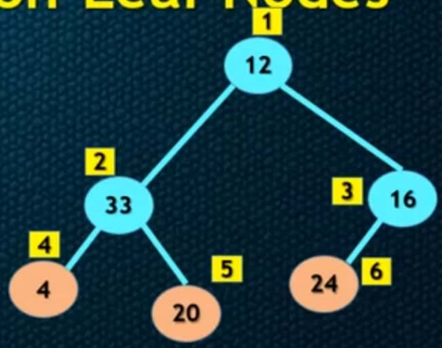

Build Heap
Apply Heapify to Each Non-Leaf Node
Starting from Last Non-Leaf Node to First
Node (Root Node)
How to Identify the Index of
Last Non-Leaf Node???
SARAL
GYAN

Identification of Leaf/ Non-Leaf Nodes

Index of First Leaf Node = [Lowerbound (N/2)+1]

e.g. N=6, Hence, 4 to 6 are the Leaf Nodes

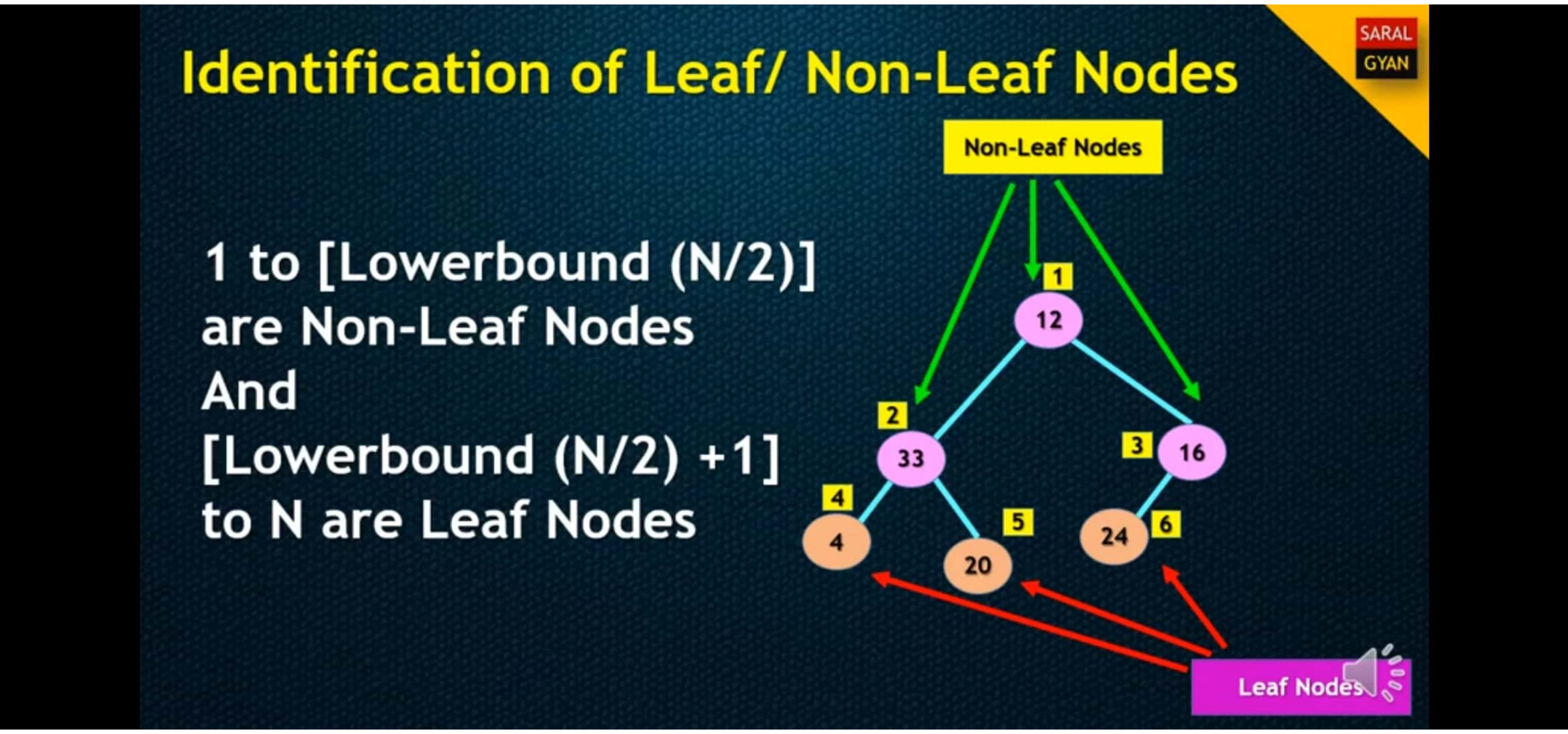
Identification of Leaf/ Non-Leaf Nodes



Index of First Leaf Node = [Lowerbound (N/2)+1]

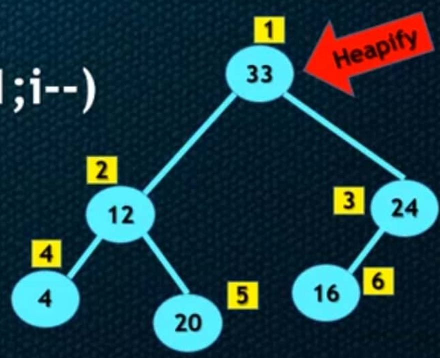
e.g. N=6, Hence, First Leaf Node is 3+1=4

SARAL GYAN


SARAL
GYAN
Identification of Leaf/ Non-Leaf Nodes
Non-Leaf Nodes
1 to [Lowerbound (N/2)]
are Non-Leaf Nodes
And
[Lowerbound (N/2) +1]
to N are Leaf Nodes
1
12
2
33
3
16
4
4
5
20
24
6
Leaf Nodes

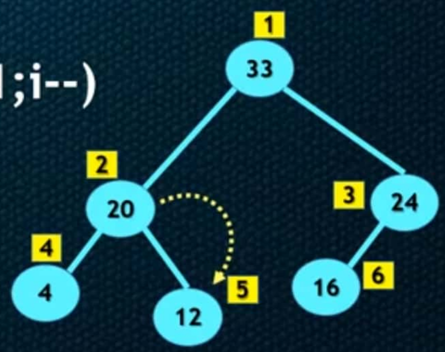
Build Heap Algorithm

```
Build_Max_Heap (A,N)
{
  for(i=lowerbound(N/2); i>=1;i--)
   {
     Heapify(A,i, N);
   }
}
```

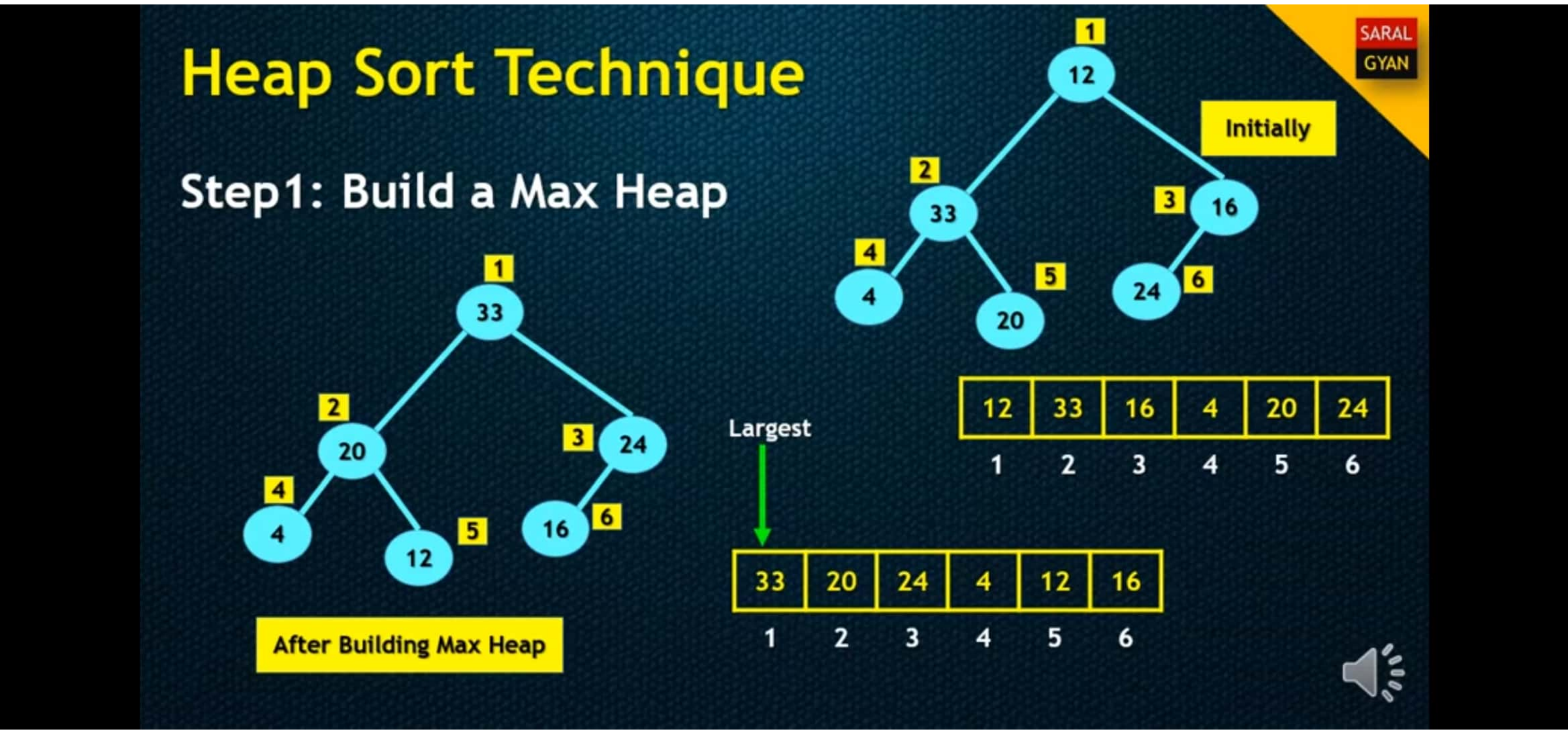


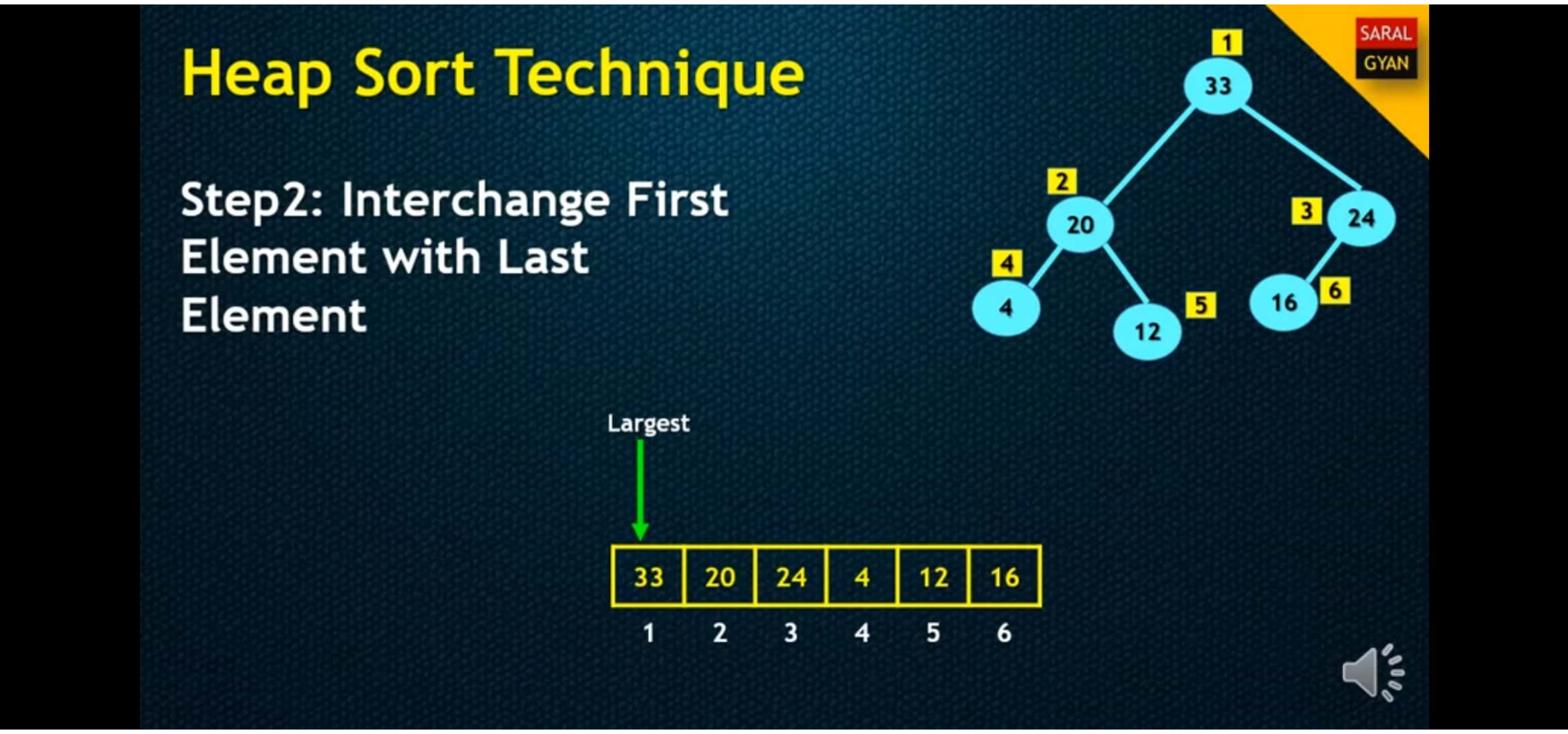
Build Heap Algorithm

```
Build_Max_Heap (A,N)
{
  for(i=lowerbound(N/2); i>=1;i--)
   {
     Heapify(A,i, N);
   }
}
```



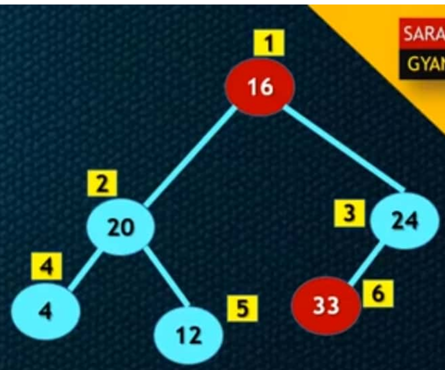
Max Heap


Heap Sort Technique
Step1: Build a Max Heap
Initially
12
33
16
4
20
24
1 2 3 4 5 6
12 33 16 4 20 24
1 2 3 4 5 6
After Building Max Heap
33
20
24
4
12
16
Largest
33 20 24 4 12 16
1 2 3 4 5 6
SARAL
GYAN


Heap Sort Technique
Step2: Interchange First Element with Last Element
SARAL
GYAN
1
33
2
20
3
24
4
4
5
12
6
16
Largest
33 20 24 4 12 16
1 2 3 4 5 6

Heap Sort Technique

Step2: Interchange First Element with Last Element

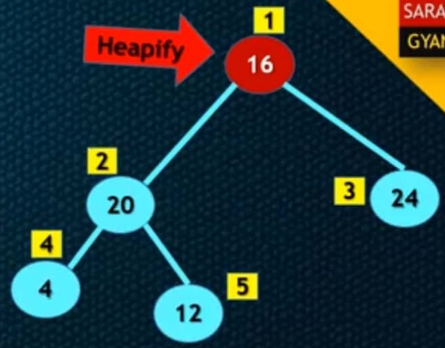


16	20	24	4	12	33
1	2	3	4	5	6

Heap Sort Technique

Step3:

a. Consider only N-1, N-2,..1 Elements
b. Apply Heapify on First Element
c. Repeat Step 2 and 3 Until We Reach
   upto Single Element



N-1

16	20	24	4	12	33
1	2	3	4	5	6

Heap Sort Technique

Step3:

a. Consider only N-1, N-2,..1 Elements
b. Apply Heapify on First Element
c. Repeat Step 2 and 3 Until We Reach
   upto Single Element

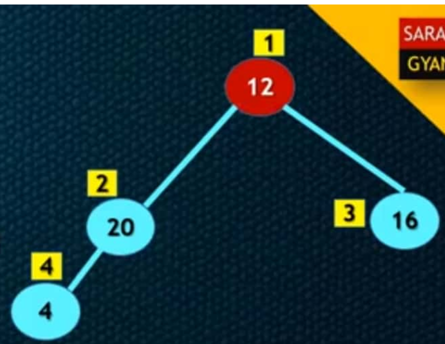
N-1

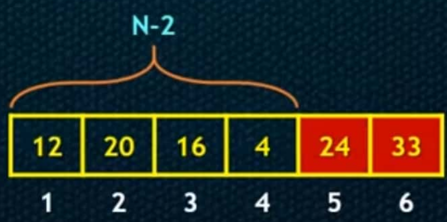
24	20	16	4	12	33
1	2	3	4	5	6

Heap Sort Technique

Step3:

a. Consider only N-1, N-2,..1 Elements
b. Apply Heapify on First Element
c. Repeat Step 2 and 3 Until We Reach
   upto Single Element





Heap Sort Technique

Step3:

a. Consider only N-1, N-2,..1 Elements
b. Apply Heapify on First Element
c. Repeat Step 2 and 3 Until We Reach
   upto Single Element

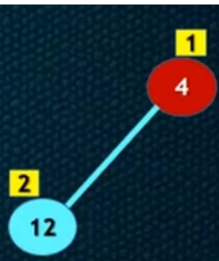
$\overbrace{\quad 4 \quad 12 \quad 16 \quad}^{N-3}$

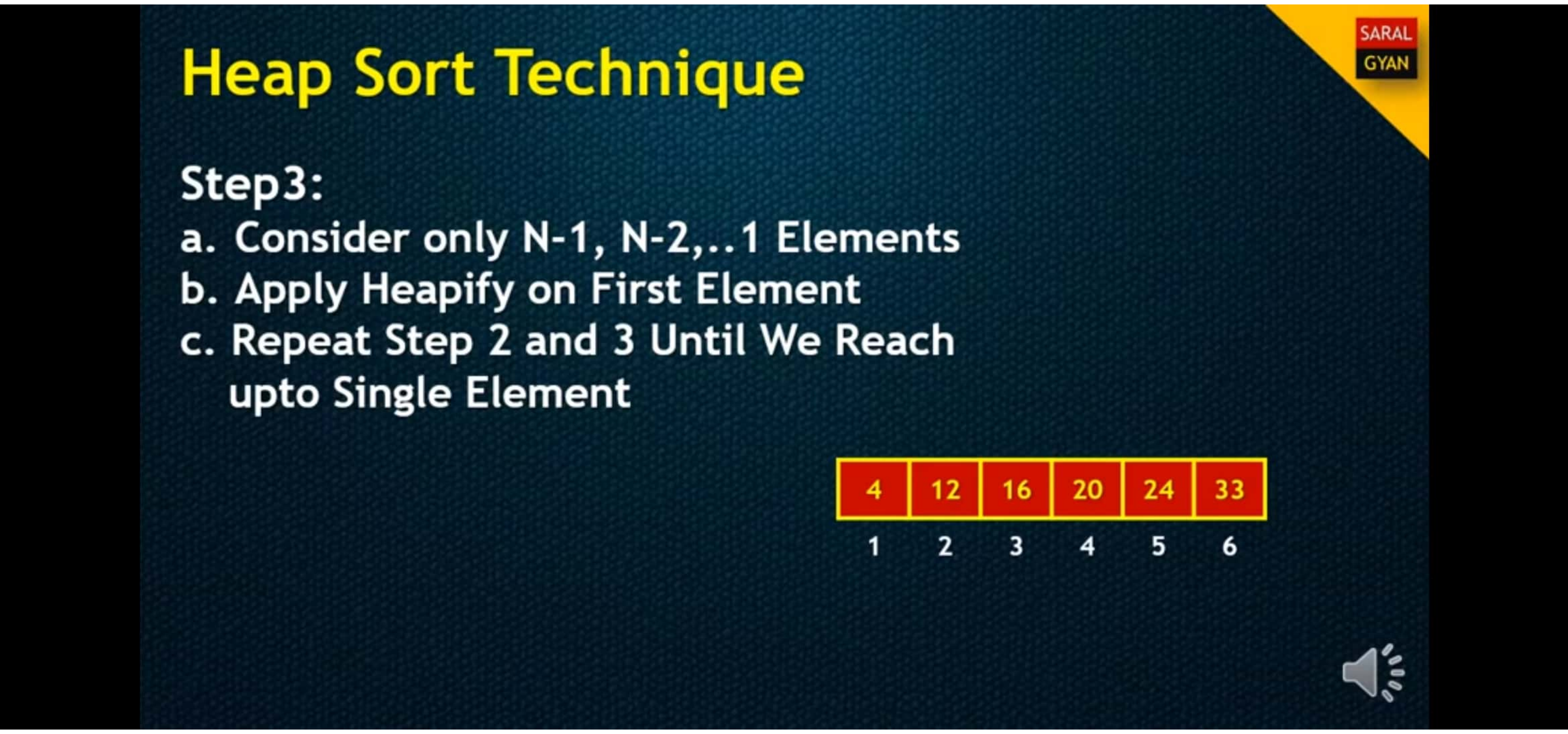
4	12	16	20	24	33
1	2	3	4	5	6

Heap Sort Technique

Step3:

a. Consider only N-1, N-2,..1 Elements
b. Apply Heapify on First Element
c. Repeat Step 2 and 3 Until We Reach
   upto Single Element




Heap Sort Technique
Step3:
a. Consider only N-1, N-2,..1 Elements
b. Apply Heapify on First Element
c. Repeat Step 2 and 3 Until We Reach upto Single Element
4 12 16 20 24 33
1 2 3 4 5 6
SARAL GYAN

Heap Sort Algorithm

SARAL GYAN

```
Heap_Sort(A,N)
{
   Build_Heap(A,N);
   for(k=N ; k>=1 ; k--)
   {
      Interchange (A[1], A[k]);
      Heapify (A,1,k-1);
    }
}
```

1	2	3	4	5	6
33	20	24	4	12	16
16	20	24	4	12	33
24	20	16	4	12	33
12	20	16	4	24	33
20	12	16	4	24	33
4	12	16	20	24	33
16	12	4	20	24	33
4	12	16	20	24	33
12	4	16	20	24	33
4	12	16	20	24	33
4	12	16	20	24	33

Complexity of Heap Sort Algorithm

SARAL GYAN

```
Heap_Sort(A,N)
{
   Build_Heap(A,N);               <----  Time= O (n)
   for(k=N ; k>=1 ; k--)          <----  Time= O (n)
   {
      Interchange (A[1], A[k]);   <----  Time= Constant
      Heapify (A,1,k-1);          <----  Time= O(n x Log (n))
   }
}
```

Total Time= n + n Log (n)
Hence, Complexity = O(n log (n))