Lecture No.25-26

Data Structures and Algorithms

Dr. Islam Zada

HEAP

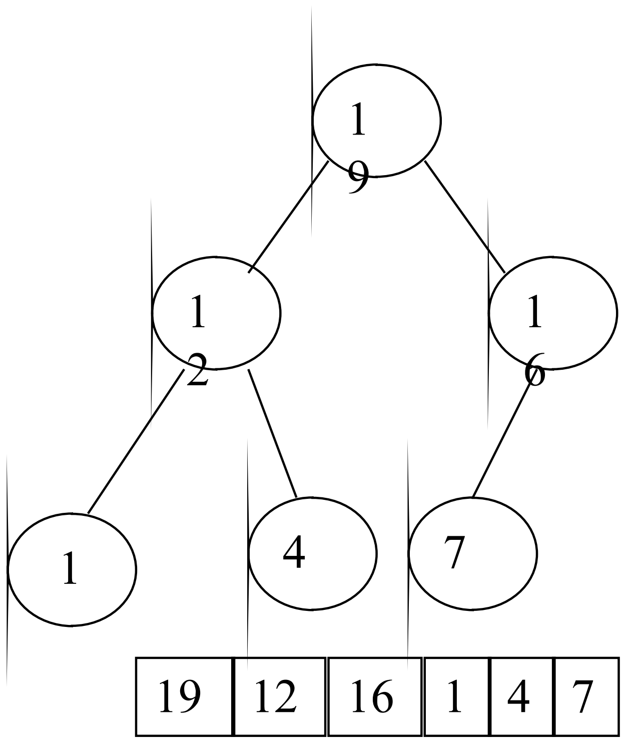
A heap is a data structure that stores a collection of objects (with keys), and has the following properties:

- Complete Binary tree
- Heap Order

It is implemented as an array where each node in the tree corresponds to an element of the array.

HEAP

- The binary heap data structures is an array that can be viewed as a complete binary tree. Each node of the binary tree corresponds to an element of the array. The array is completely filled on all levels except possibly lowest.



Array A

HEAP

- The root of the tree A[1] and given index *i* of a node, the indices of its parent, left child and right child can be computed

```
PARENT ($i$)
        return floor($i$/2)
LEFT ($i$)
        return $2i$
RIGHT ($i$)
        return $2i + 1$
```

HEAP ORDER PROPERTY

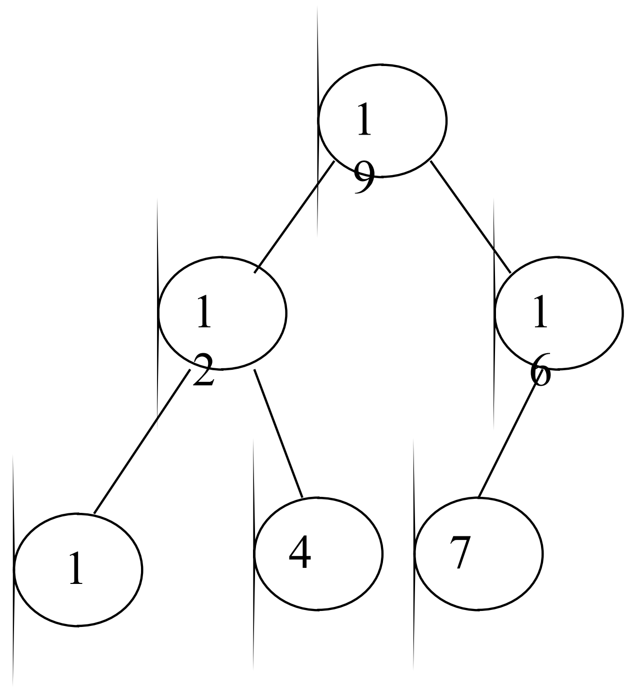
- For every node *v*, other than the root, the key stored in *v* is greater or equal (smaller or equal for max heap) than the key stored in the parent of *v*.

- In this case the maximum value is stored in the root

DEFINITION

- Max Heap
  - Store data in ascending order
  - Has property of
    $A[Parent(i)] \geq A[i]$
- Min Heap
  - Store data in descending order
  - Has property of
    $A[Parent(i)] \leq A[i]$

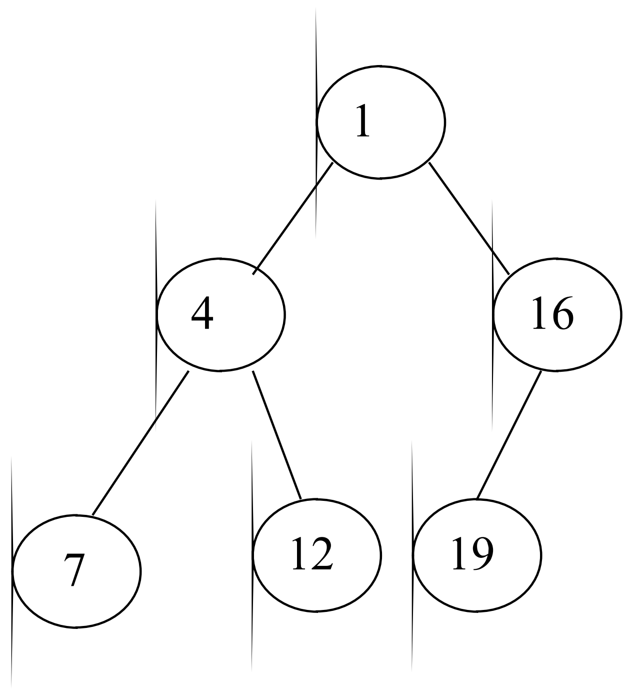
MAX HEAP EXAMPLE



19	12	16	1	4	7

Array A

MIN HEAP EXAMPLE



1	4	16	7	12	19

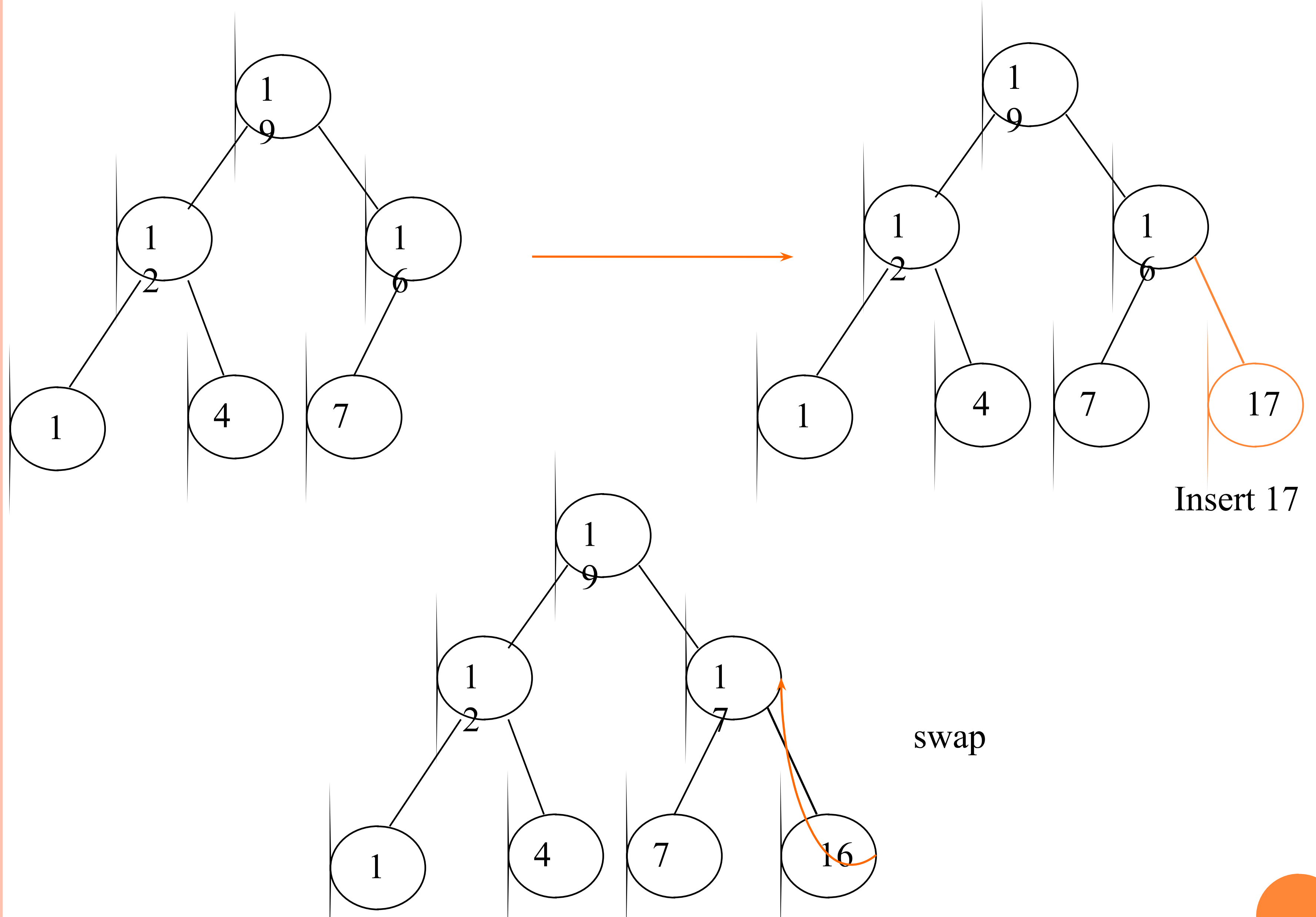
Array A

INSERTION

- Algorithm
  1. Add the new element to the next available position at the lowest level
  2. Restore the max-heap property if violated
     - General strategy is percolate up (or bubble up): if the parent of the element is smaller than the element, then interchange the parent and child.

     OR

     Restore the min-heap property if violated
     - General strategy is percolate up (or bubble up): if the parent of the element is larger than the element, then interchange the parent and child.



Percolate up to maintain the heap property

DELETION

- Delete max
  - Copy the last number to the root (overwrite the maximum element stored there).
  - Restore the max heap property by percolate down.

- Delete min
  - Copy the last number to the root (overwrite the minimum element stored there).
  - Restore the min heap property by percolate down.

HEAP SORT

A sorting algorithm that works by first organizing the data to be sorted into a special type of binary tree called a heap

PROCEDURES ON HEAP

- Heapify
- Build Heap
- Heap Sort

HEAPIFY

- Heapify picks the largest child key and compare it to the parent key. If parent key is larger than heapify quits, otherwise it swaps the parent key with the largest child key. So that the parent is now becomes larger than its children.

```
Heapify(A, i)
{
      l □ left(i)
      r □ right(i)
      if l <= heapsize[A] and A[l] > A[i]
         then largest □l
         else largest □ i
      if r <= heapsize[A] and A[r] > A[largest]
         then largest □ r
      if largest != i
         then swap A[i] □□ A[largest]
            Heapify(A, largest)
}
```

BUILD HEAP

- We can use the procedure 'Heapify' in a bottom-up fashion to convert an array A[1 . . *n*] into a heap. Since the elements in the subarray A[*n*/2 +1 . . . *n*] are all leaves, the procedure BUILD_HEAP goes through the remaining nodes of the tree and runs 'Heapify' on each one. The bottom-up order of processing node guarantees that the subtree rooted at children are heap before 'Heapify' is run at their parent.

```
Buildheap(A)
{
    heapsize[A] □length[A]
    for i □|length[A]/2  //down to 1
        do Heapify(A, i)
}
```

HEAP SORT ALGORITHM

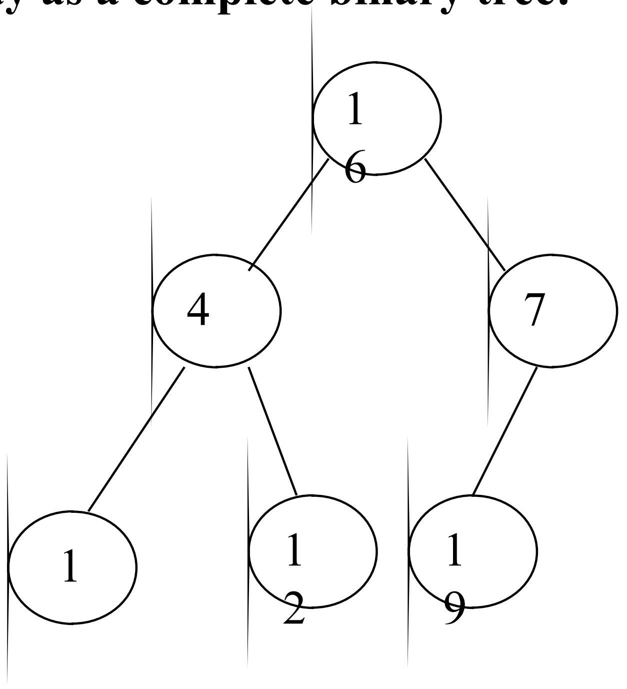
- The heap sort algorithm starts by using procedure BUILD-HEAP to build a heap on the input array A[1 . . *n*]. Since the maximum element of the array stored at the root *A*[1], it can be put into its correct final position by exchanging it with *A*[*n*] (the last element in *A*). If we now discard node n from the heap than the remaining elements can be made into heap. Note that the new element at the root may violate the heap property. All that is needed to restore the heap property.

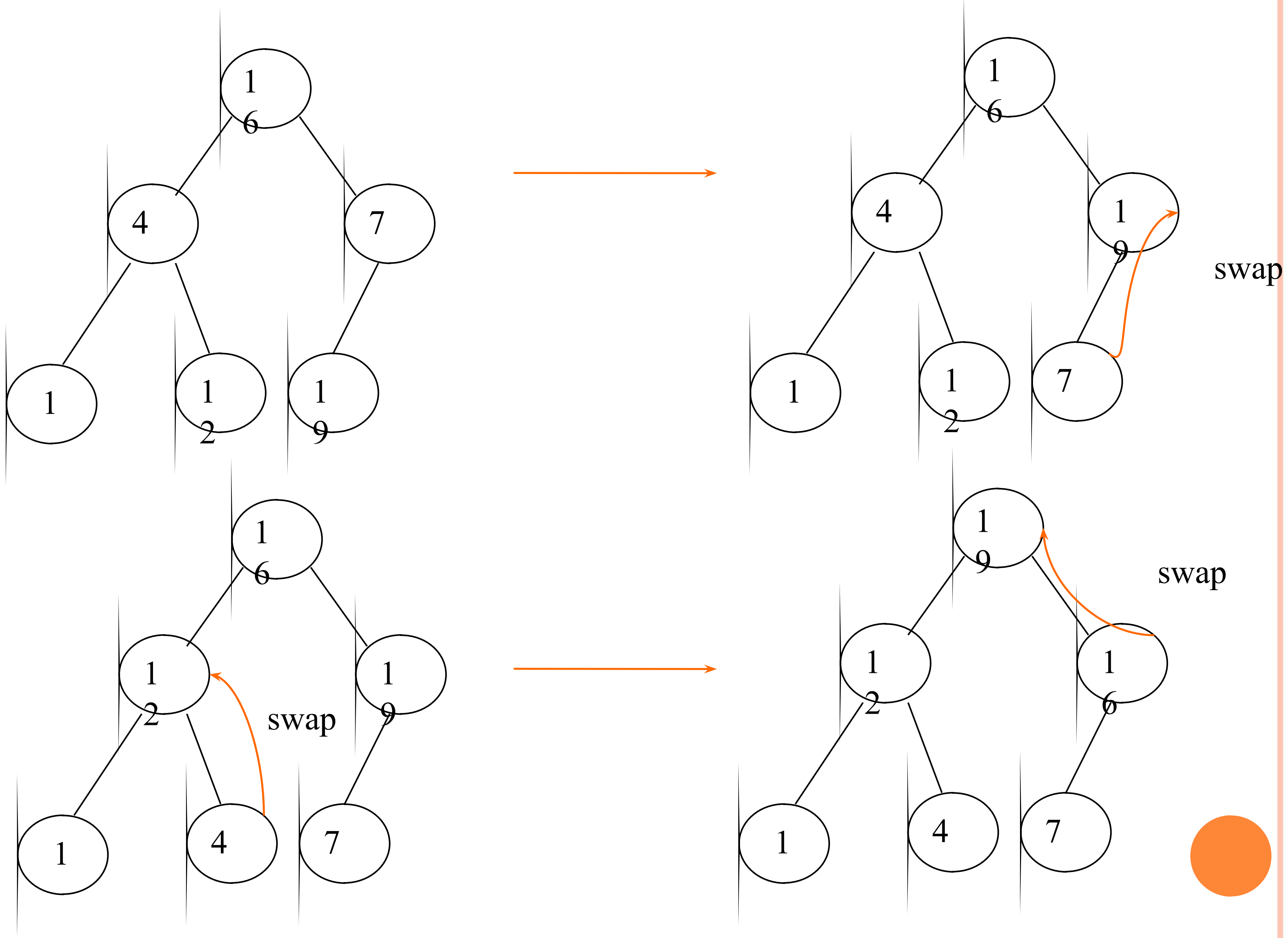
```
Heapsort(A)
{
      Buildheap(A)
      for i □ length[A] //down to 2
         do swap A[1] □□ A[i]
         heapsize[A] □ heapsize[A] - 1
         Heapify(A, 1)
}
```

Example: Convert the following array to a heap

16	4	7	1	12	19

Picture **the array as a complete binary tree:**



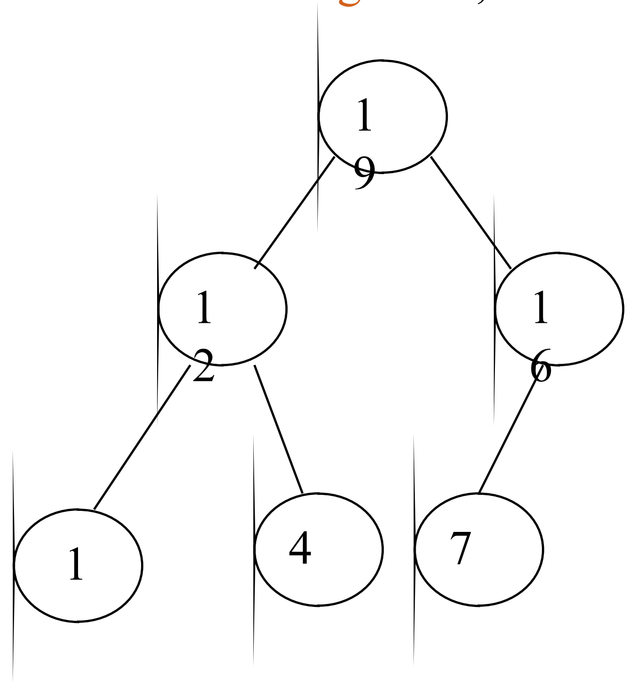


swap
swap
swap

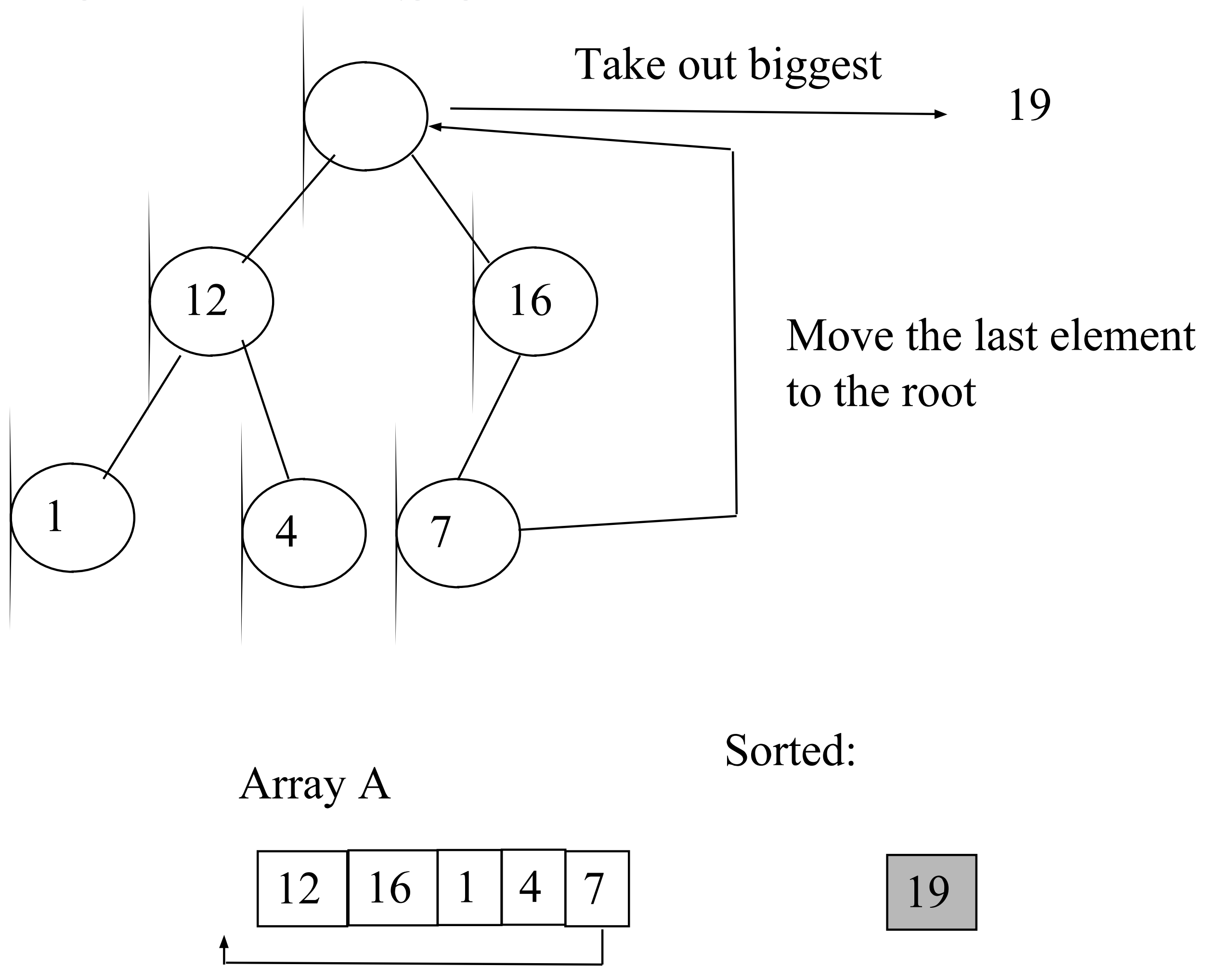
HEAP SORT

- The heapsort algorithm consists of two phases:
  - build a heap from an arbitrary array
  - use the heap to sort the data

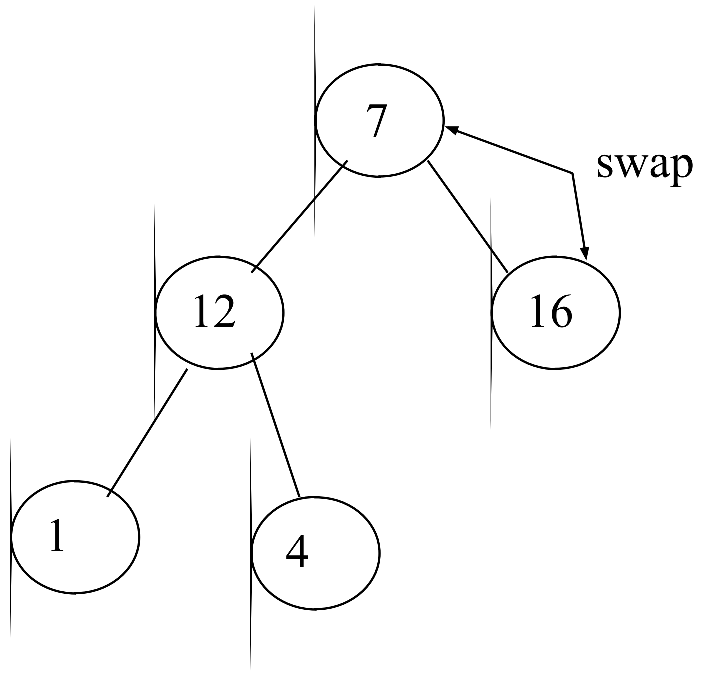
- To sort the elements in the decreasing order, use a min heap
- To sort the elements in the increasing order, use a max heap



EXAMPLE OF HEAP SORT



HEAPIFY()

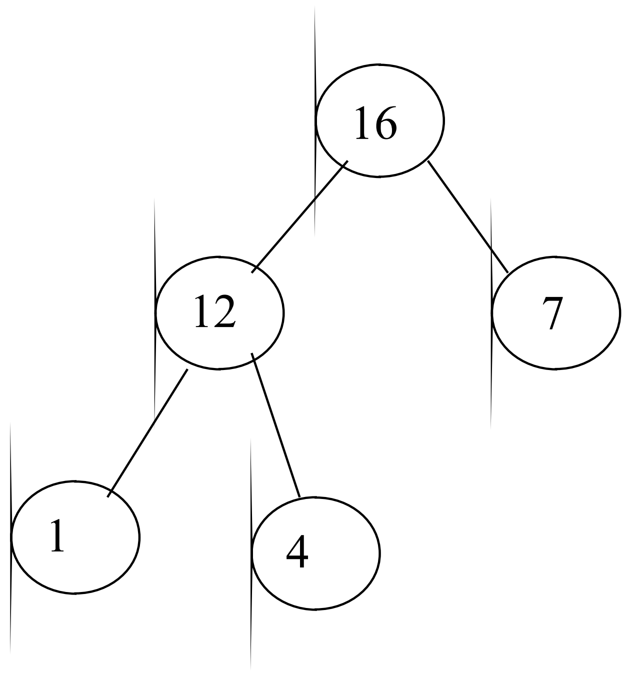


Array A

7	12	16	1	4

Sorted:

19

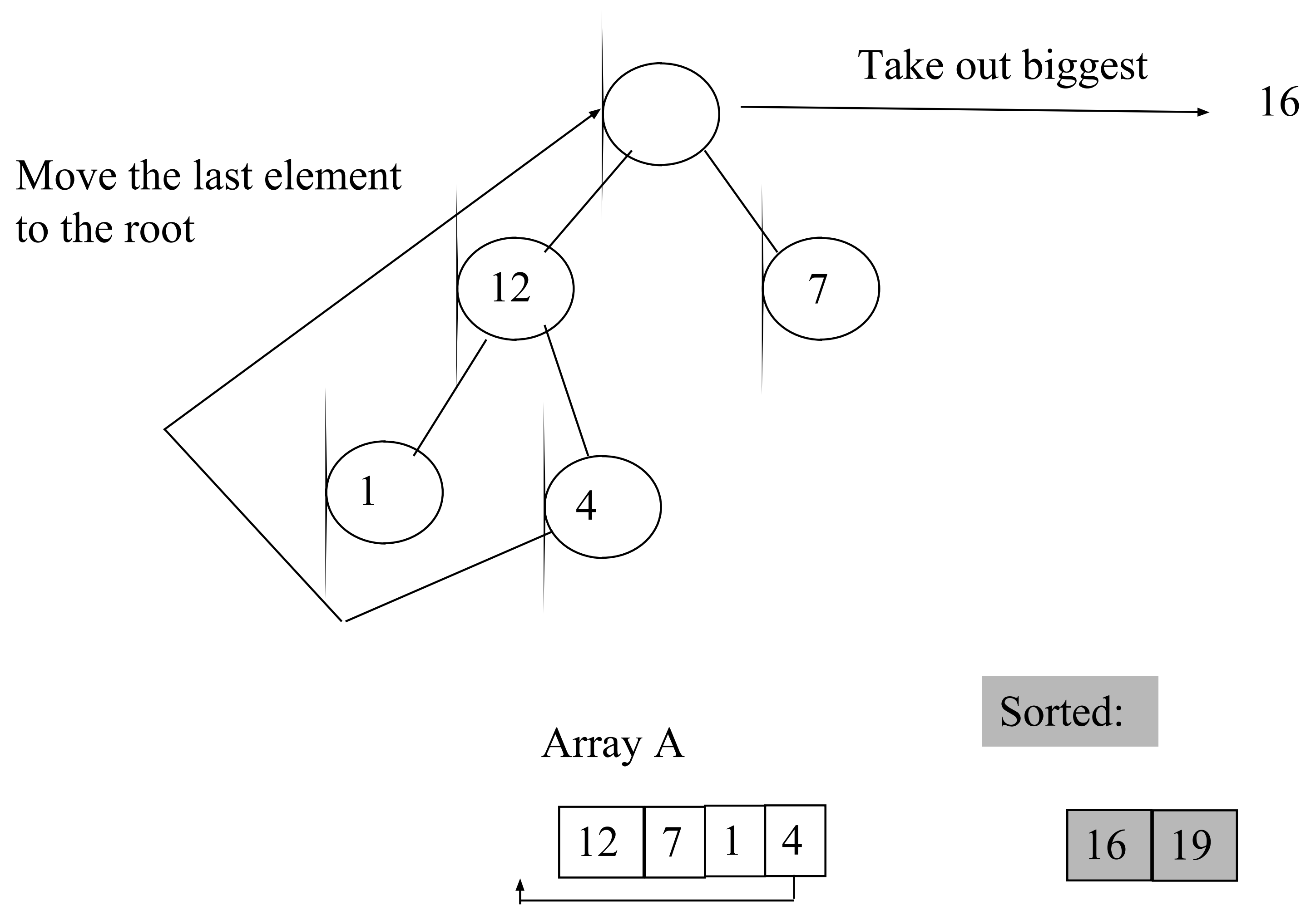


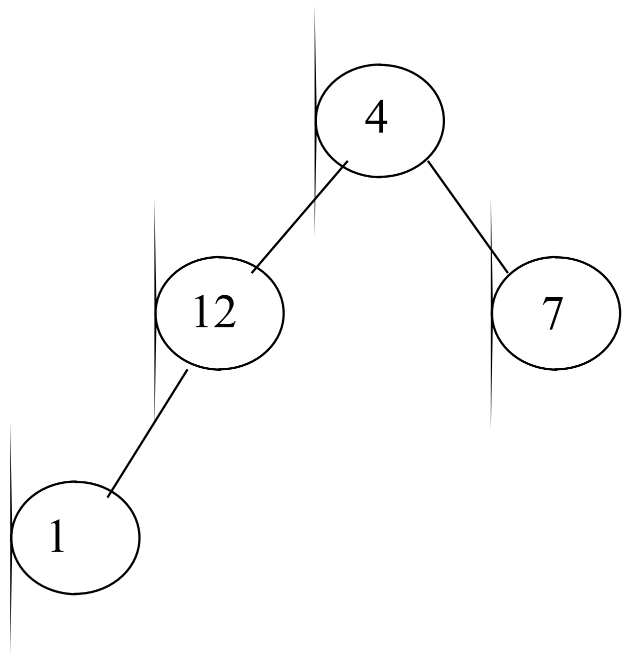
Array A

16	12	7	1	4

Sorted:

19


Move the last element
to the root
Take out biggest
16
12
7
1
4
Array A
12
7
1
4
Sorted:
16
19



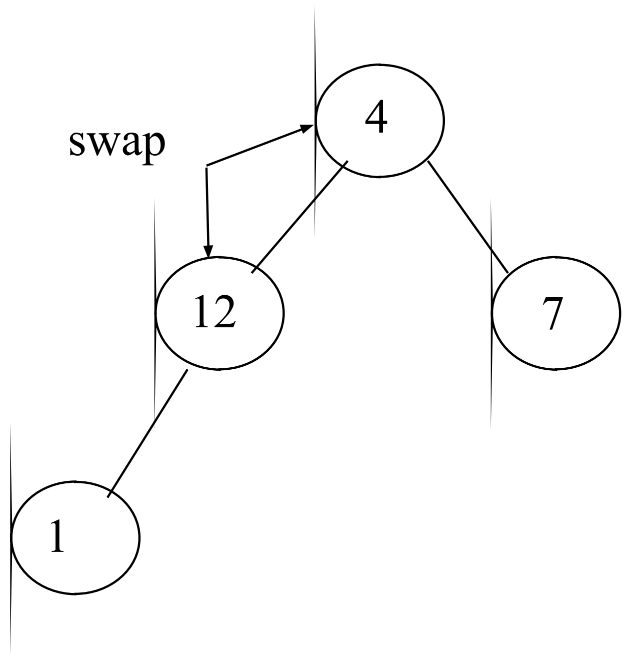
Array A

4	12	7	1

Sorted:

16	19

HEAPIFY()

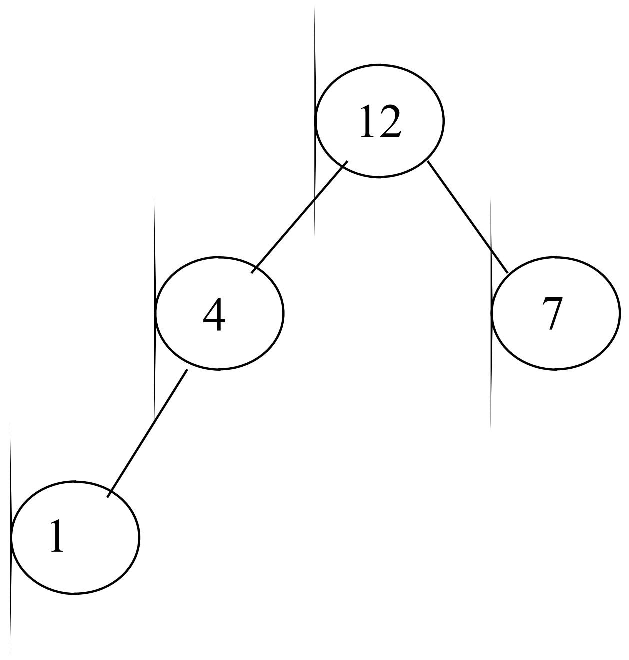


Array A

4	12	7	1

Sorted:

16	19

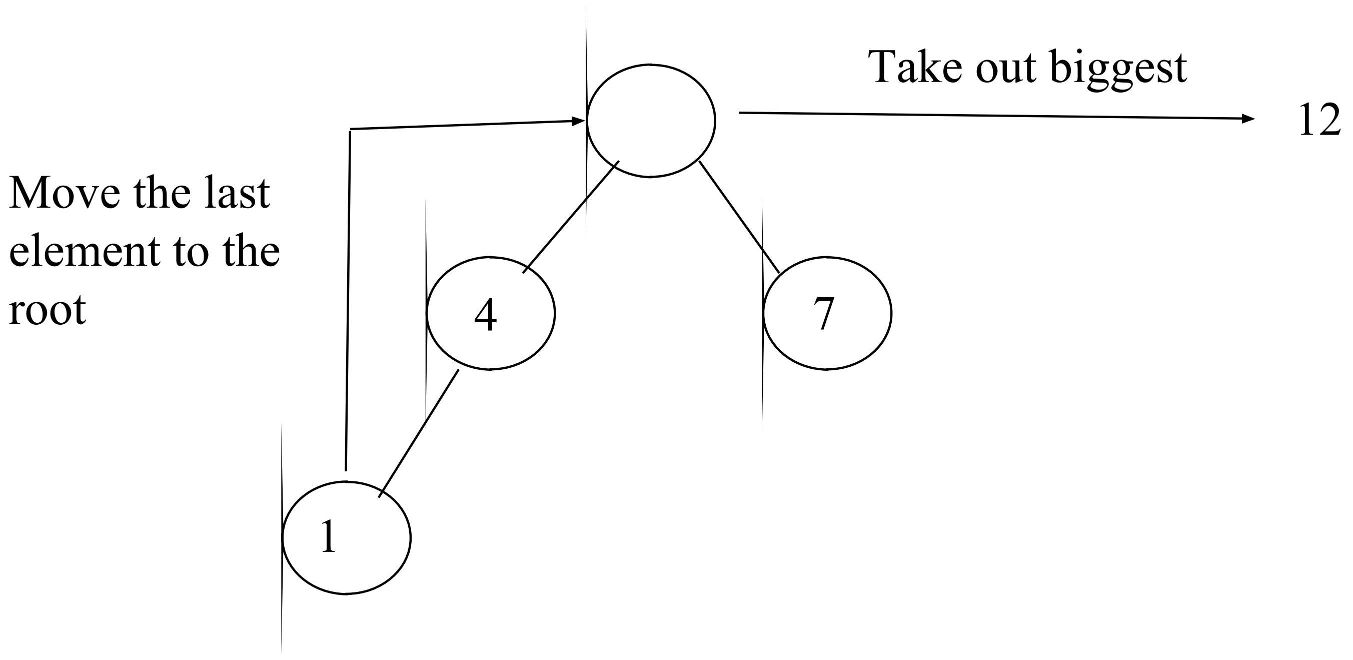


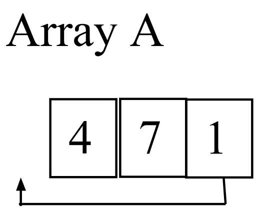
Array A

12	4	7	1

Sorted:

16	19

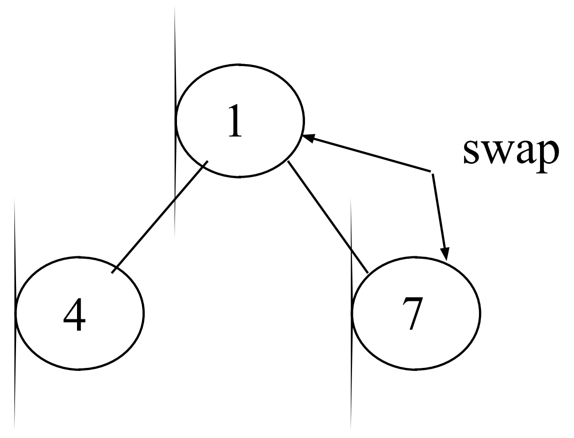




4	7	1

Sorted:

12	16	19

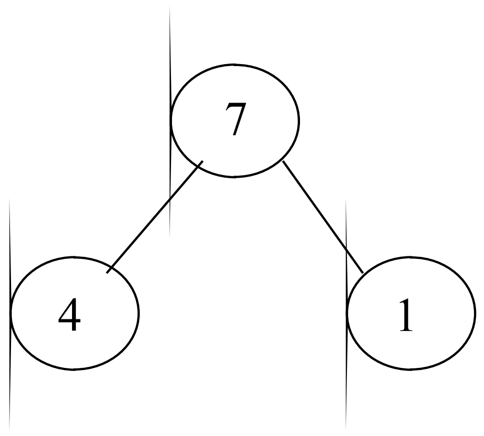


Array A

1	4	7

Sorted:

12	16	19

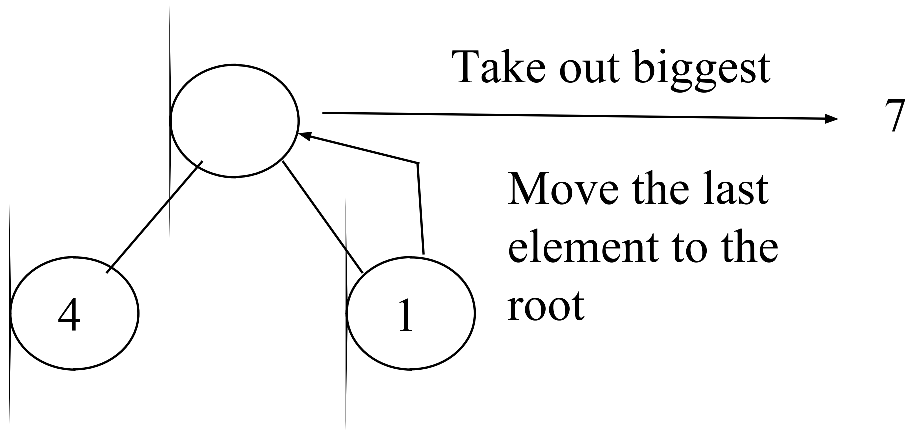


Array A

7	4	1

Sorted:

12	16	19



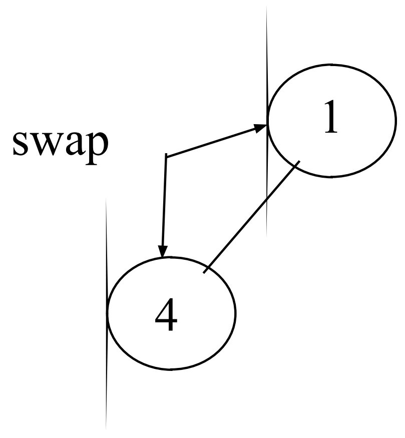
Array A

1	4

Sorted:

7	12	16	19

HEAPIFY()



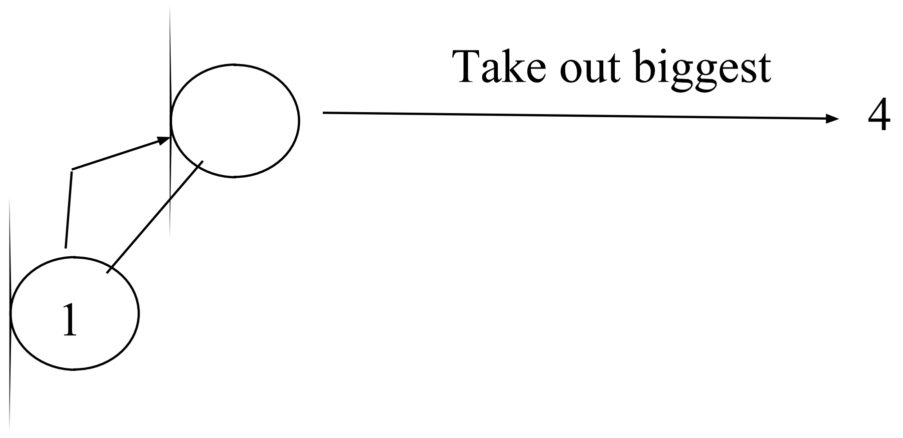
Array A

4	1

Sorted:

7	12	16	19

Move the last element to the root

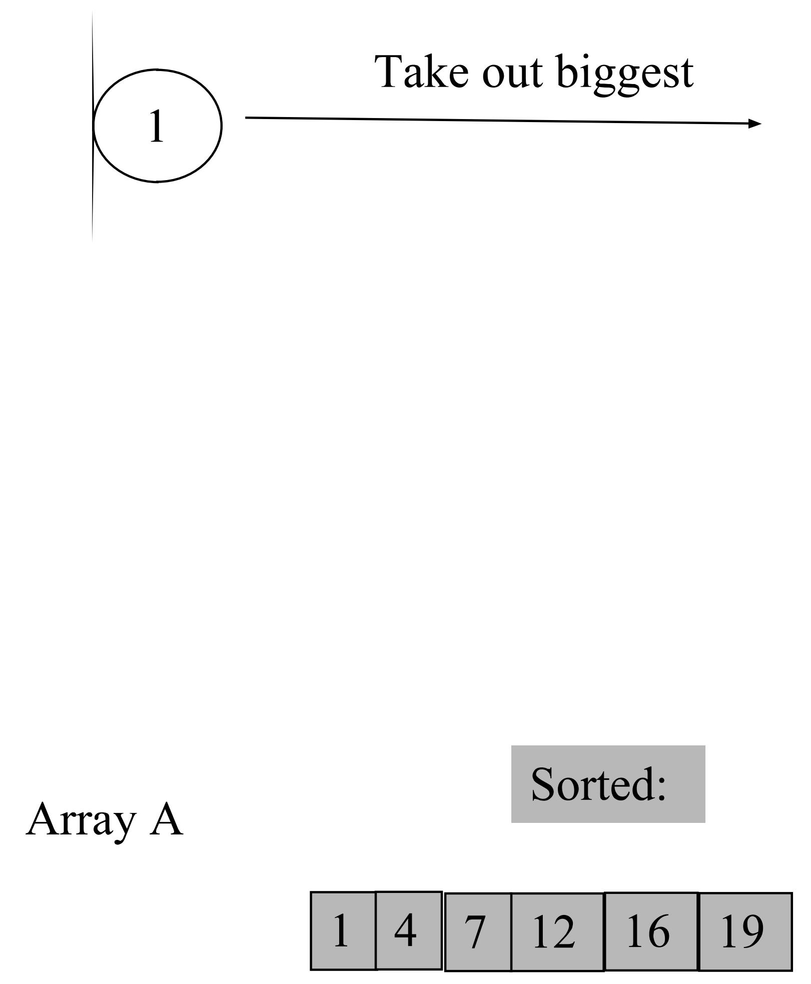


Array A

1

Sorted:

4	7	12	16	19



1
Take out biggest
Array A
Sorted:
1
4
7
12
16
19

Sorted:

1	4	7	12	16	19

TIME ANALYSIS

- Build Heap Algorithm will run in O(n) time
- There are n-1 calls to Heapify each call requires O(log n) time
- Heap sort program combine Build Heap program and Heapify, therefore it has the running time of O(n log n) time
- Total time complexity: O(n log n)

POSSIBLE APPLICATION

- When we want to know the task that carry the highest priority given a large number of things to do
- Interval scheduling, when we have a lists of certain task with start and finish times and we want to do as many tasks as possible
- Sorting a list of elements that needs and efficient sorting algorithm

CONCLUSION

- The primary advantage of the heap sort is its efficiency. The execution time efficiency of the heap sort is O(n log n). The memory efficiency of the heap sort, unlike the other n log n sorts, is constant, O(1), because the heap sort algorithm is not recursive.
- The heap sort algorithm has two major steps. The first major step involves transforming the complete tree into a heap. The second major step is to perform the actual sort by extracting the largest element from the root and transforming the remaining tree into a heap.