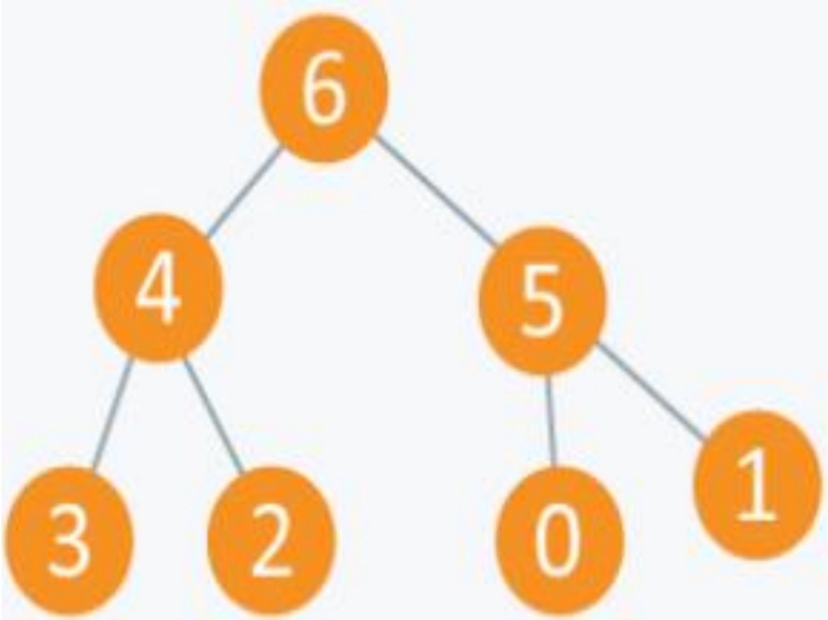
Heap and Heap Sort

CSIT 3RD SEM

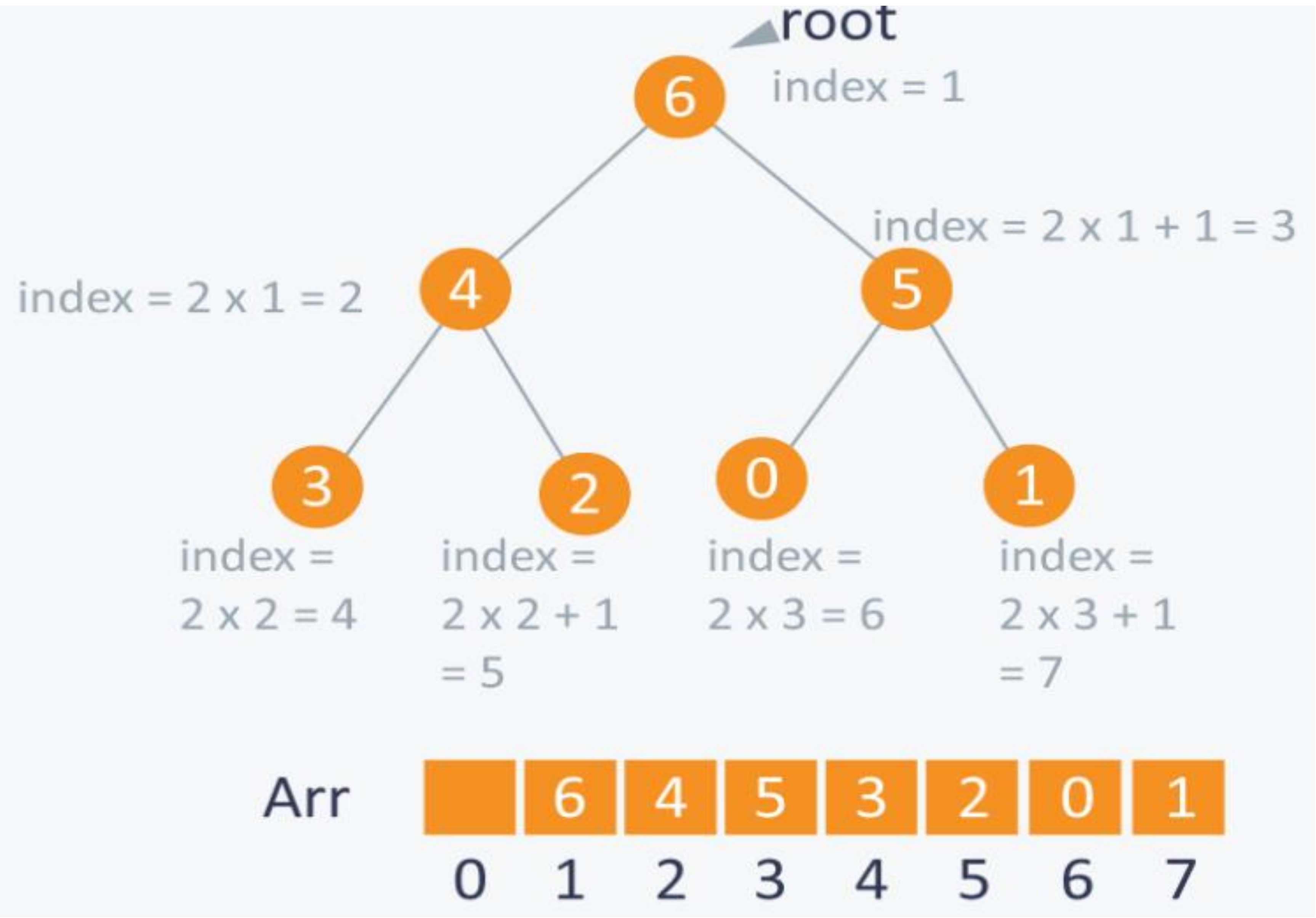
Compiled by: Chakra Narayan Rawal

Heap

- A heap is a tree-based data structure in which all the nodes of the tree are in a specific order.
- The maximum number of children of a node in a heap depends on the type of heap. However, in the more commonly-used heap type, there are at most **2 children** of a node and it's known as a **Binary heap.**
- In binary heap, if the heap is a complete binary tree with **N** nodes, then it has smallest possible height which is $\mathbf{\log_2 N}$



- An array can be used to simulate a tree in the following way.
- If we are storing one element at index *i* in array *Arr*, then its parent will be stored at index ***i/2*** (unless its a root, as root has no parent) and can be accessed by **Arr[i/2],** and
- its left child can be accessed by **Arr[2∗i]** and
- its right child can be accessed by **Arr[2∗i+1]**. Index of root will be 1 in an array.


root
index = 1
index = 2 x 1 + 1 = 3
index = 2 x 1 = 2
6
4
5
3
2
0
1
index =
2 x 2 = 4
index =
2 x 2 + 1
= 5
index =
2 x 3 = 6
index =
2 x 3 + 1
= 7
Arr
6 4 5 3 2 0 1
0 1 2 3 4 5 6 7

Heaps can be of two types:

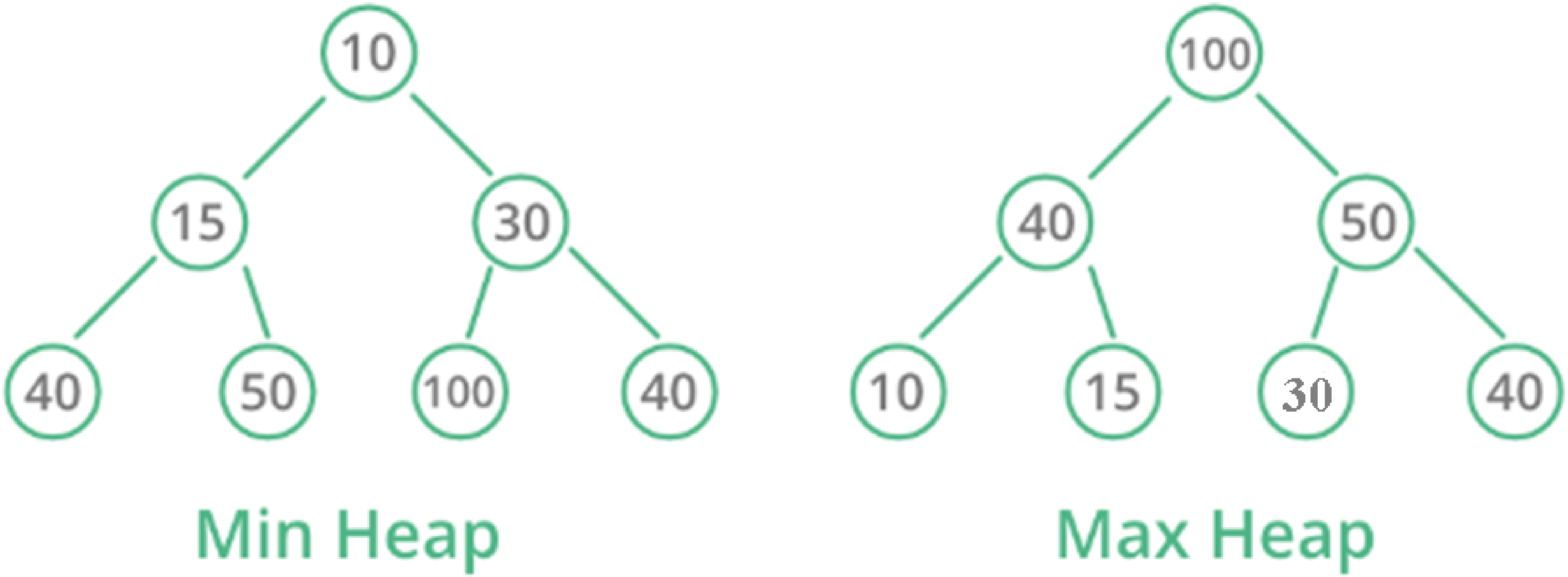
Max-Heap:

- In a Max-Heap the key present at the root node must be greatest among the keys present at all of it's children.
- The same property must be recursively true for all sub-trees in that Binary Tree.

Min-Heap:

- In a Min-Heap the key present at the root node must be minimum among the keys present at all of it's children.
- The same property must be recursively true for all sub-trees in that Binary Tree.

Heap Data Structure



Min-max heap

A min-max heap is defined as a complete binary tree containing alternating min (or even) and max (or odd) levels.

Properties of Min-max heap

- Each node in a min-max heap is associated with a data member (usually called key) whose value is implemented to calculate the order of the node in the min-max heap.
- The root element is the minimum element in the min-max heap.
- One of the two elements in the second level, which is a max (or odd) level, is the maximum element in the min-max heap

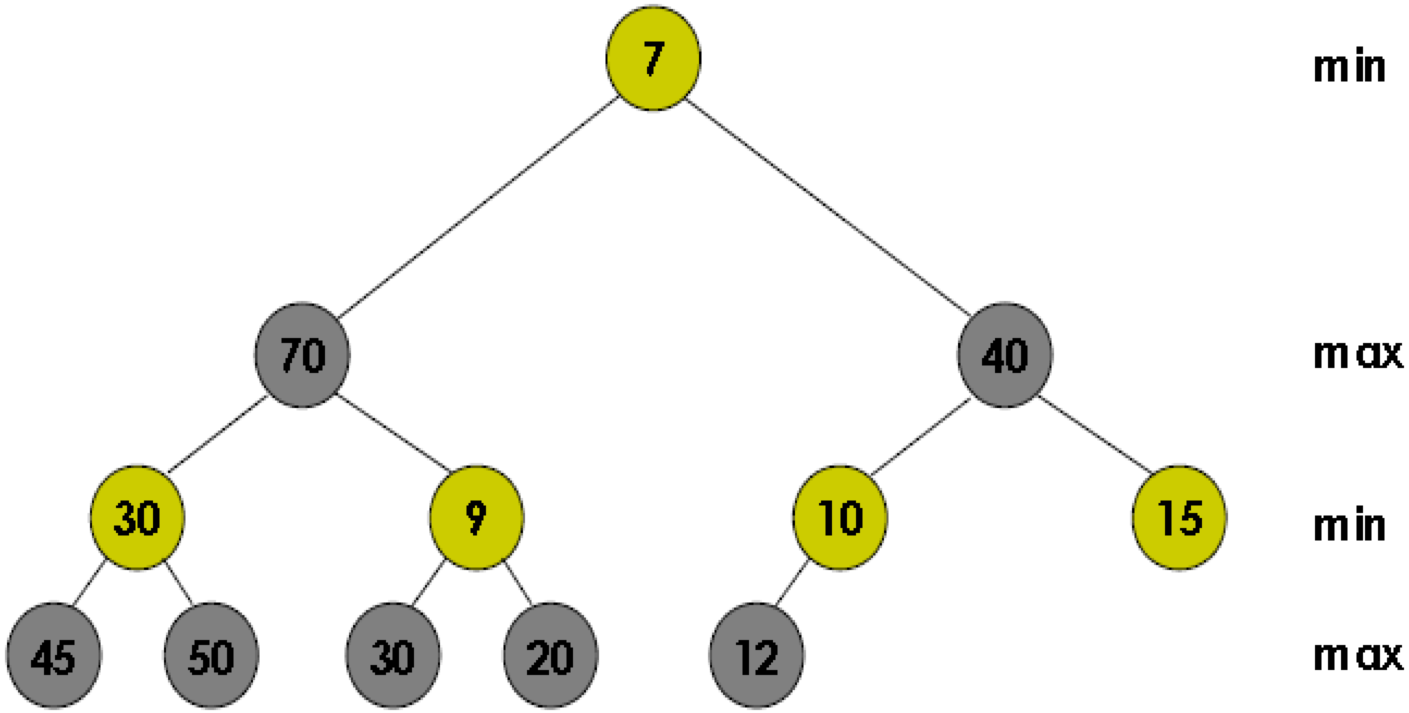


Fig. A min-max heap

Note: A max-min heap is defined as opposite to min-max heap; in such a heap, the highest value is stored at the root, and the minimum value is stored at one of the root's children.

Heap Construction (Construct Max-heap)

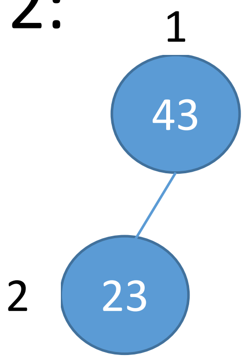
43 23 74 11 65 58 94 36 99 87

Step 1:

43

step 2:

23



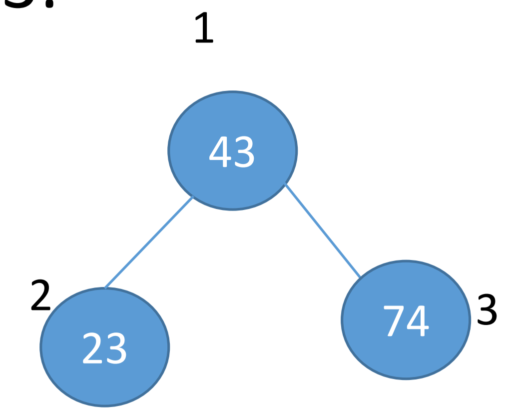
Now check max property i=2 and compare with root.

=floor(i/2)=1

it means compare node 2 with node 1, Which satisfy property of heap.

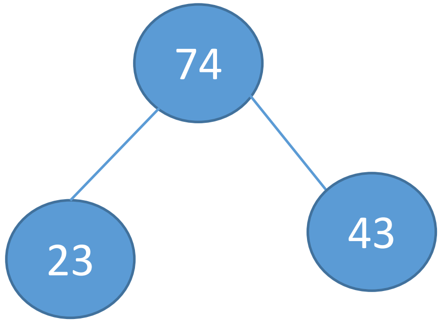
Step 3:

74

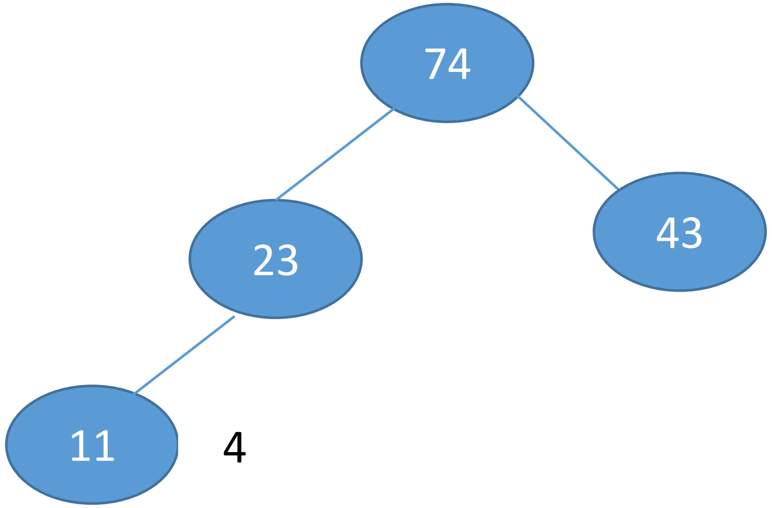


Again check heap proptertyi=3, =floor(i/2)=1. it’s not satisfy heap property so heapify. Swap 74 and 43

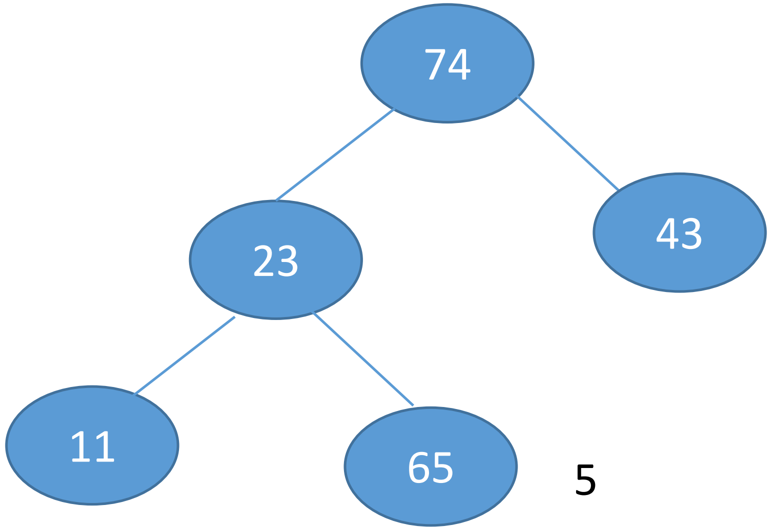
After swapping



similarly 11

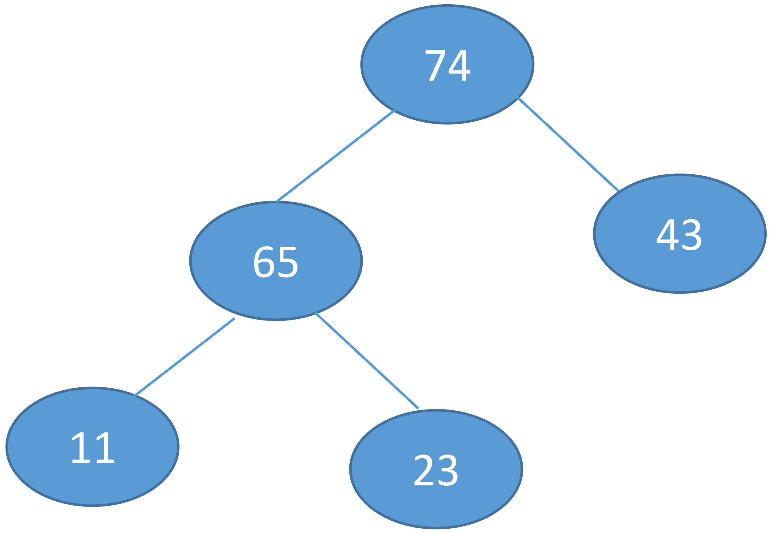


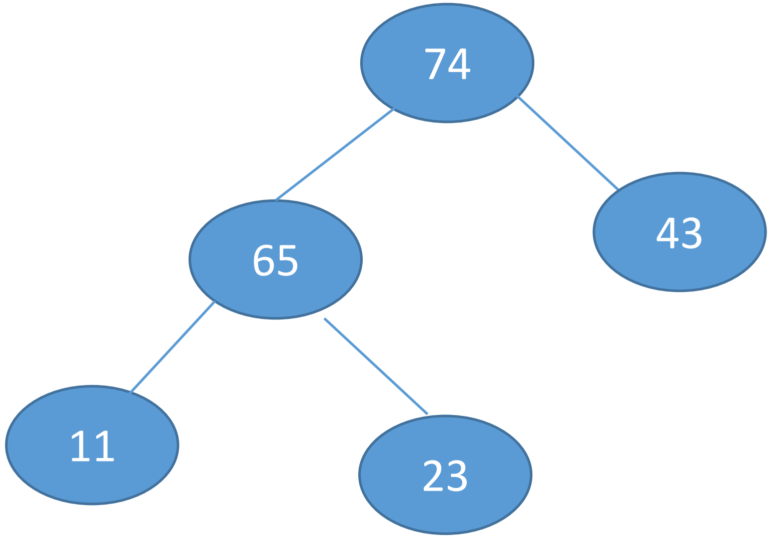
Again 65



Check max heap property which is not maintain because of element position 5 so heapify

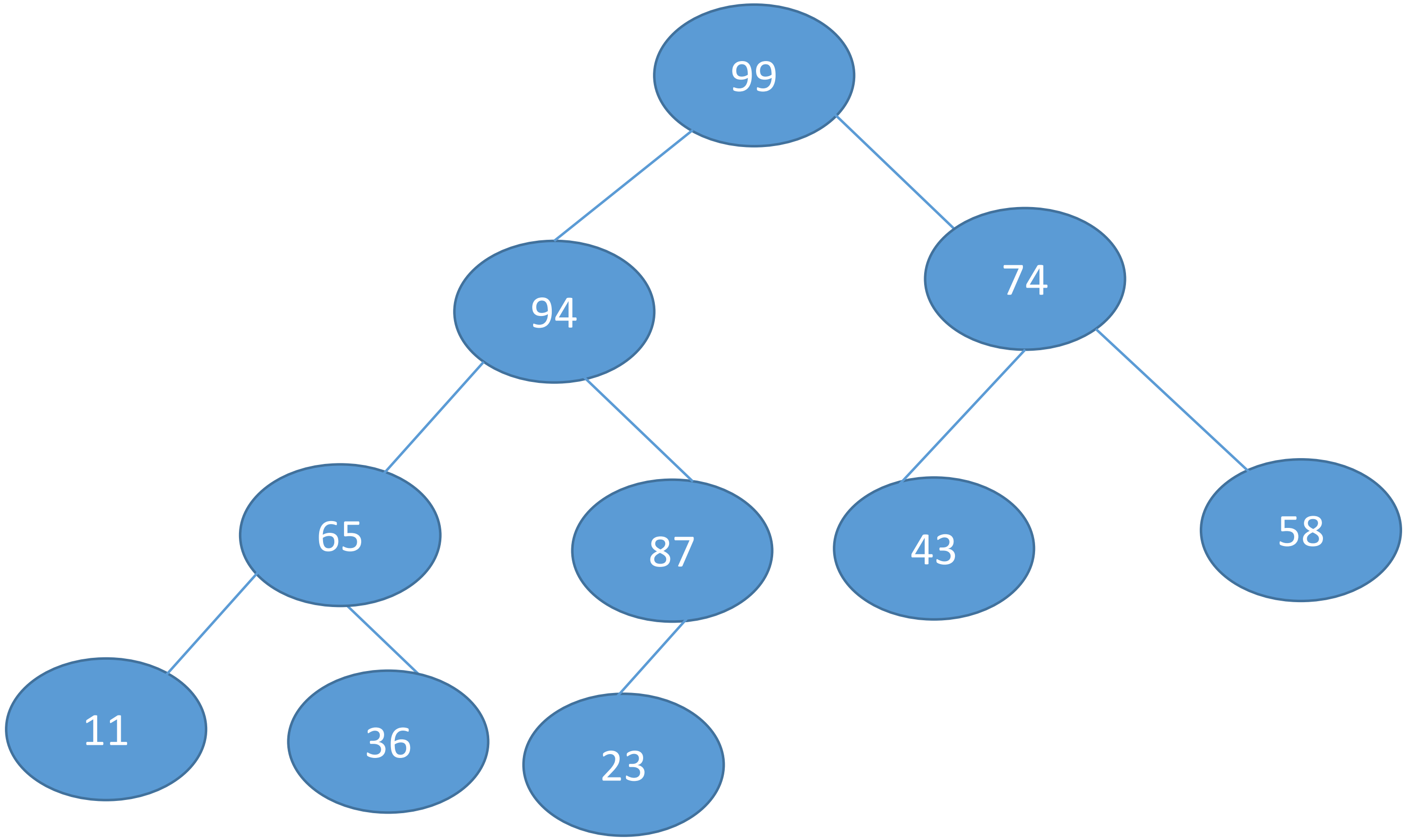
So the heap is like



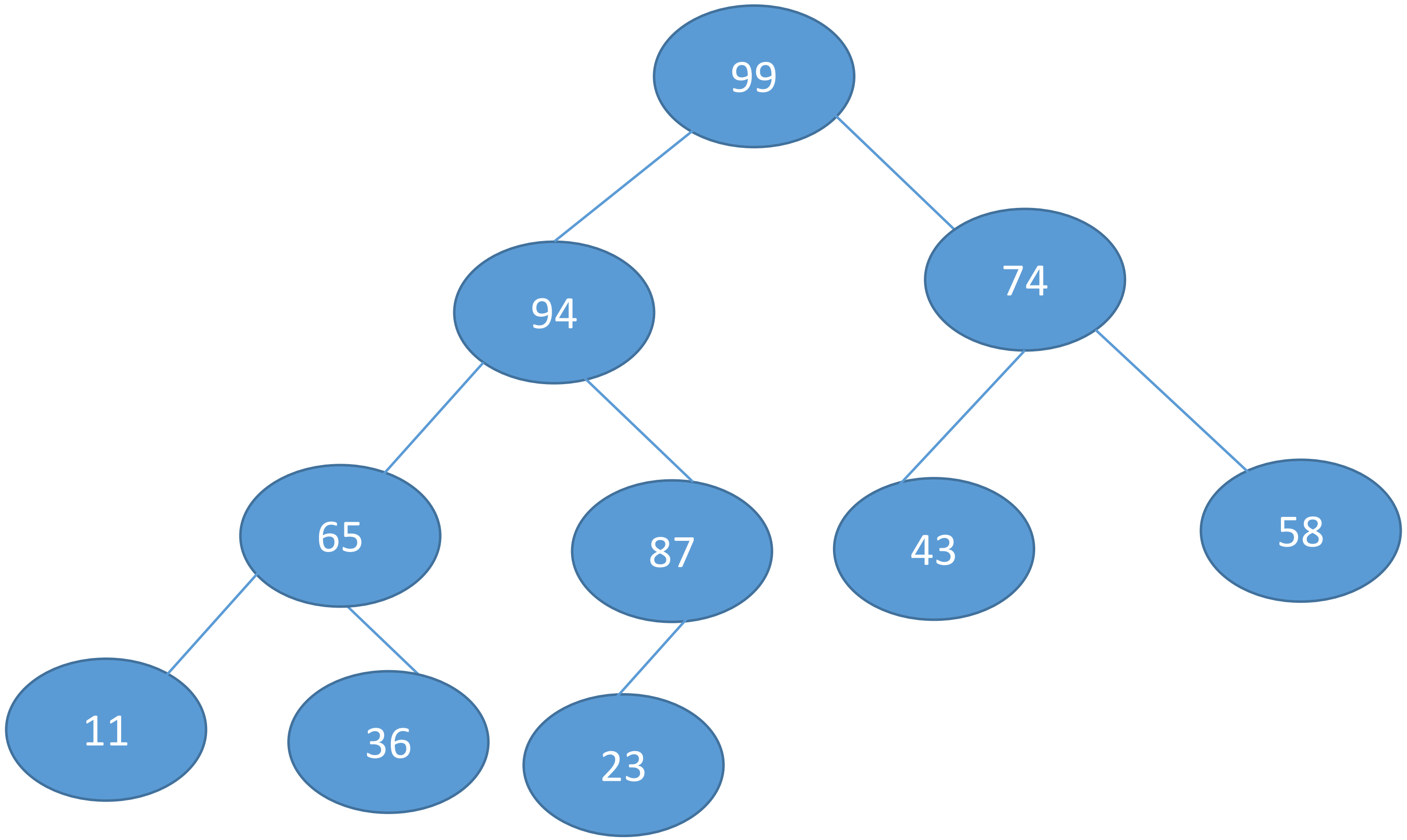


Similarly we get

Max-heap



HEAP SORT

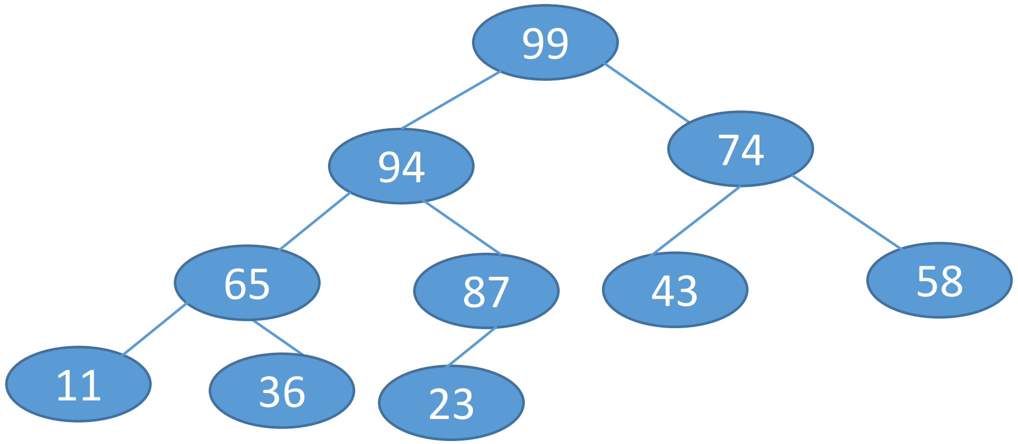


STEP 1;

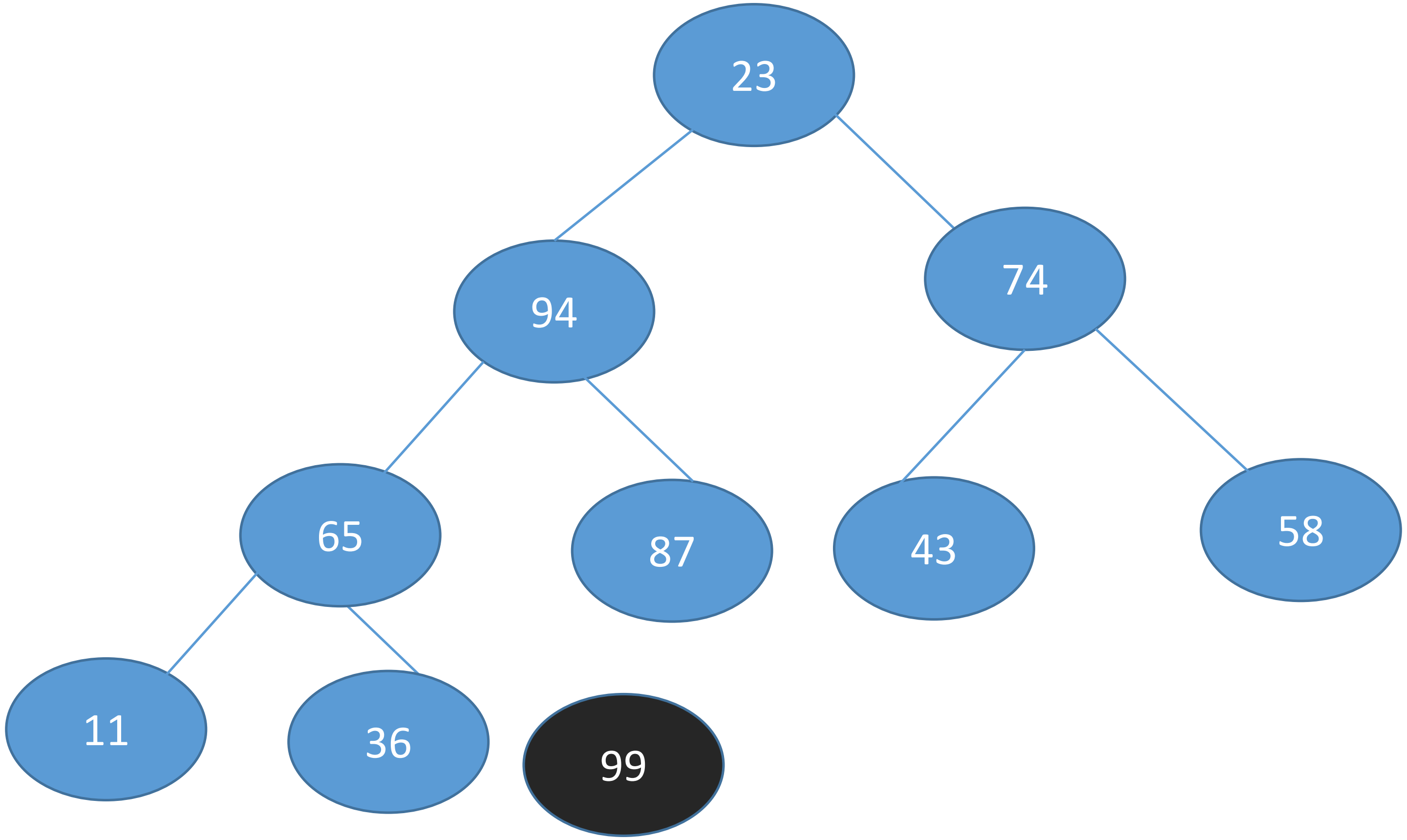
Swap leftmost child and root,

A[1] and A[10]

99	94	74	65	87	43	58	11	36	23
1	2	3	4	5	6	7	8	9	10

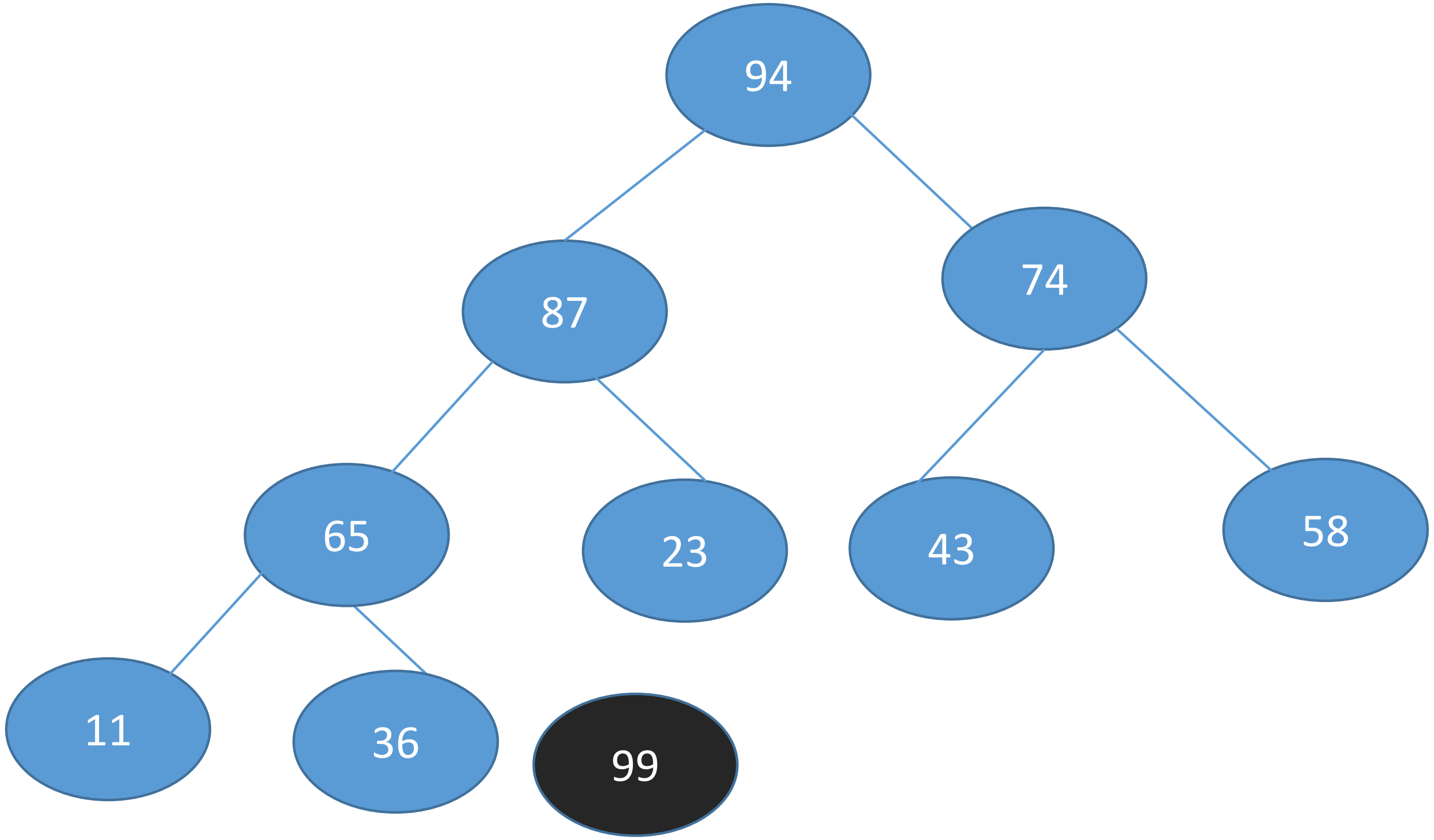


Step 1:



- *Again check max heap Property rest of 99.*
- *Heap property is not maintain so, heapify*

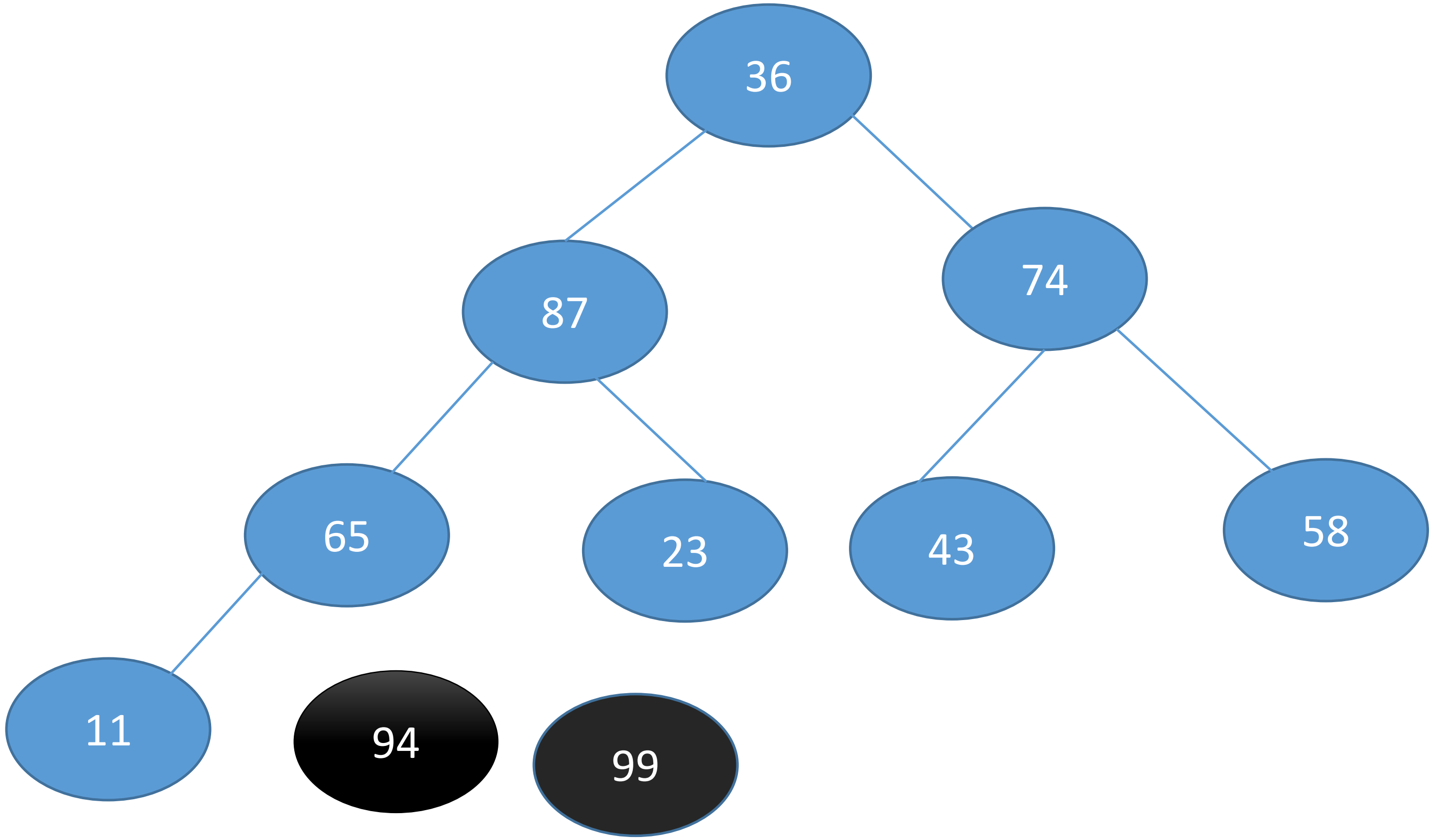
Step 2nd heapify

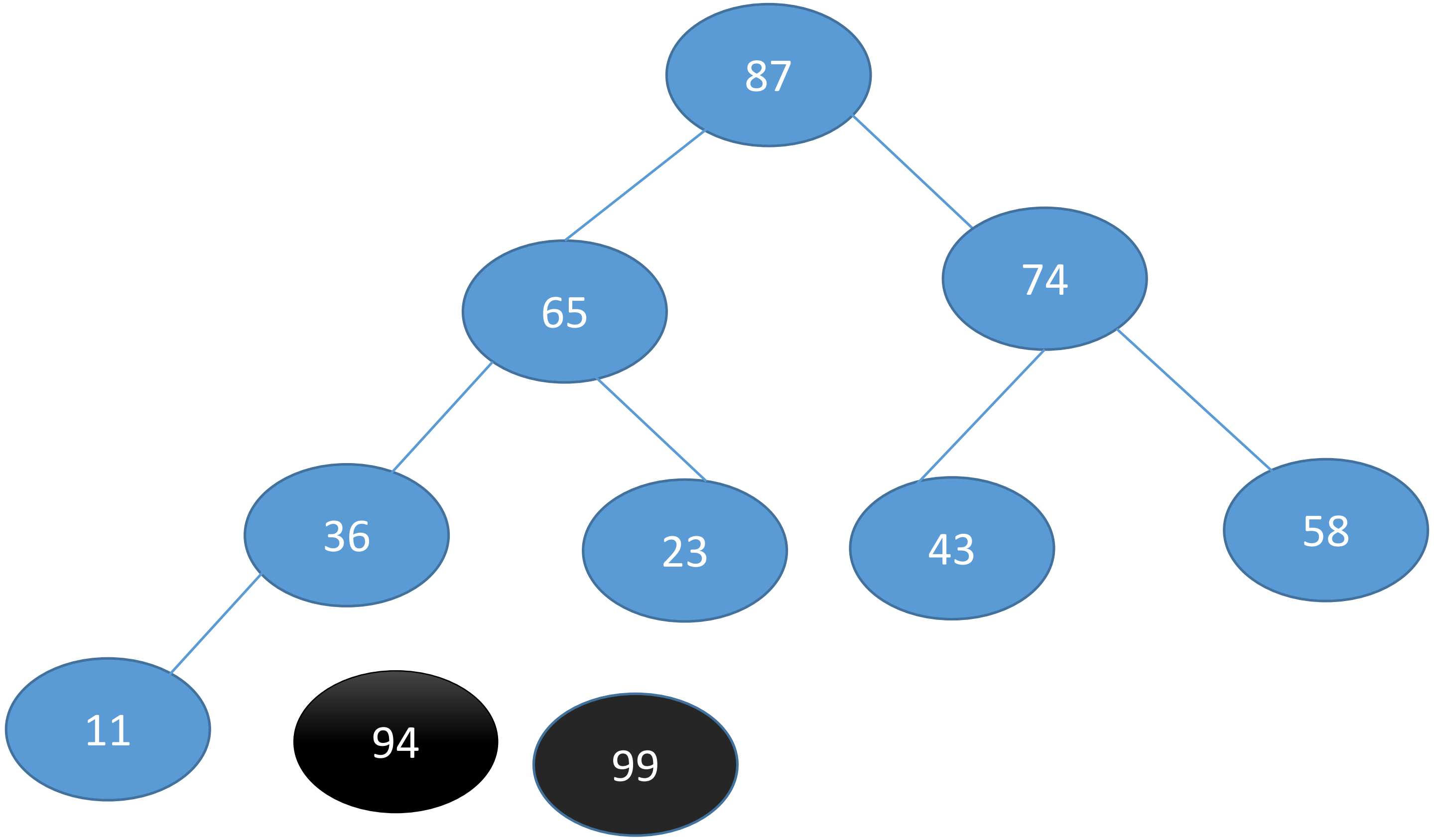


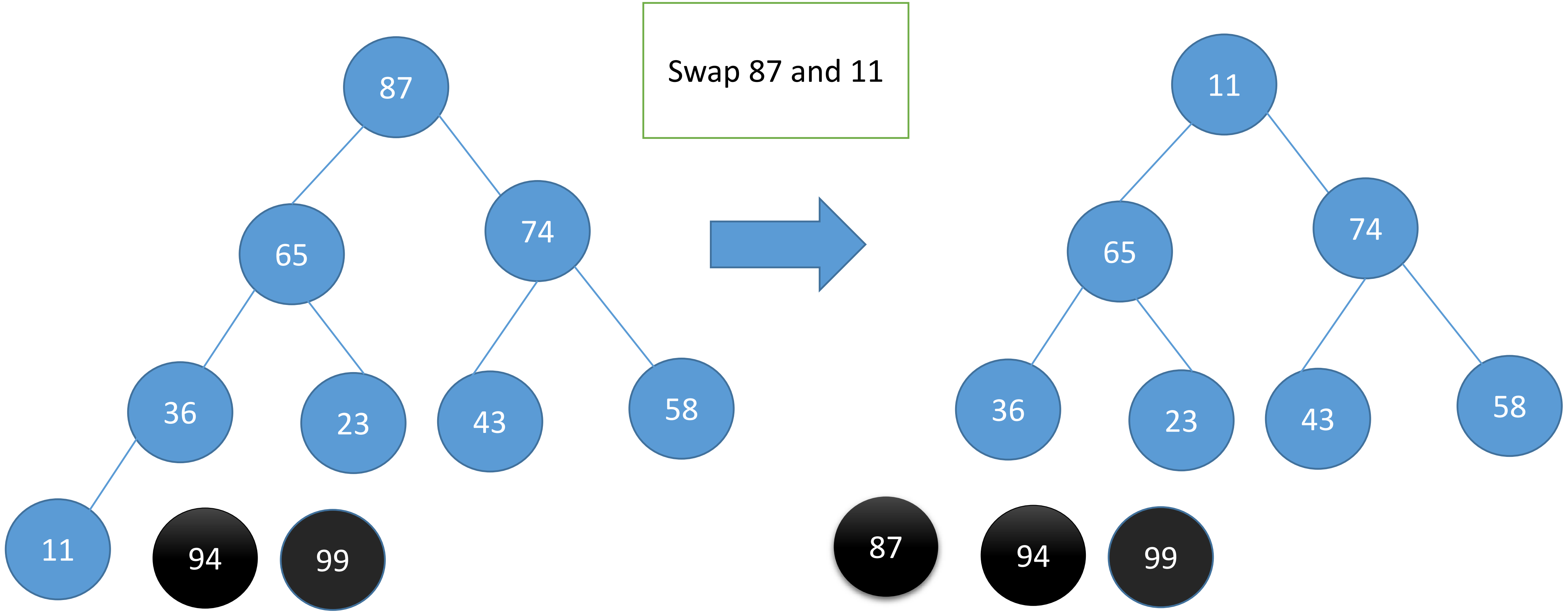
- Again check max-heap proptery
- It maintains max-heap proptery then again swap element pos. 9 with position 1.

Step 3:

Again maintain max –heap property and then swap…..

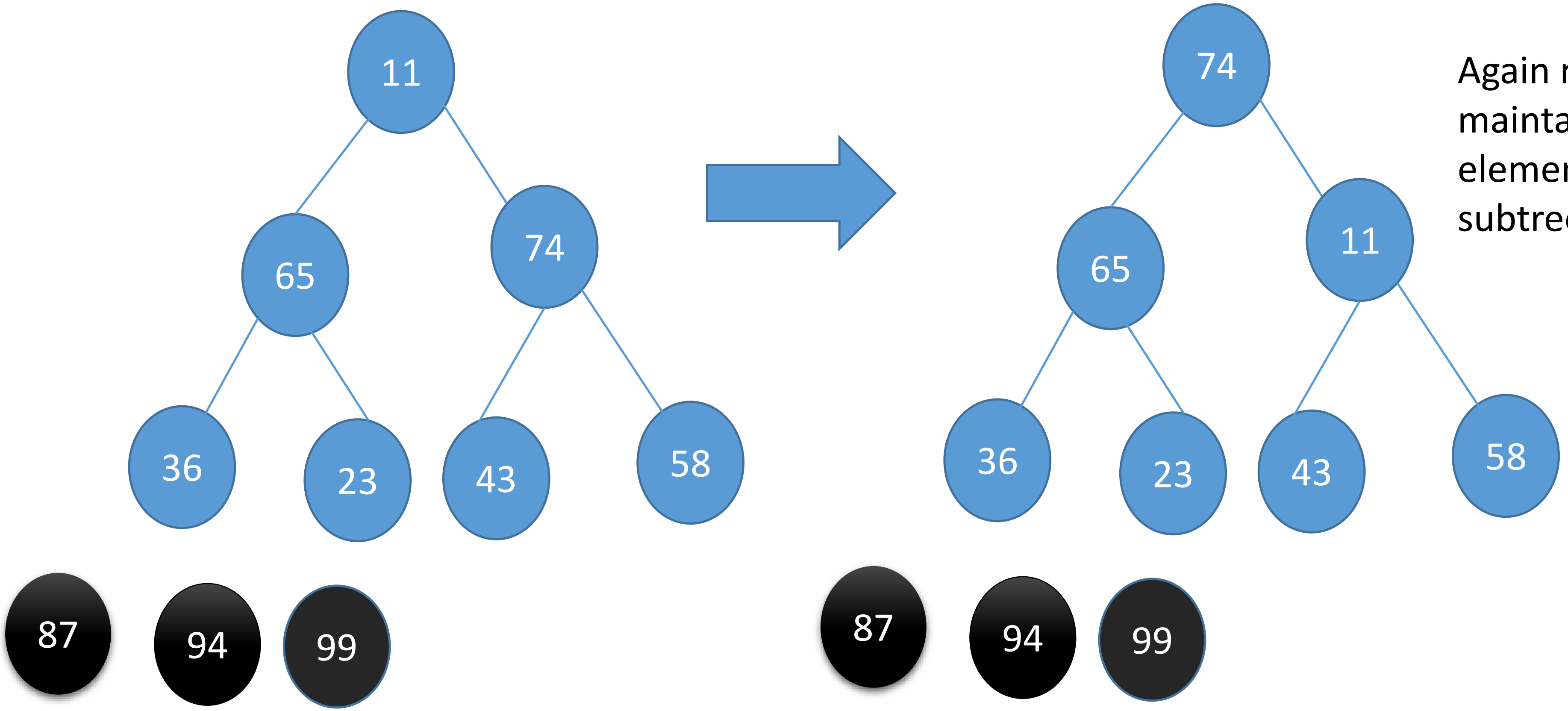



87
65
74
36
23
43
58
11
94
99



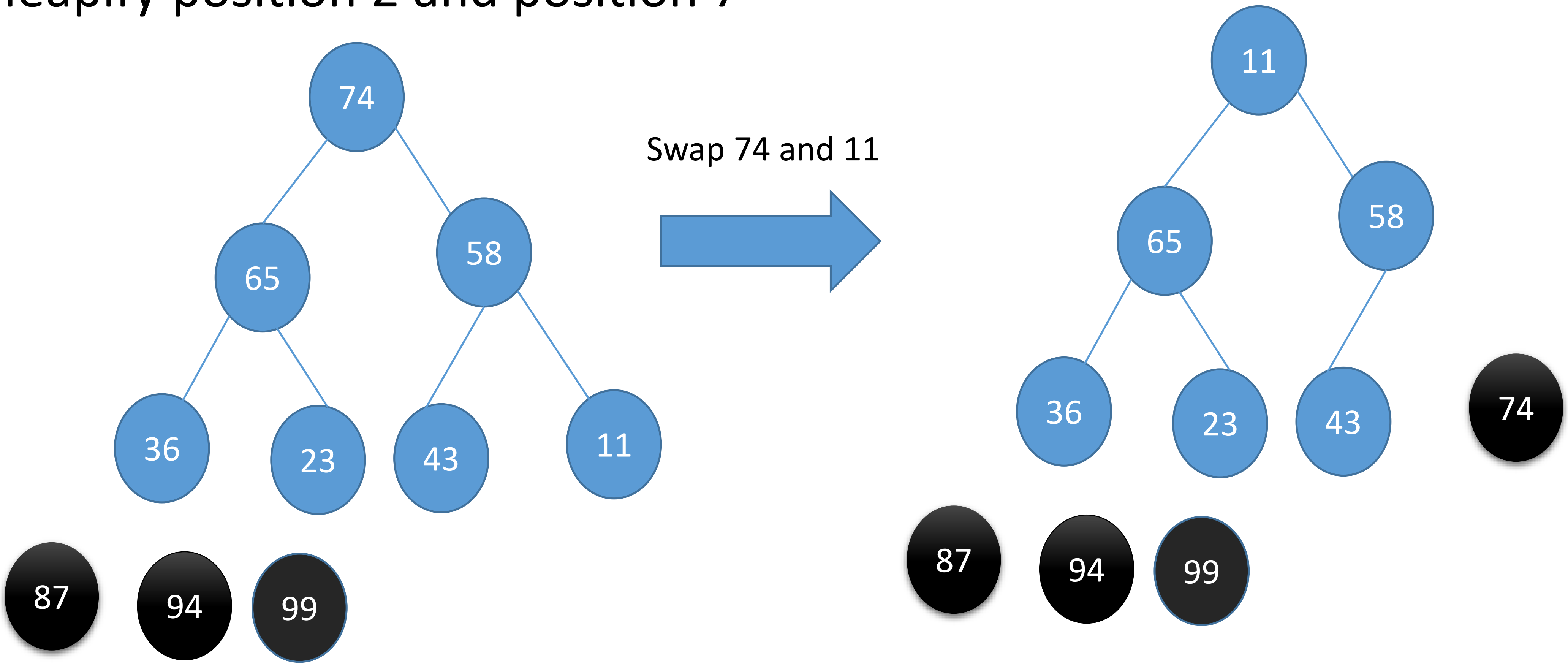
- Again check max-heap property which is not maintain because of root 11 then heapify and maintain max-heap property.

74 is largest among all element so swap 11 with 74



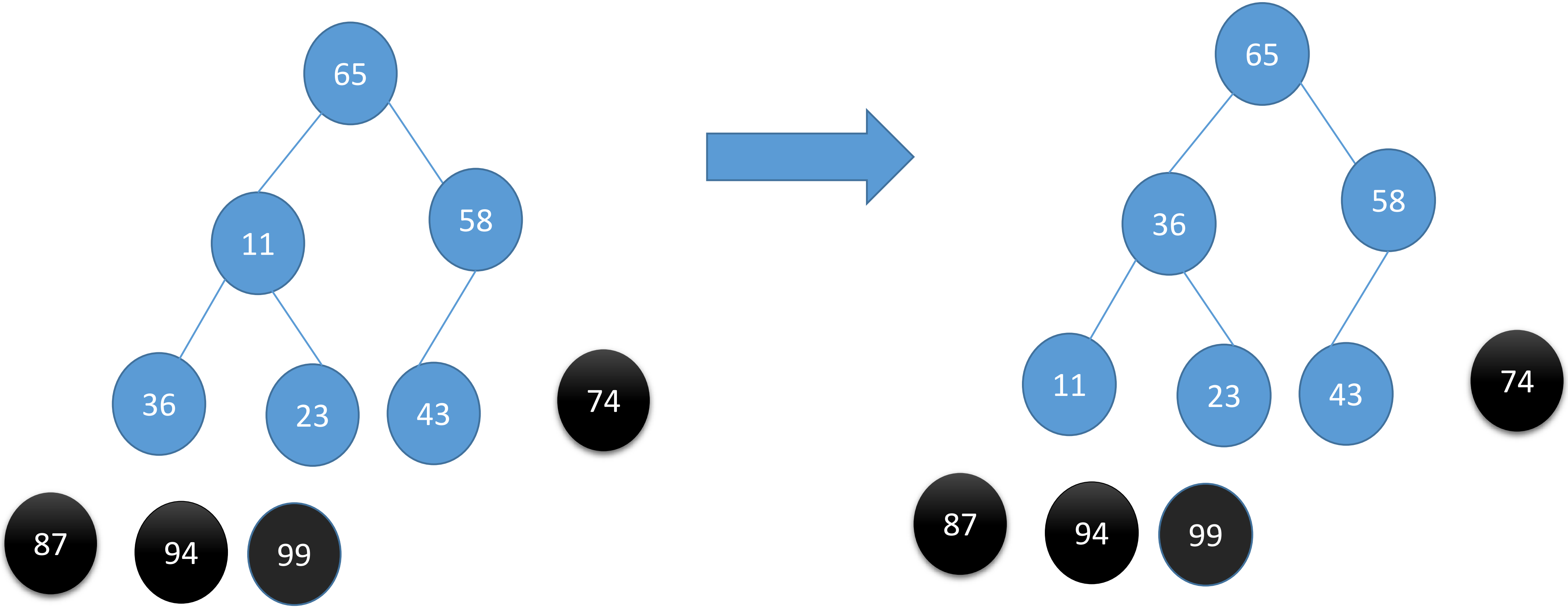
Again max heap property is not maintain so swap 11 with largest element form it's left and right subtree.

Heapify position 2 and position 7



- **Again heapify to maintain max-heap property.**
- **Swap 11 with largest child from left and right subtree**

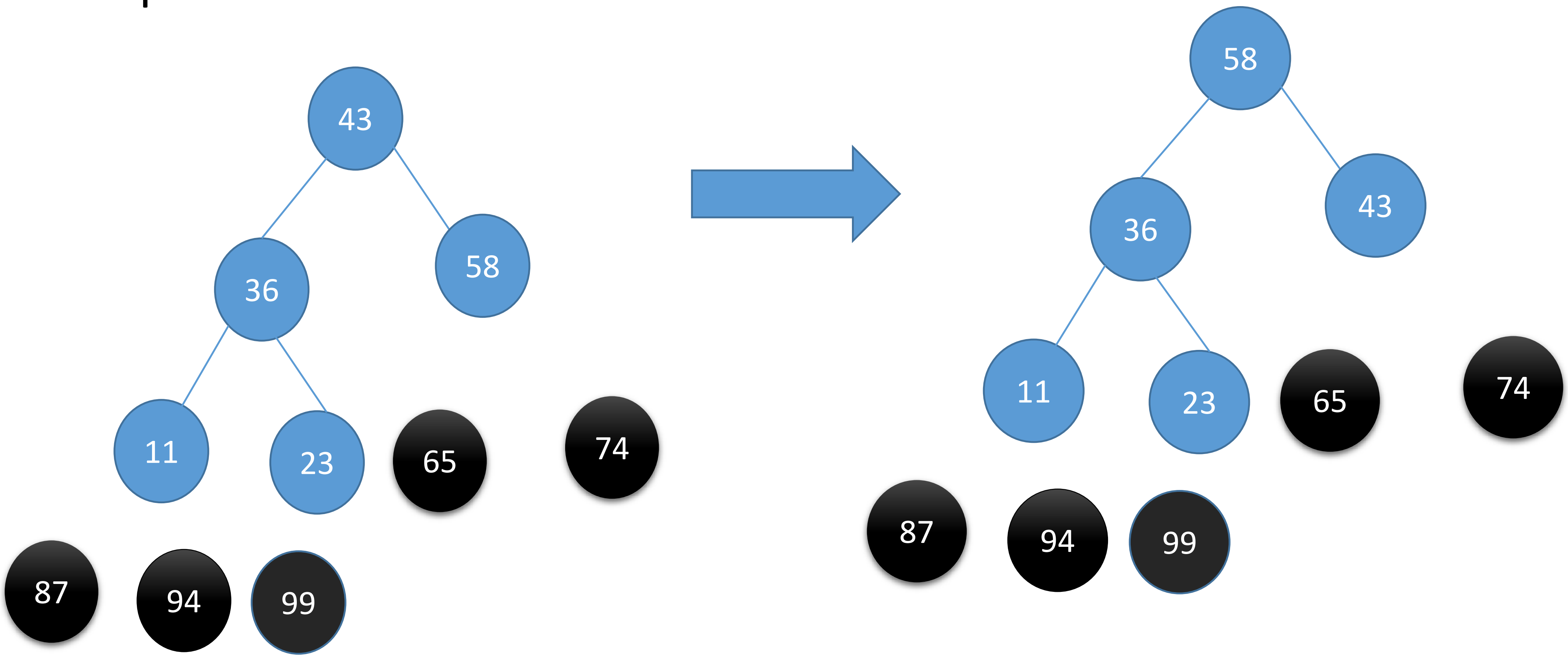
Swap 11 and 65



- Max-heap property is not maintain so, again swap 11 with largest child ie. 36

- Now Max-heap property is maintain
- Again swap 43 and 65

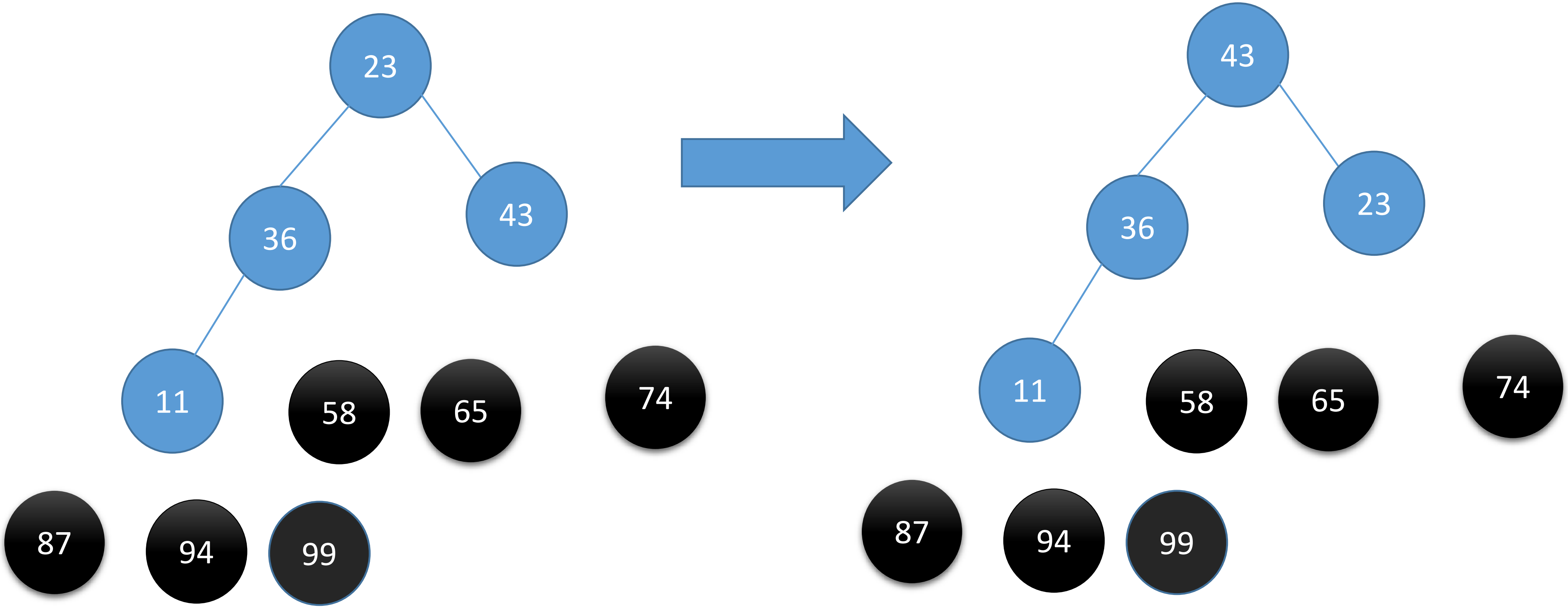
Swap 43 and 65



- Max-heap property is not maintain so, again swap 43 with largest child ie. 58

- Now Max-heap property is maintain
- Again swap 58 and 23

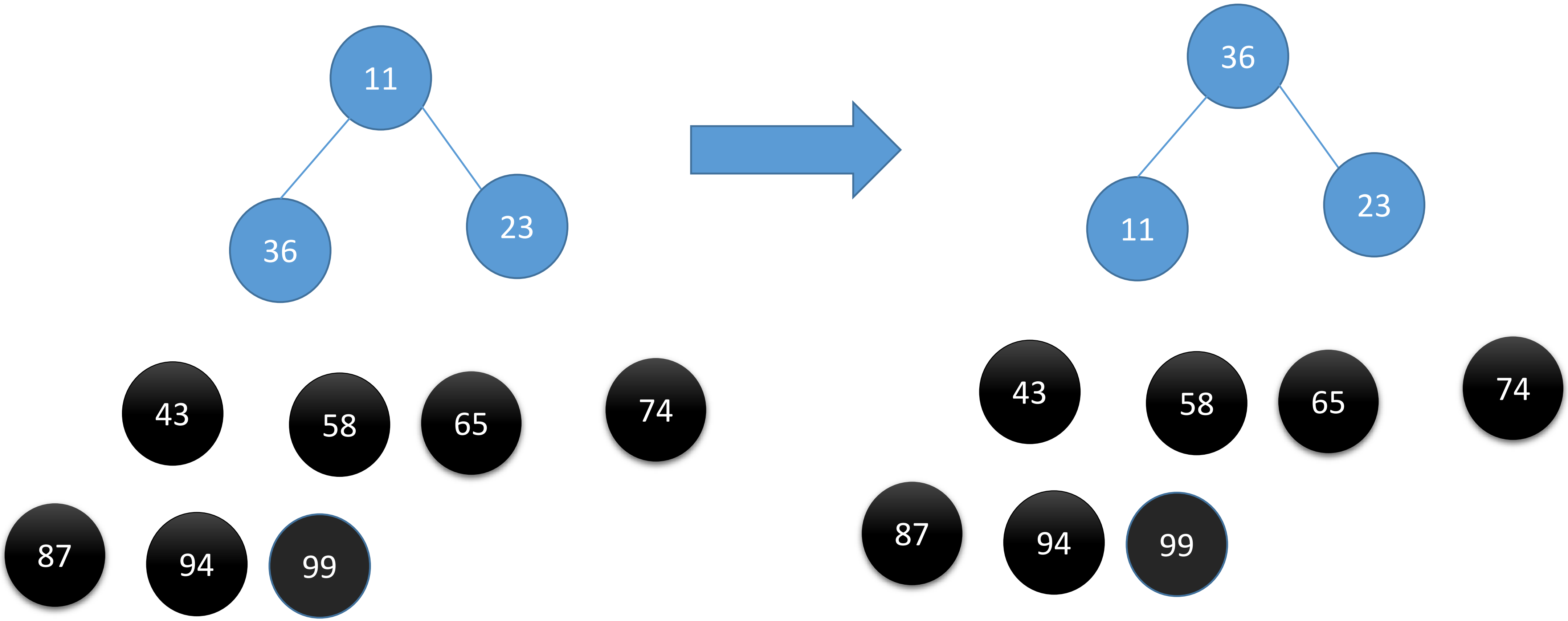
Swap 58 and 23



- Max-heap property is not maintain so, again swap 23 with largest child ie. 43

- Now Max-heap property is maintain
- Again swap 43 and 11

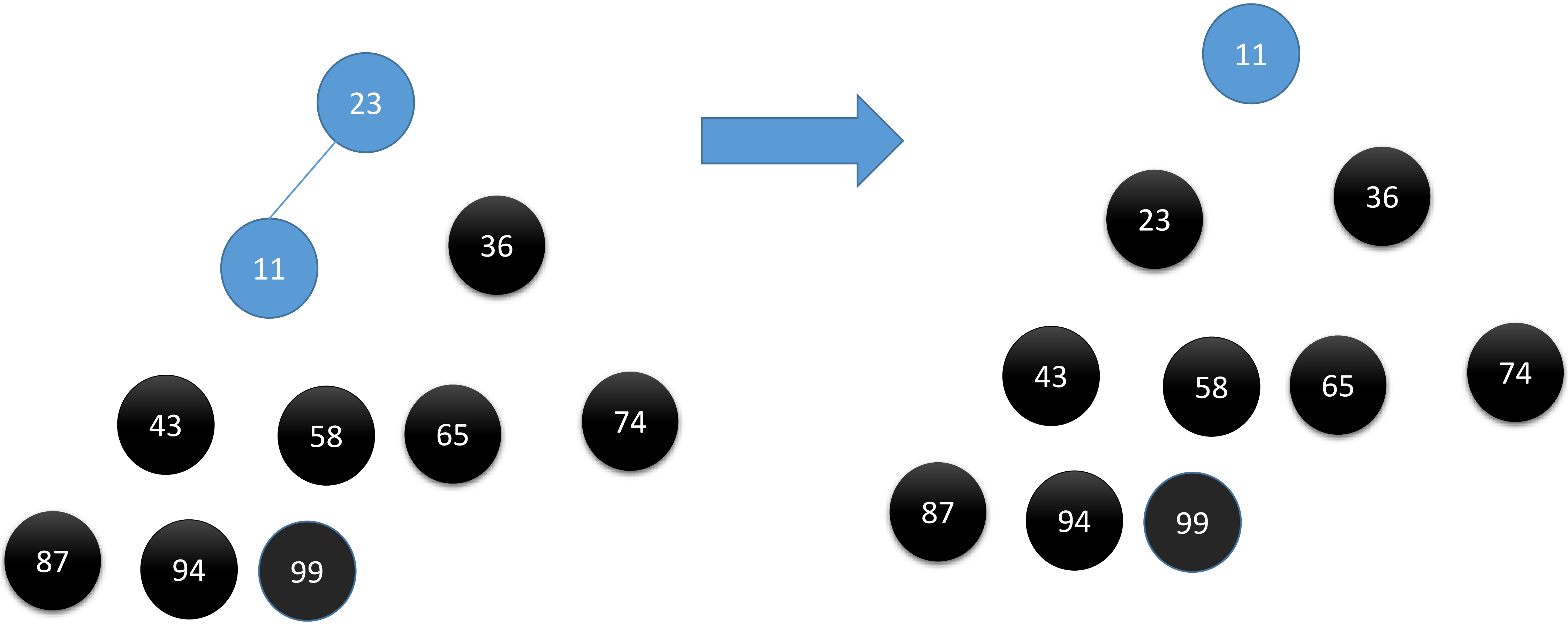
Swap 58 and 23



- Max-heap property is not maintain so, again swap 11 with largest child ie. 36

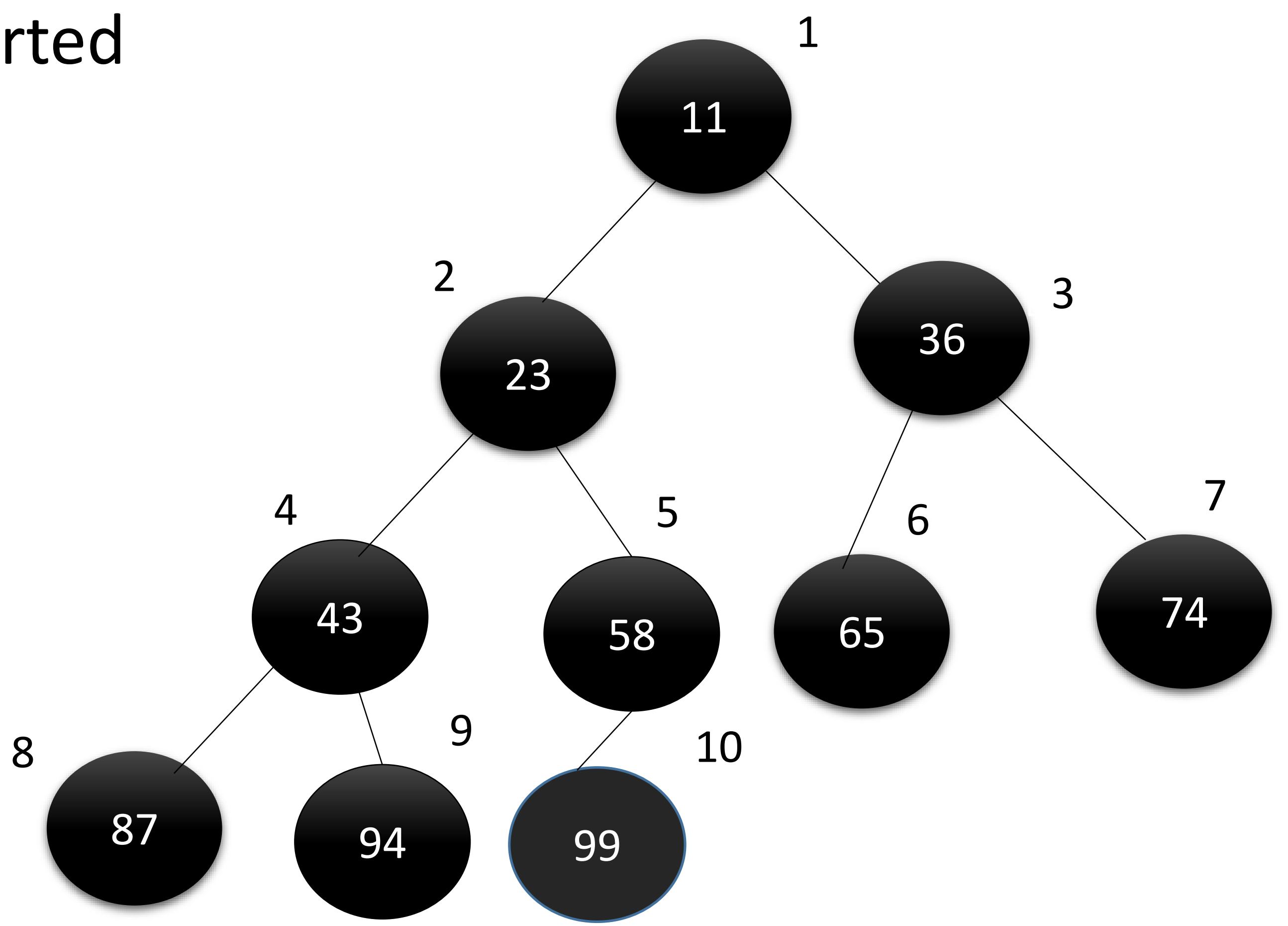
- Now Max-heap property is maintain
- Again swap 36 and 23

Swap 58 and 23



- Max-heap property is maintain so, no need to heapify. Swap 23 and 11.

Finally, Data is in sorted order



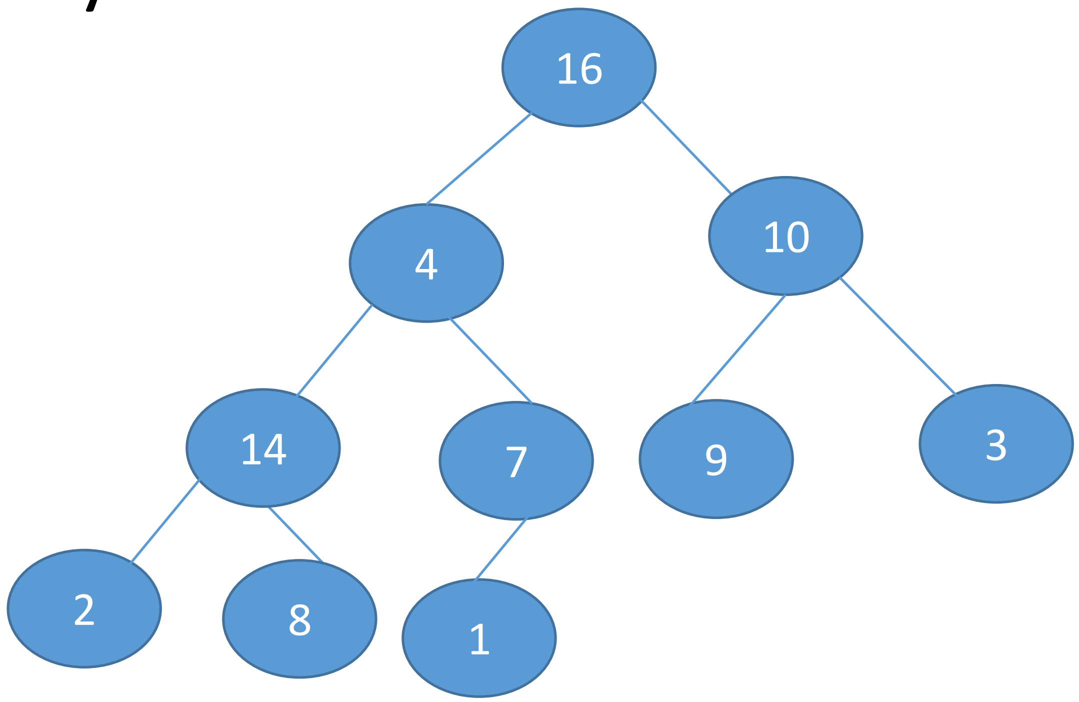
11	23	36	43	58	65	74	87	94	99
1	2	3	4	5	6	7	8	9	10

Max-heap

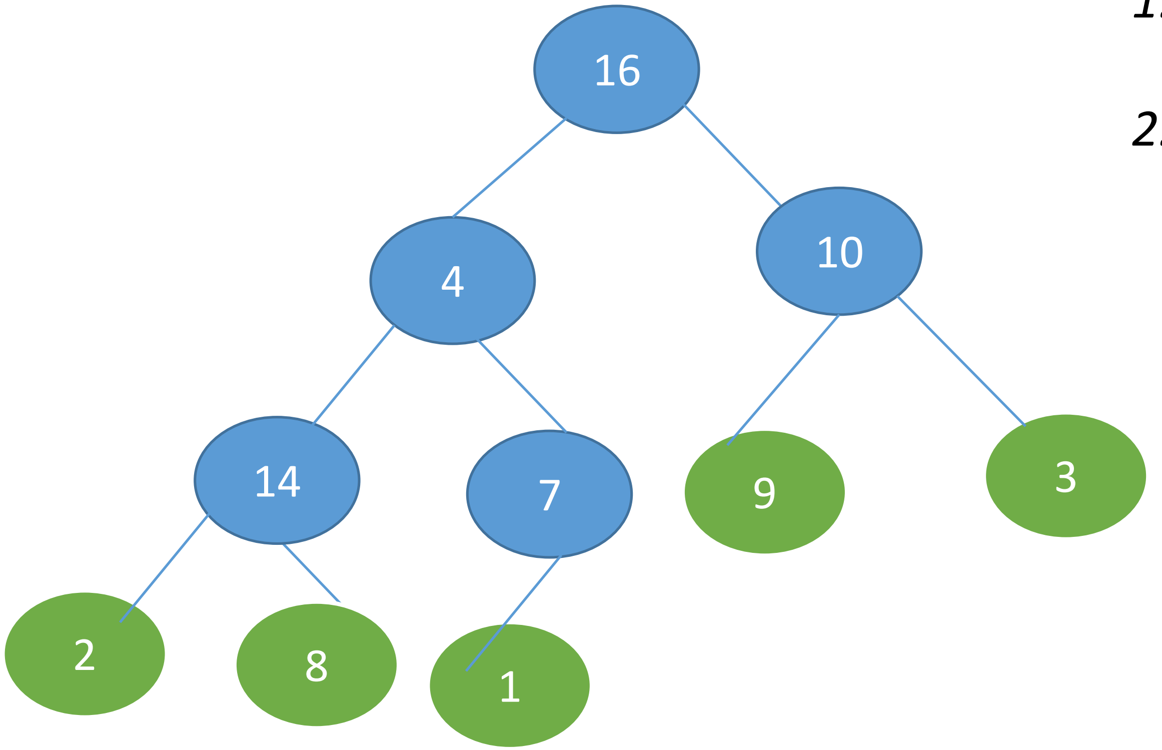
16 4 10 14 7 9 3 2 8 1

-construct almost complete binary tree.

-heapify to make max-heap.



Heapify to build max heap



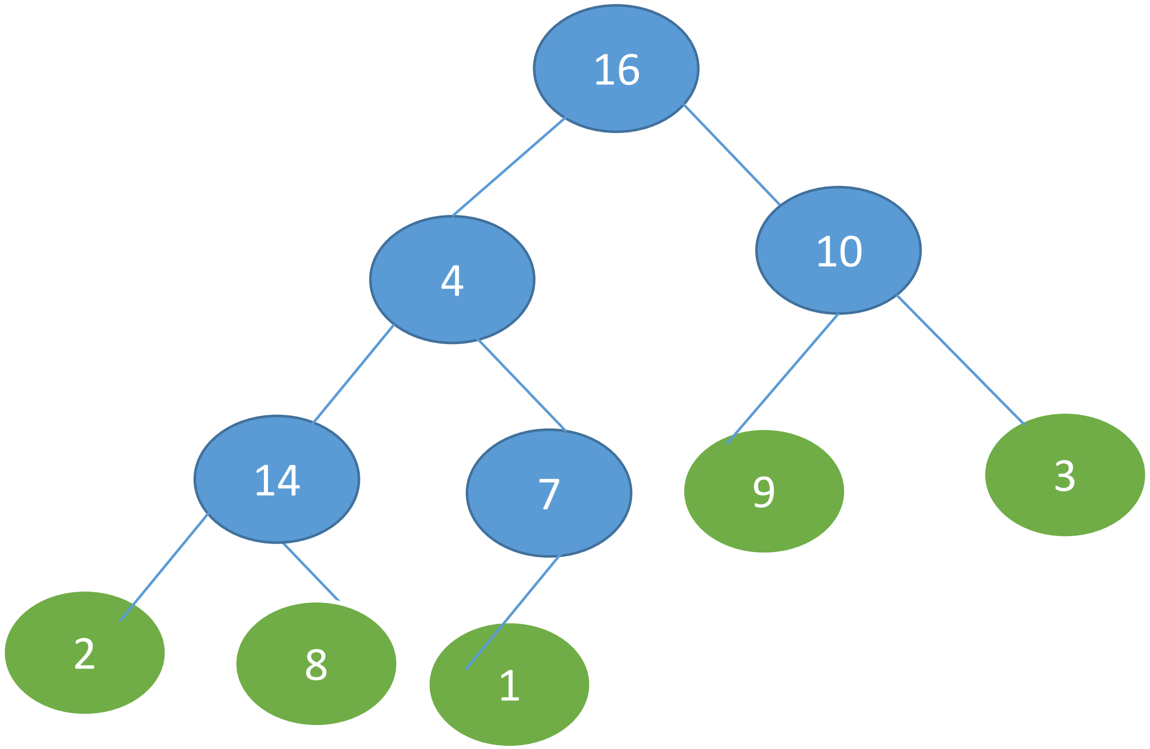
We have to heapify form non-leaf node so,

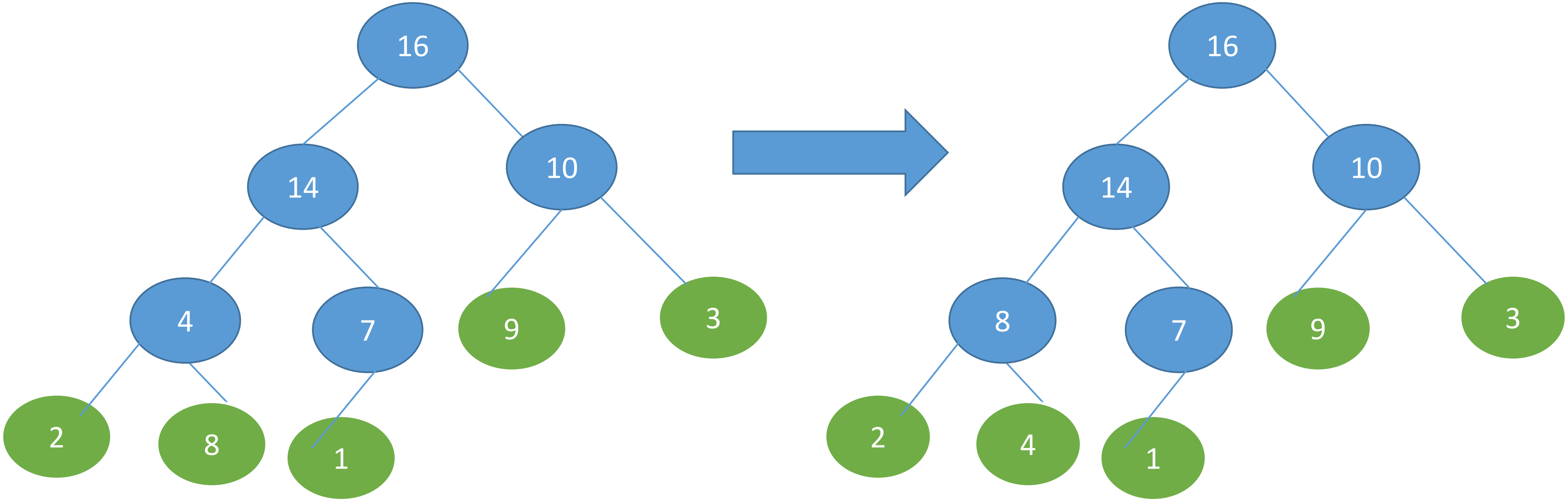
1. *We should find leaves of the tree:*

   ***leaves=floor(n/2)+1 to n** and*

2. Non-leaves node form n/2 to 1

16	**4**	**10**	**14**	**7**	**9**	**3**	**2**	**8**	**1**


16
4
10
14
7
9
3
2
8
1


16
14
10
4
7
9
3
2
8
1
16
14
10
8
7
9
3
2
4
1

Algorithm to construct max-heap/heap

Build-maxHeap(A)

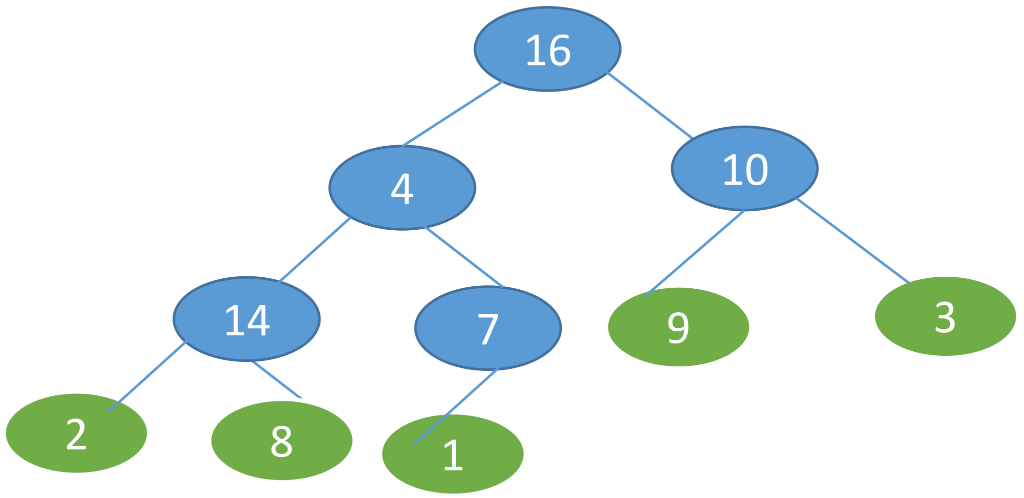
{

HeapSize(A)=length(A)

for(i=floor(length[A]/2) down to 1) // i=floor(n/2)to i>=1, i--

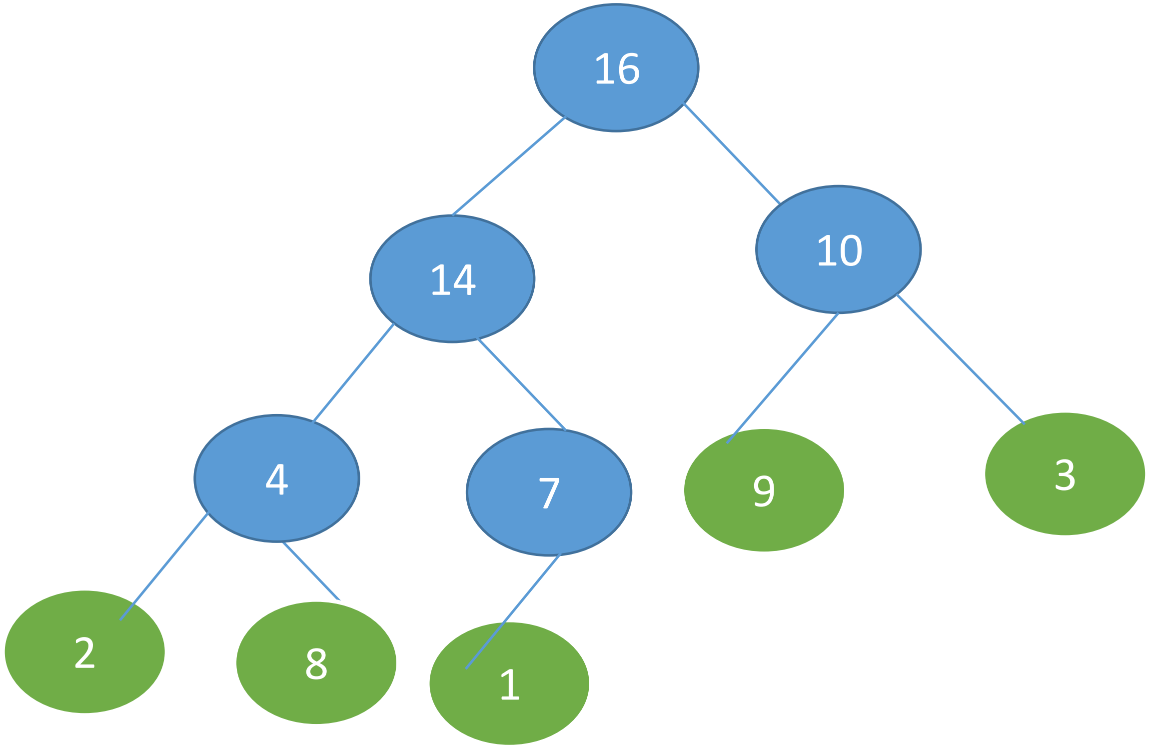
max-Heapify(A, i);

}



```
max-heapify(A, i){
            l=2i;
            r=2i+1;
 if(l<=heapsize(A) and A[l]>A[i])){
            largest= l;
            }
 else
            largest =i;
if(r<=heapsize(a) and A[r]>A[largest]){
            largest =r;
          }
if(largest!= i){
            exchange A[i] <->A[largest];

max-heapify(A, largest);
}}
```



Algorithm for heap sort

Input : an array A[] of size n

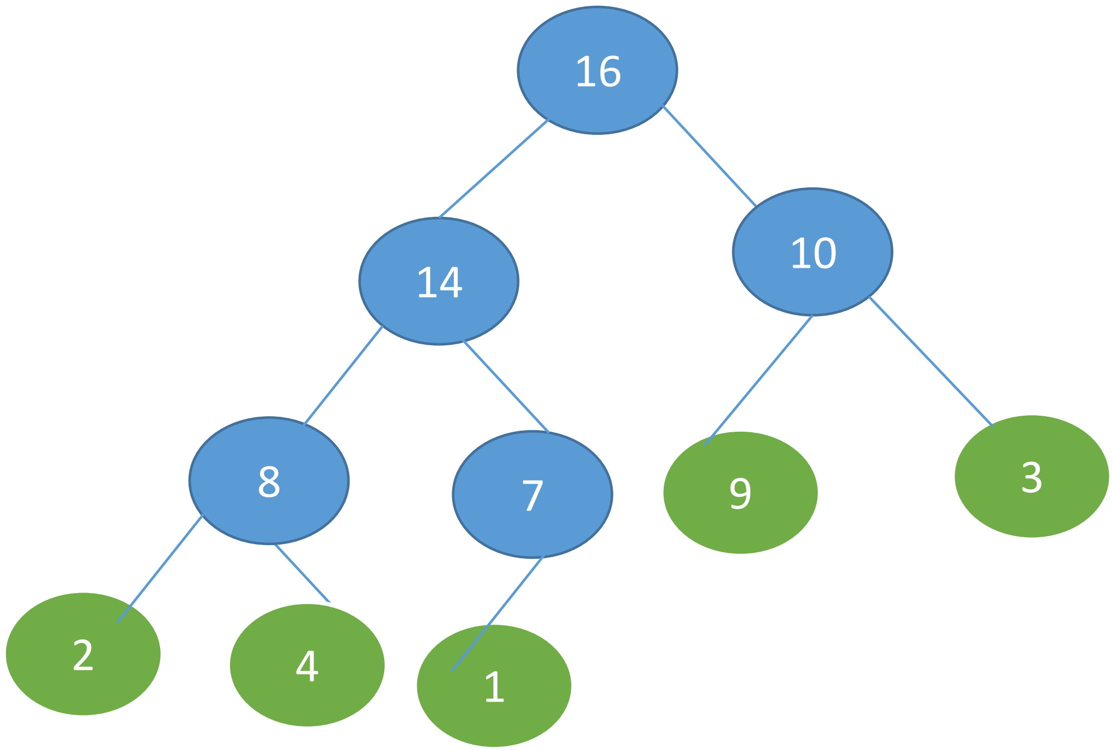
1. Build heap(A)
2. m=n;
3. For(i=m; i>=1; i--){

    swap (A[1], A[m]);

    m=m-1;

    max-heapify(A, 1);

  }



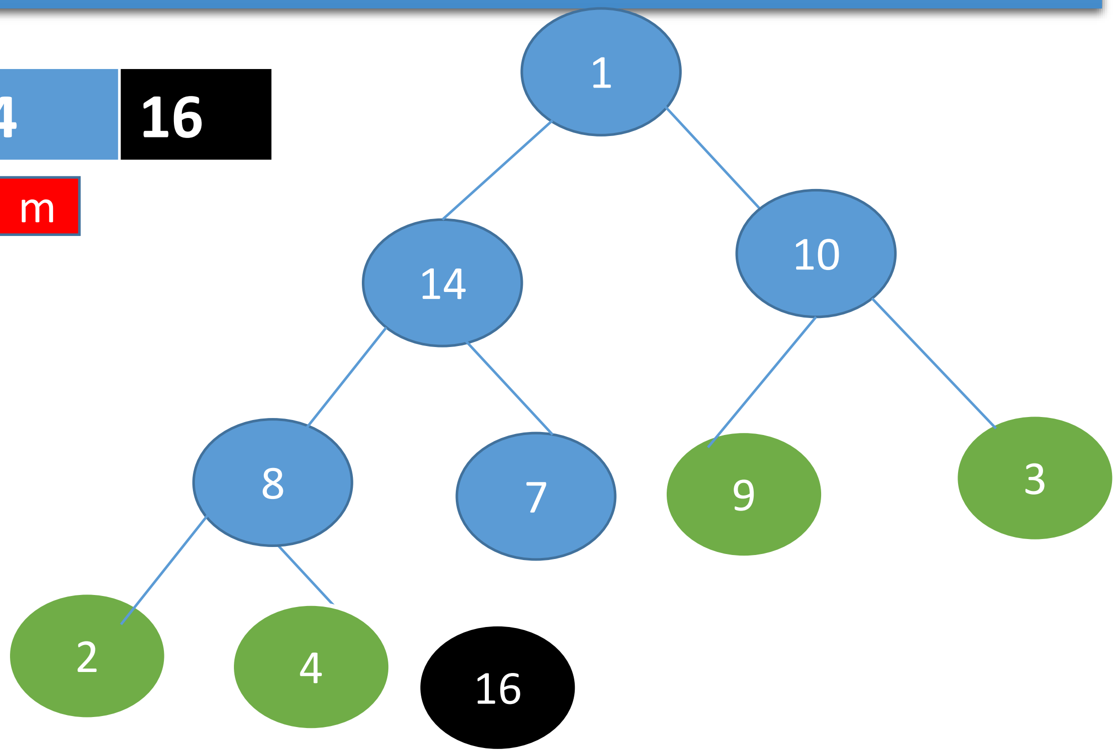
16	14	10	8	7	9	3	2	4	1

1 m

Algorithm for heap sort

1	14	10	8	7	9	3	2	4	16
1								m	

Again heapify from 1to m



To be continue..........

Time Complexity

- Build heap takes O(n) times
- For loop executes n times : o(n) times
- And heapify operation in for loop is: O(logn) times
- Total time complexity is : o(n)+o(n) {(logn)}

  =O(nlogn)

H/w

- Write a program to implement heap sort.