Definitions of 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

The heap sort algorithm has two major steps :

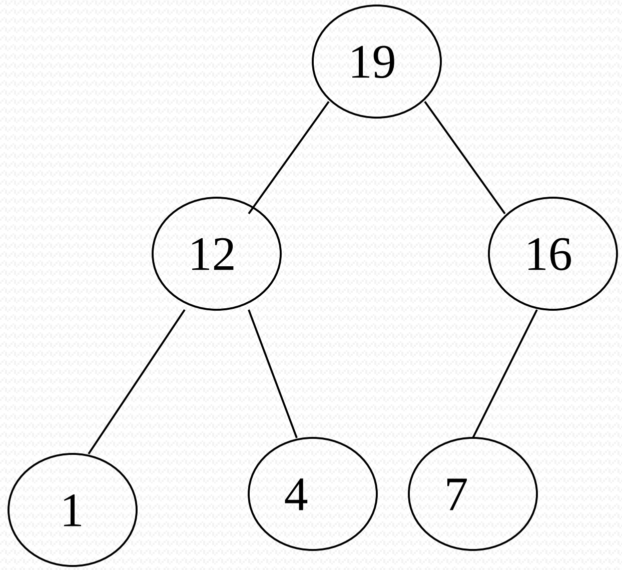
i. The first major step involves transforming the complete tree into a heap.

ii. The second major step is to perform the actual sort by extracting the largest or lowerst element from the root and transforming the remaining tree into a heap.

Types of heap

- Max Heap
- Min Heap

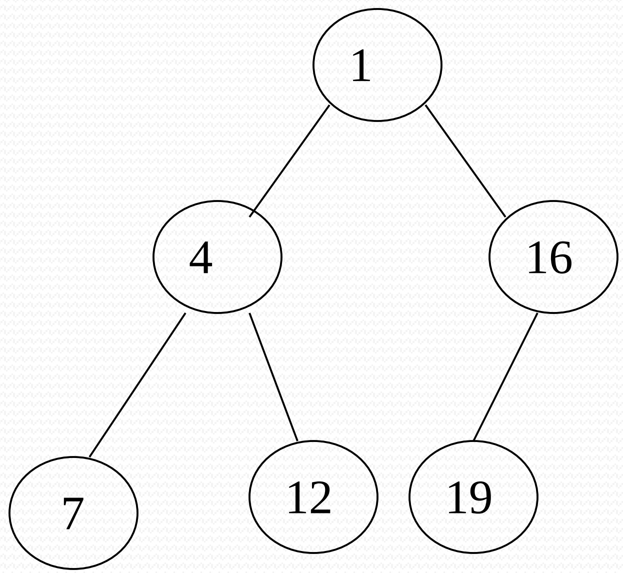
Max Heap Example



19	12	16	1	4	7

Array A

Min heap example



1	4	16	7	12	19

Array A

1-Max heap :

max-heap Definition:

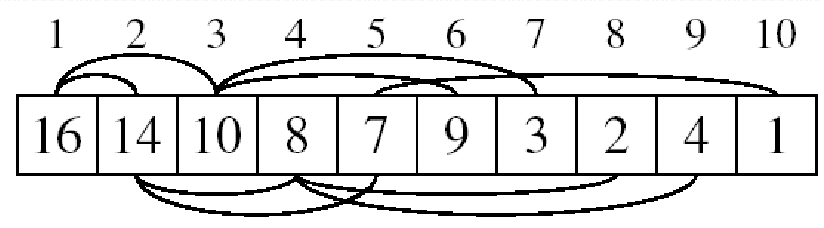
is a complete binary tree in which the value in each internal node is greater than or equal to the values in the children of that node.

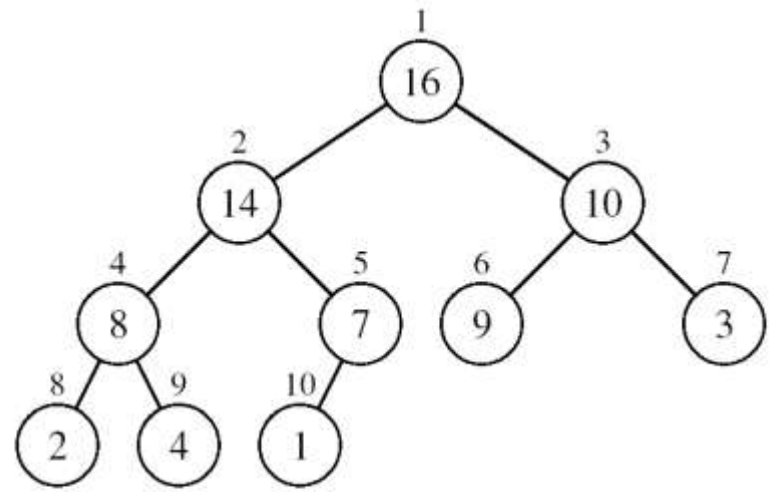
Max-heap property:

- The key of a node is ≥ than the keys of its children.

Max heap Operation

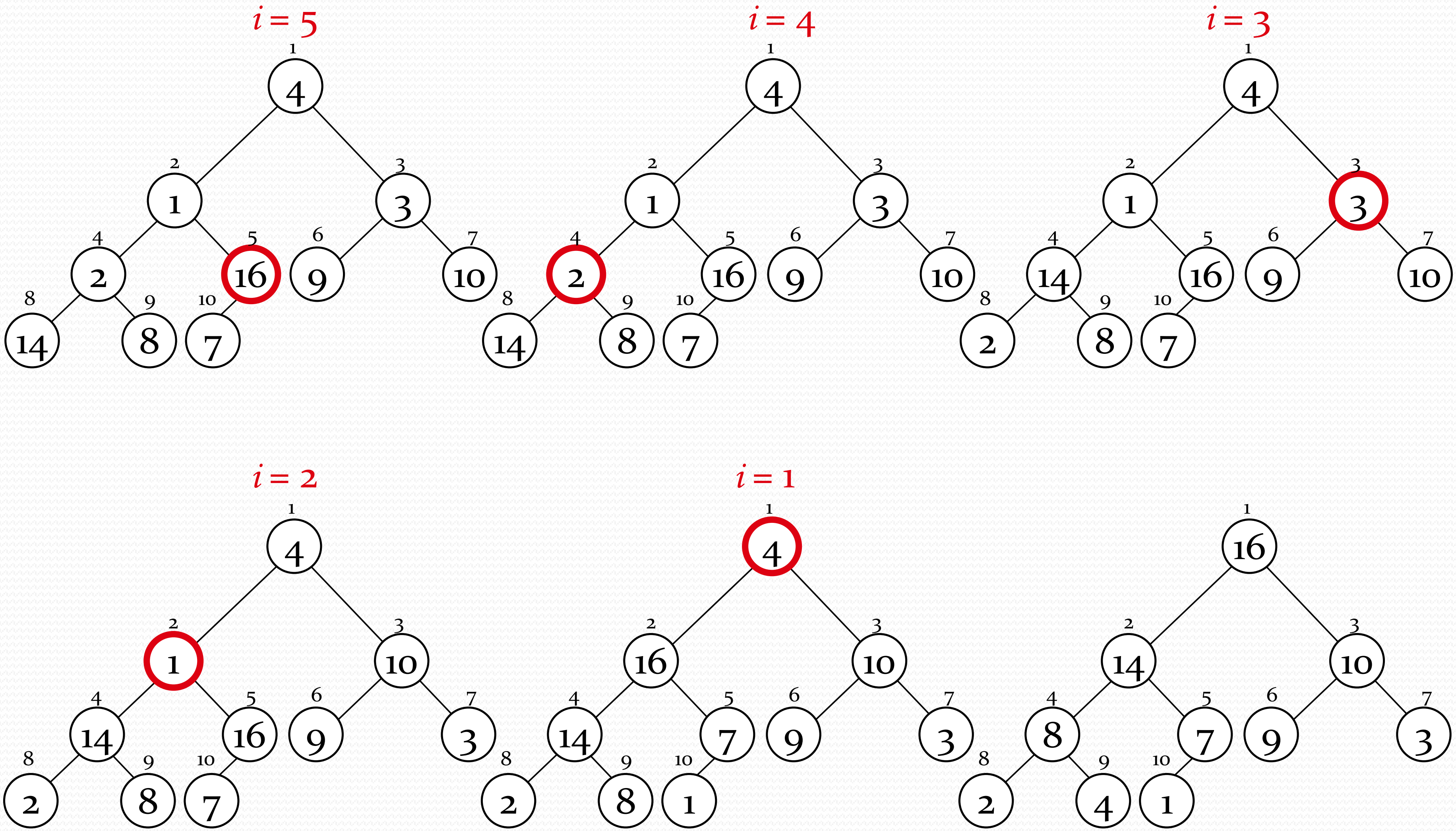
- A heap can be stored as an array *A*.
  - Root of tree is A[1]
  - Left child of A[i] = A[2i]
  - Right child of A[i] = A[2i + 1]
  - Parent of A[i] = A[$\lfloor i/2 \rfloor$]





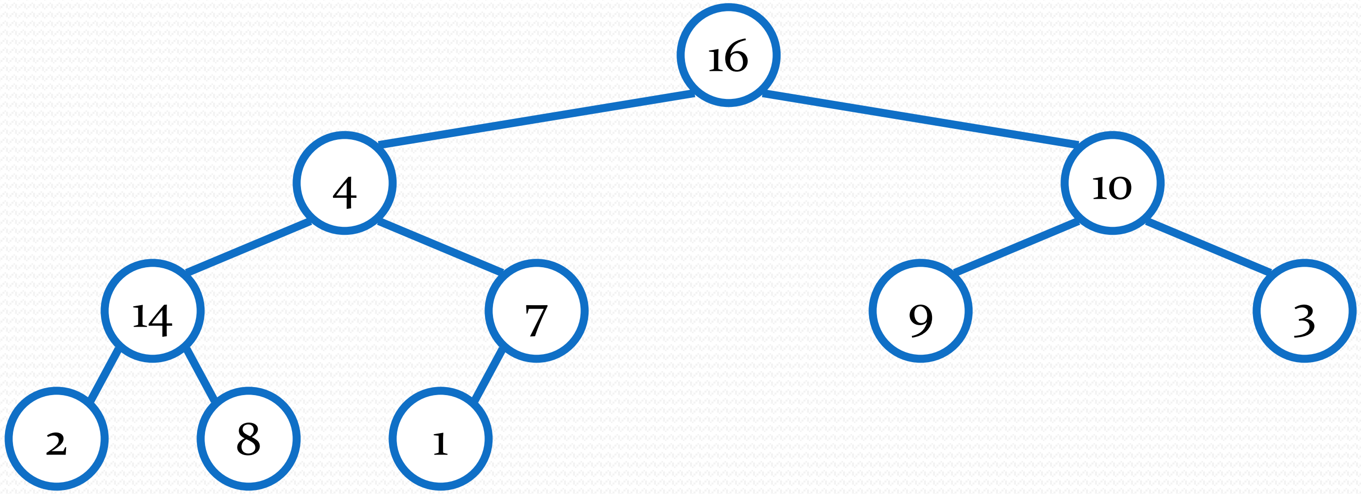
Example Explaining： A

4	1	3	2	16	9	10	14	8	7



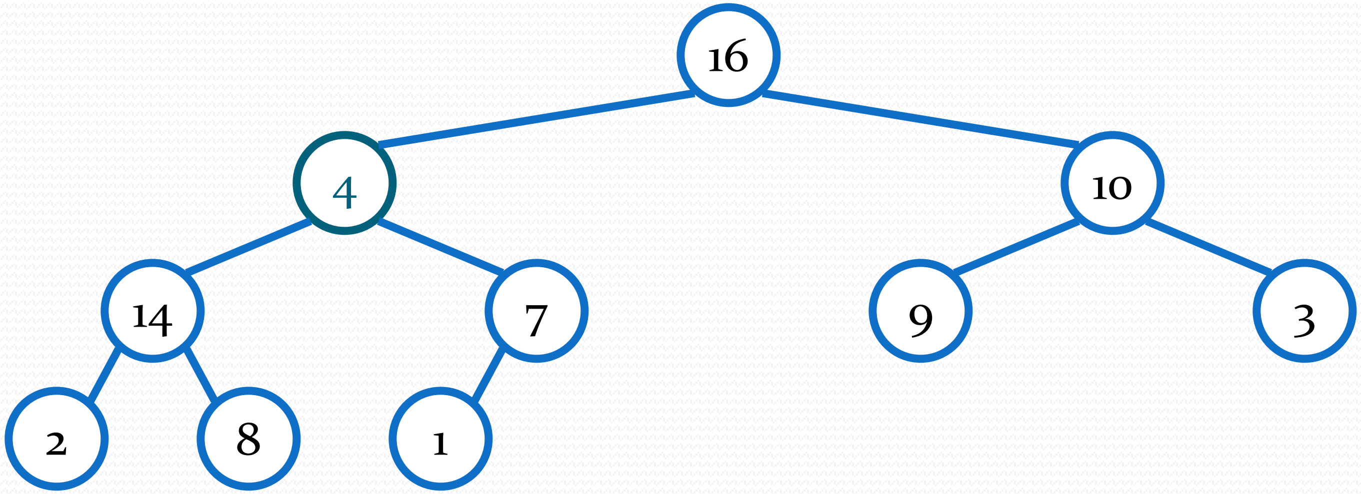
Build Max-heap

Heapify() Example



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

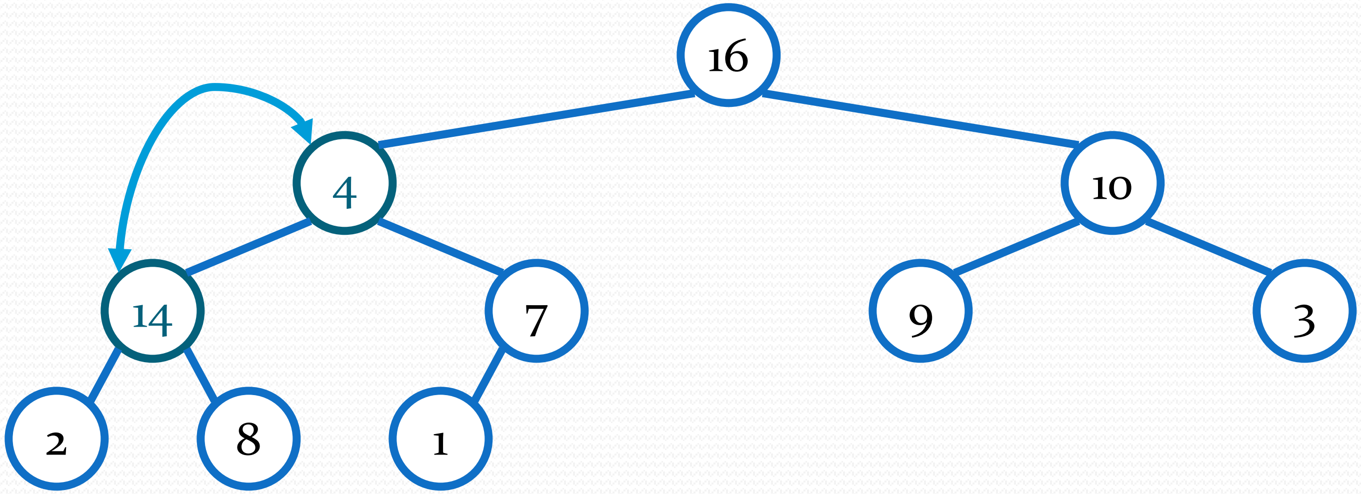
Heapify() Example

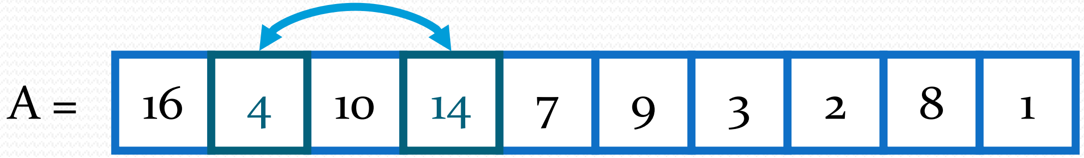


A =

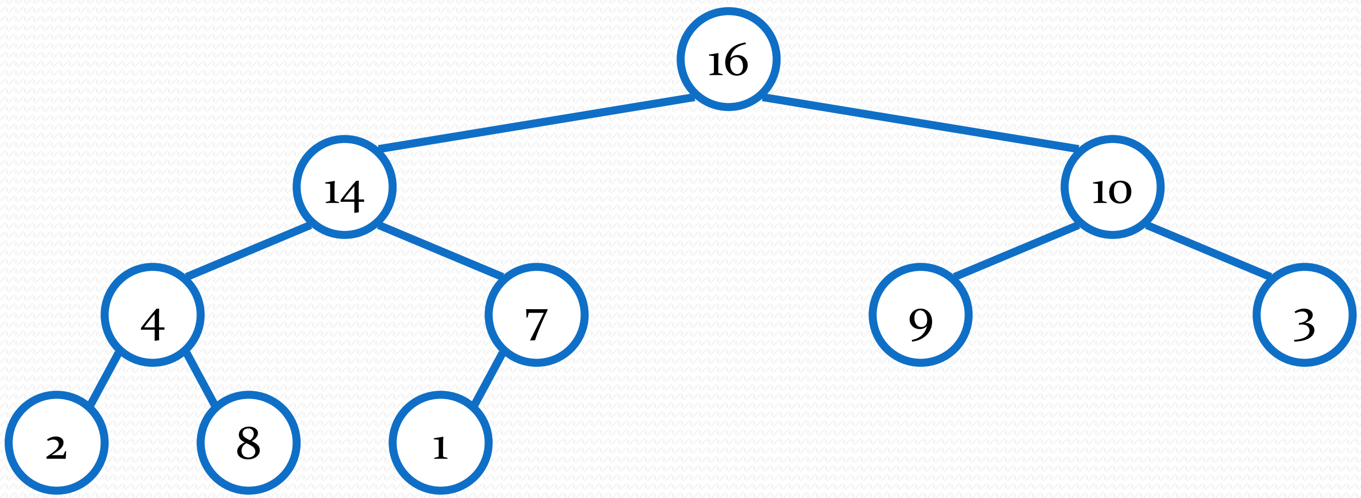
16	4	10	14	7	9	3	2	8	1

Heapify() Example





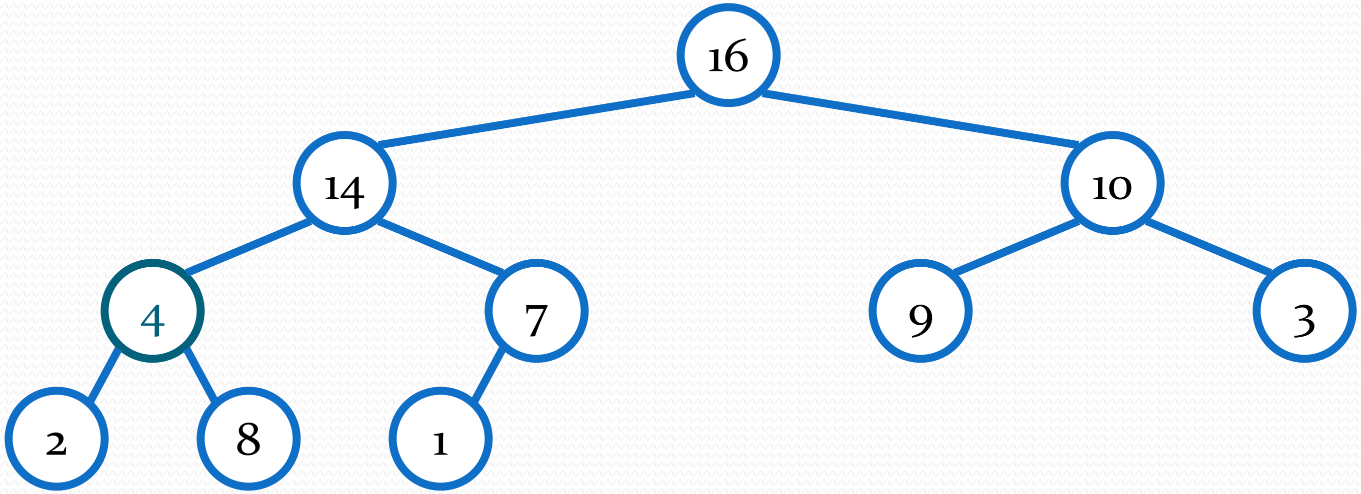
Heapify() Example



A =

16	14	10	4	7	9	3	2	8	1

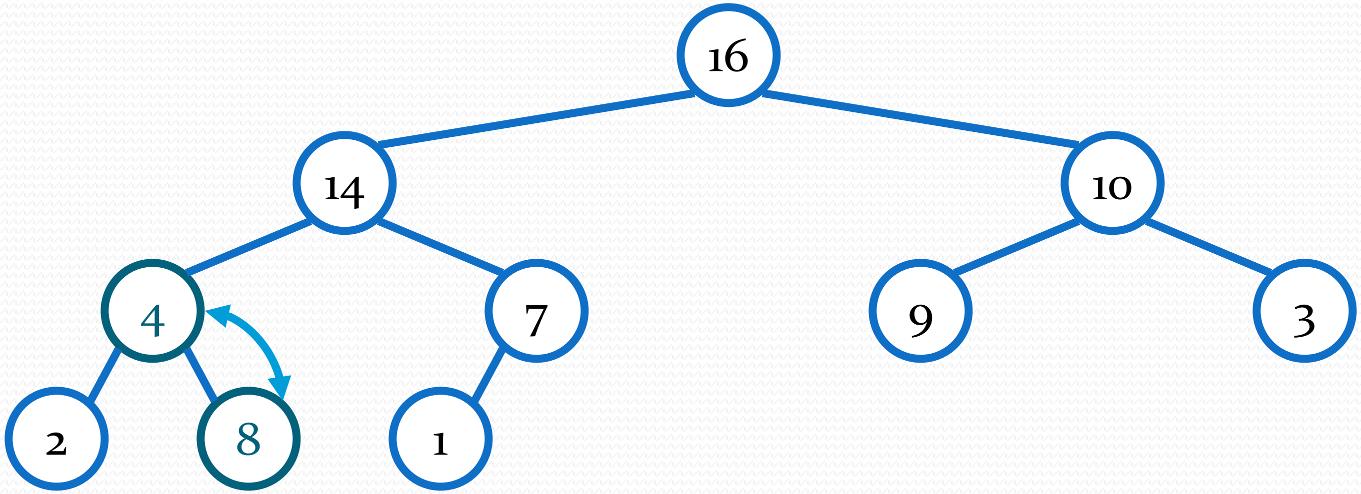
Heapify() Example



A =

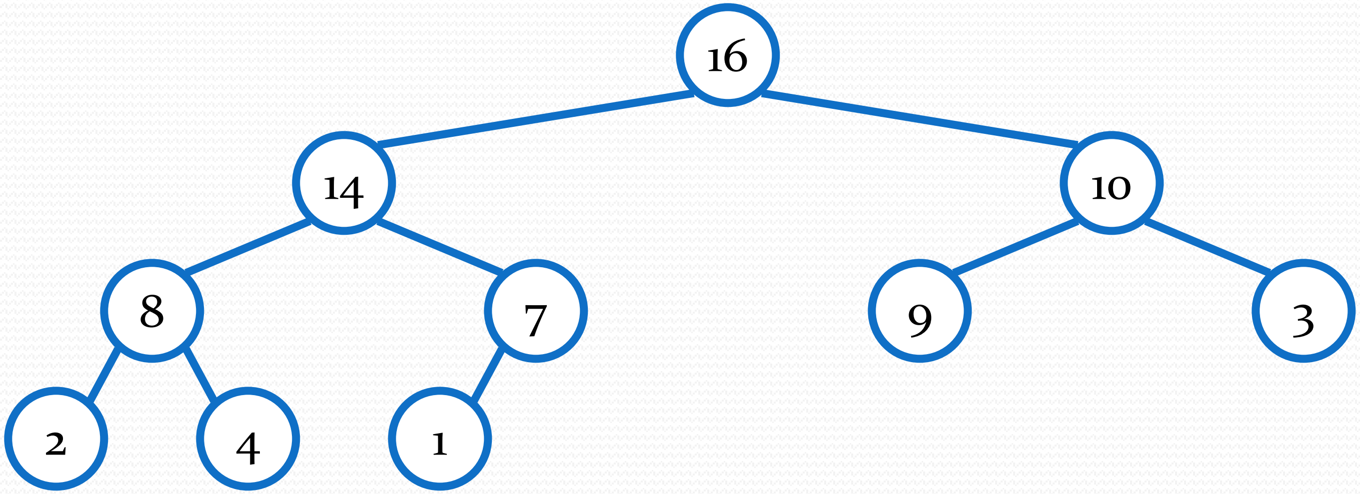
16	14	10	4	7	9	3	2	8	1

Heapify() Example



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

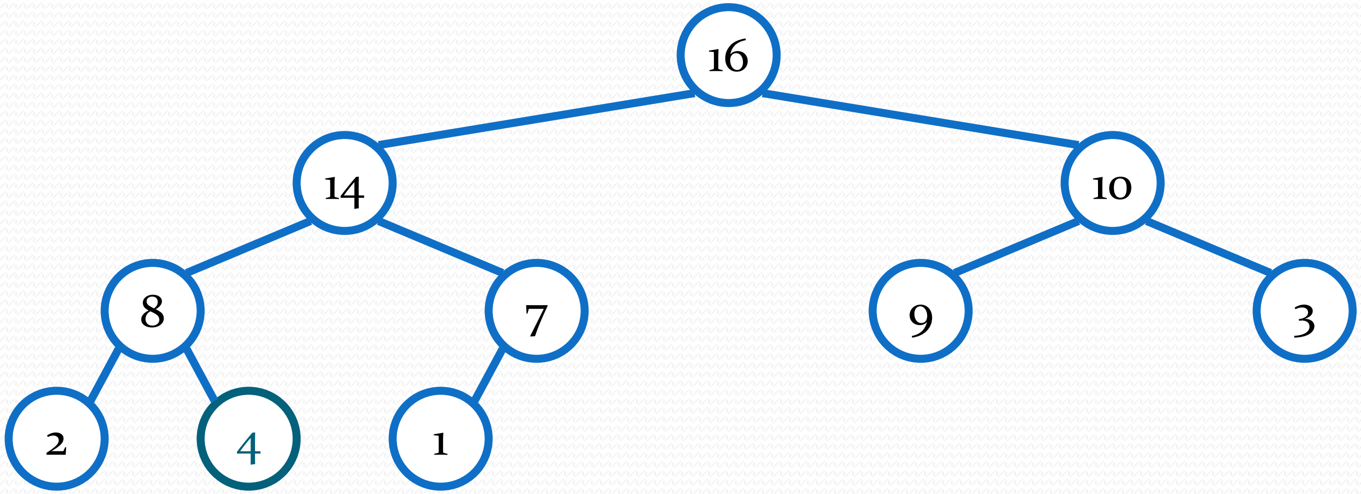
Heapify() Example



A =

16	14	10	8	7	9	3	2	4	1

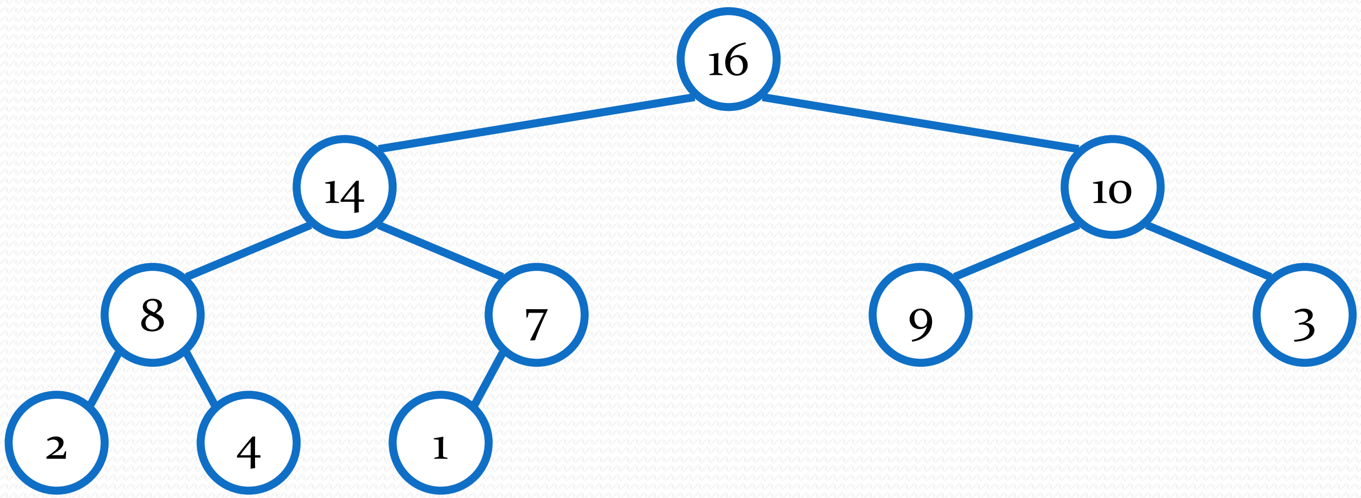
Heapify() Example



A =

16	14	10	8	7	9	3	2	4	1

Heapify() Example



A =

16	14	10	8	7	9	3	2	4	1

Heap-Sort

sorting strategy:

1. Build Max Heap from unordered array;
2. Find maximum element A[1];

3. Swap elements A[n] and A[1] :

   now max element is at the end of the array! .

4. Discard node n from heap

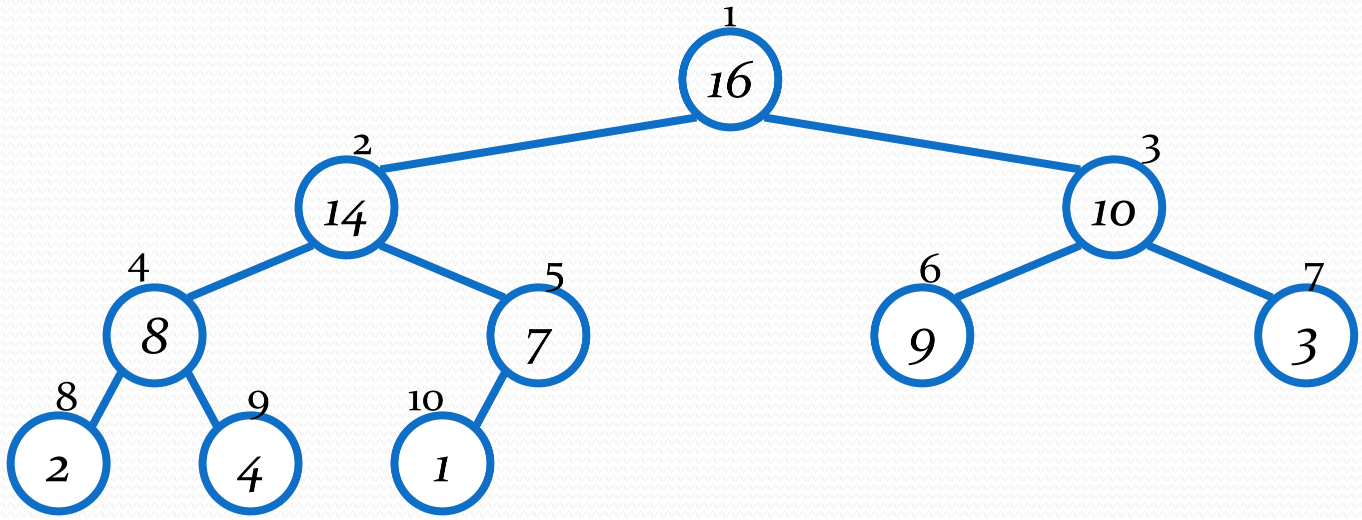
   (by decrementing heap-size variable).

5. New root may violate max heap property, but its children are max heaps. Run max_heapify to fix this.

6. Go to Step 2 unless heap is **empty**.

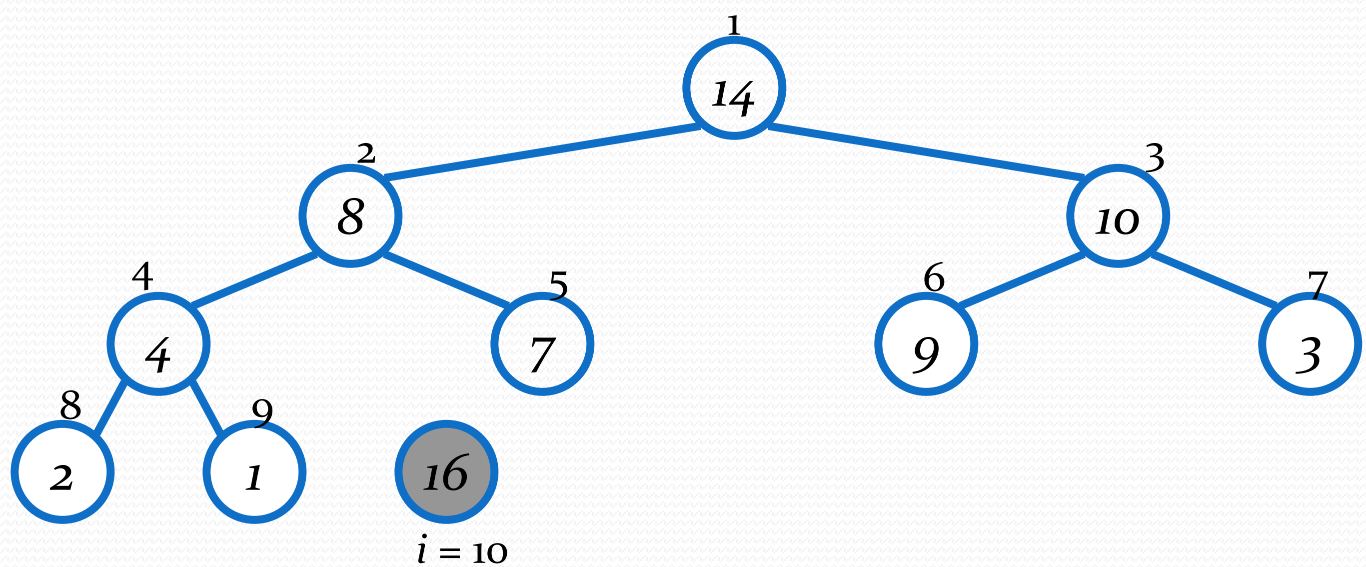
HeapSort() Example

- A = {16, 14, 10, 8, 7, 9, 3, 2, 4, 1}



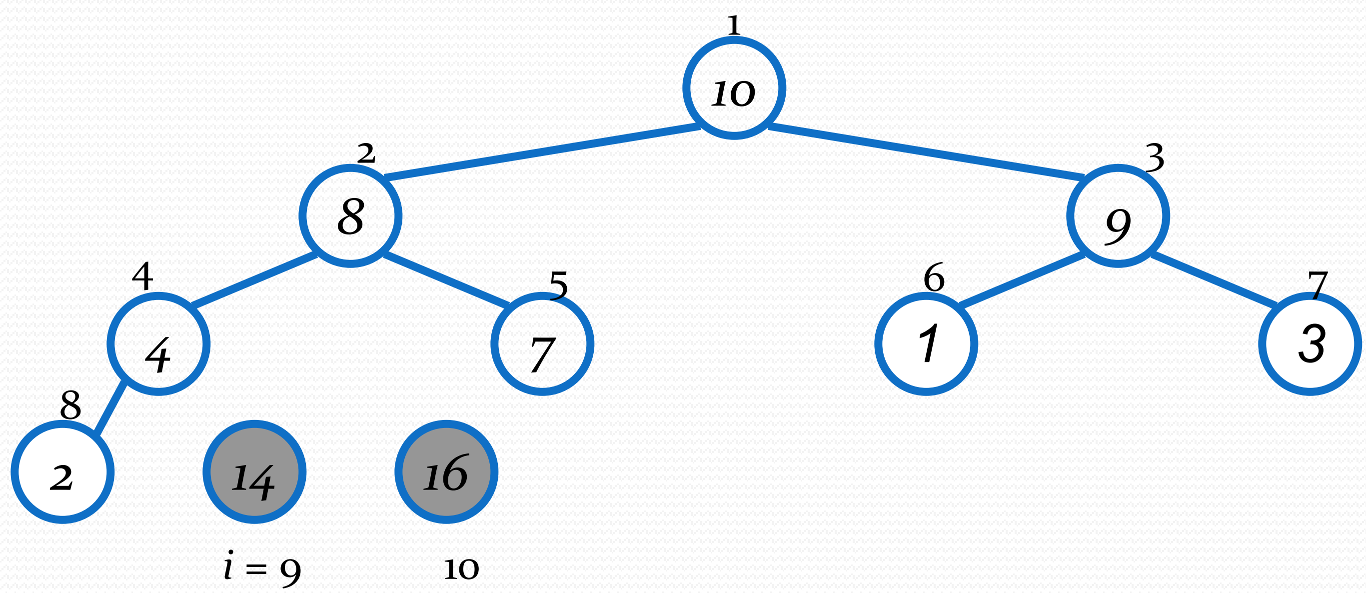
HeapSort() Example

- A = {14, 8, 10, 4, 7, 9, 3, 2, 1, **16**}



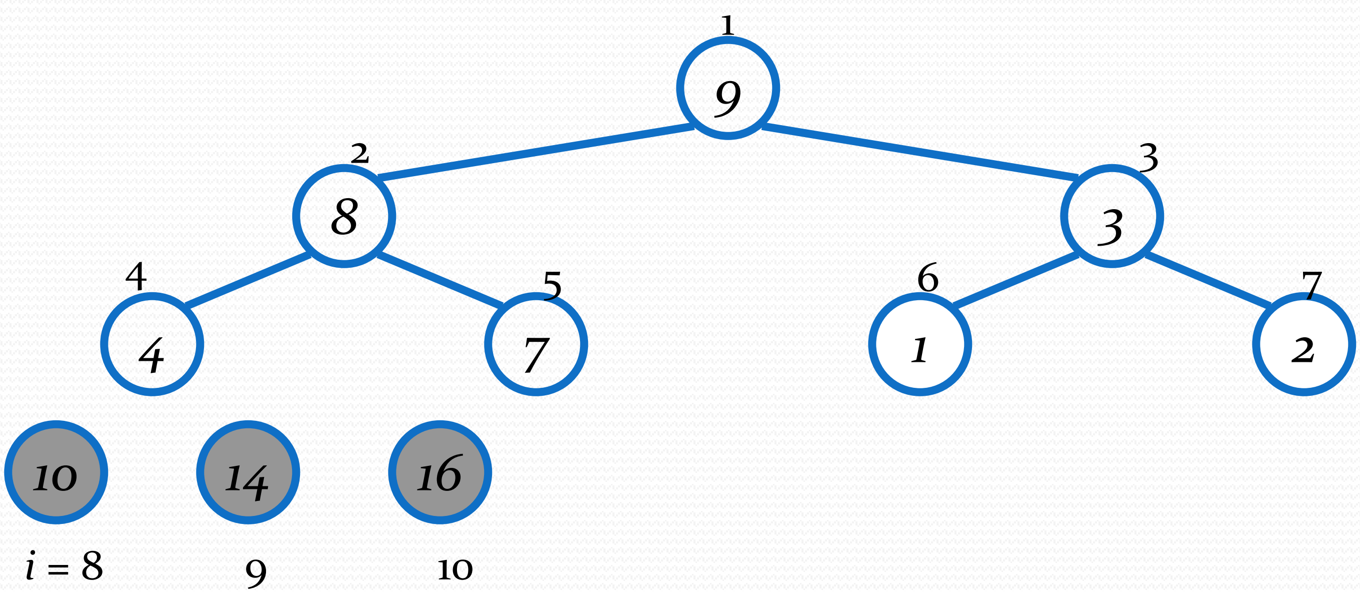
HeapSort() Example

- A = {10, 8, 9, 4, 7, 1, 3, 2, **14**, **16**}



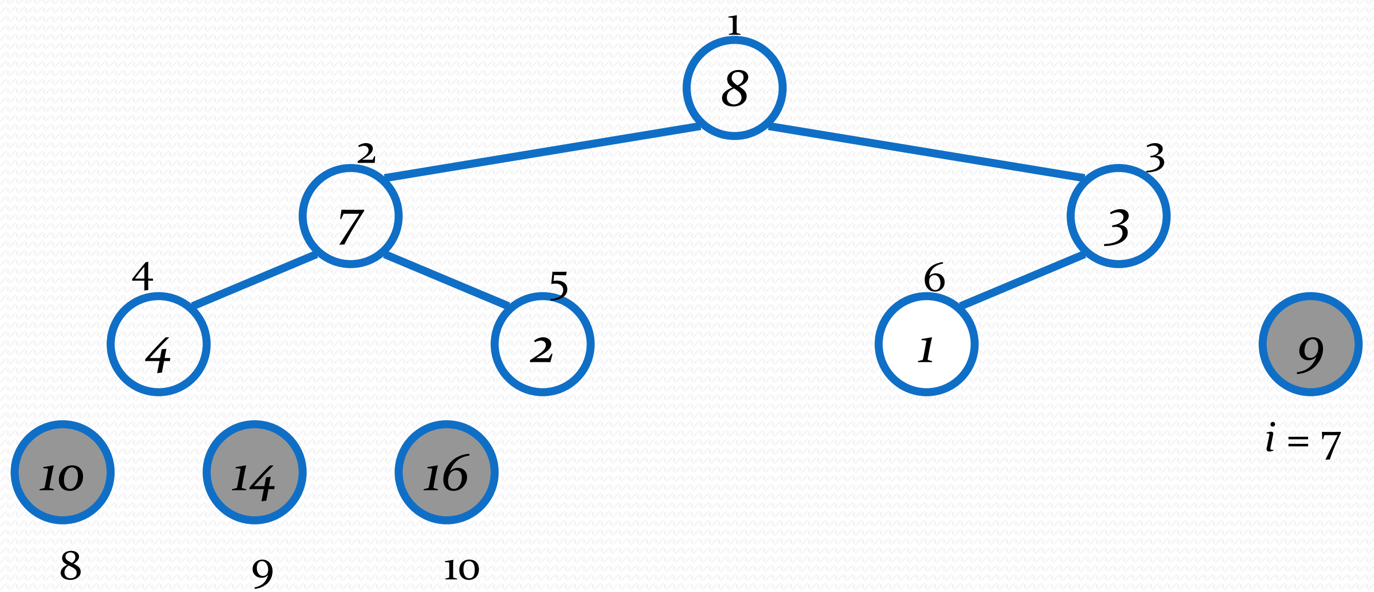
HeapSort() Example

- A = {9, 8, 3, 4, 7, 1, 2, **10**, **14**, **16**}



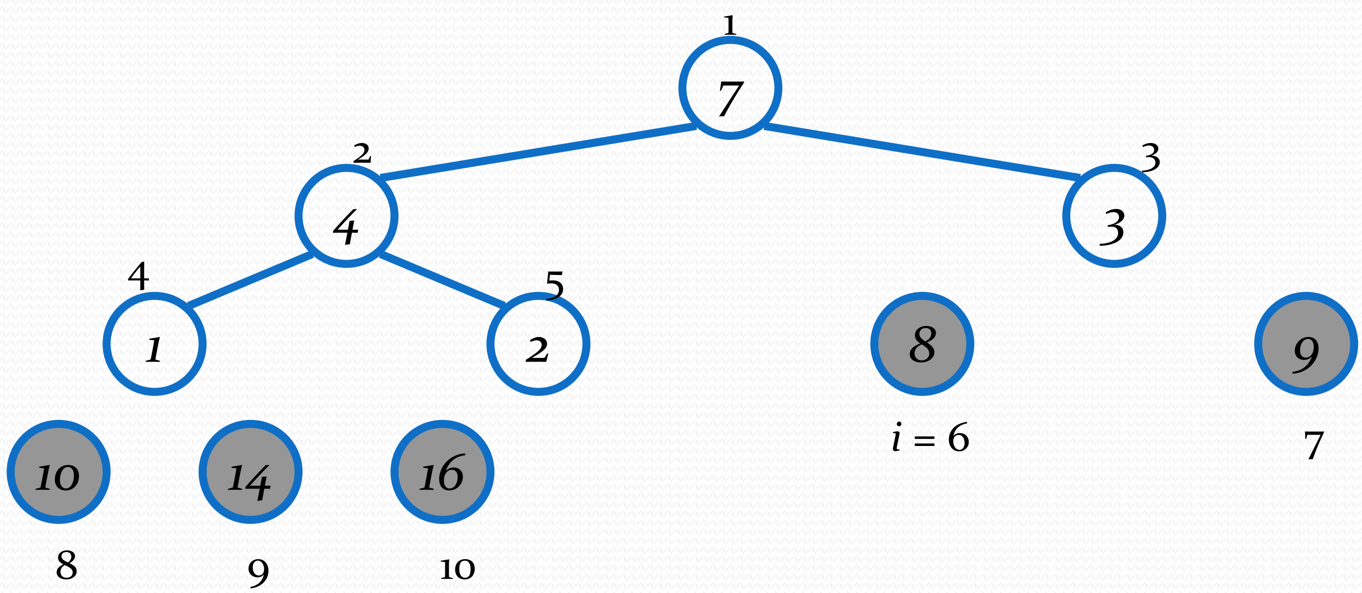
HeapSort() Example

- A = {8, 7, 3, 4, 2, 1, **9**, **10**, **14**, **16**}



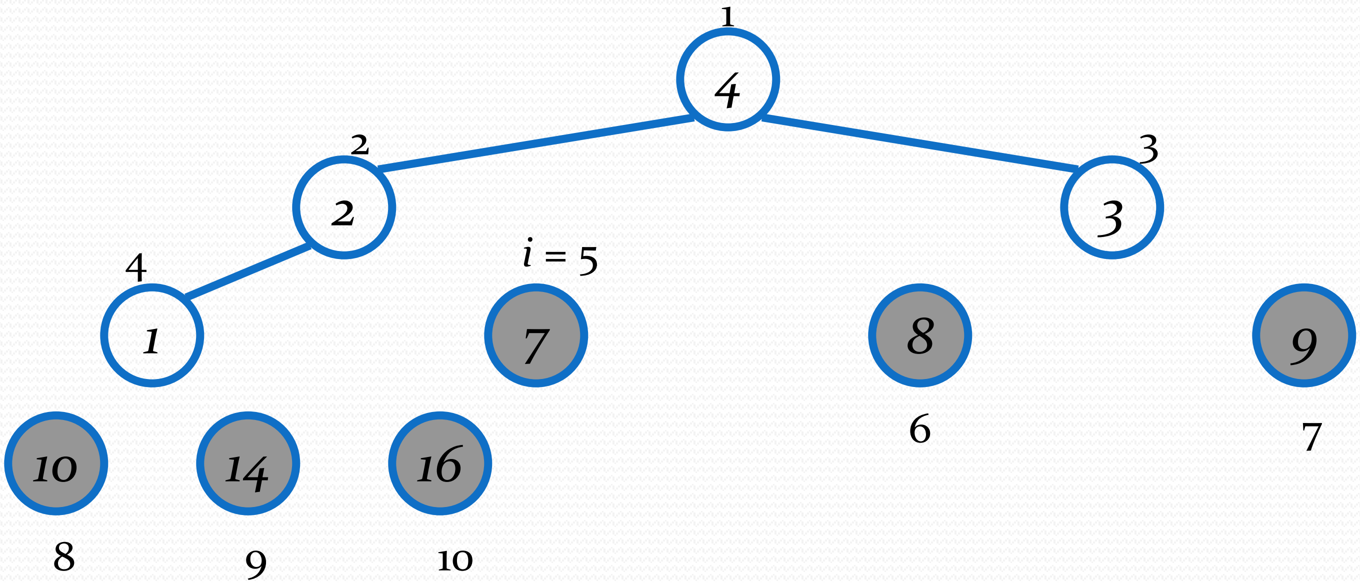
HeapSort() Example

- A = {7, 4, 3, 1, 2, **8**, **9**, **10**, **14**, **16**}



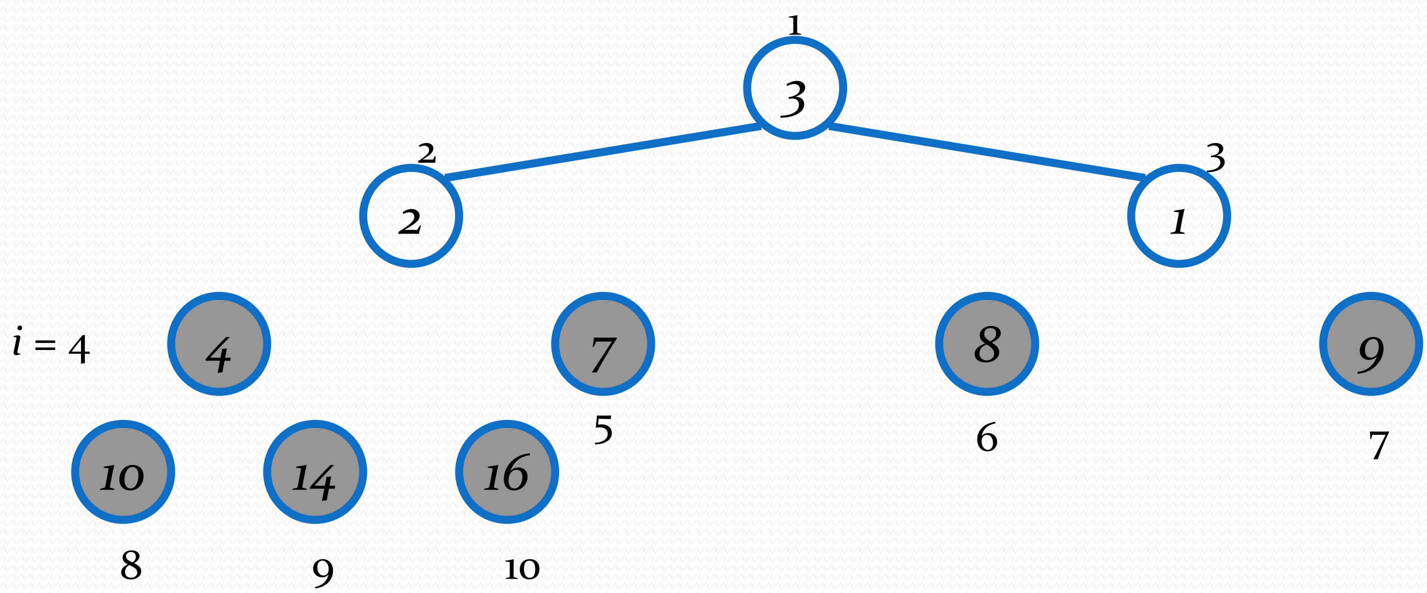
HeapSort() Example

- A = {4, 2, 3, 1, **7**, **8**, **9**, **10**, **14**, **16**}



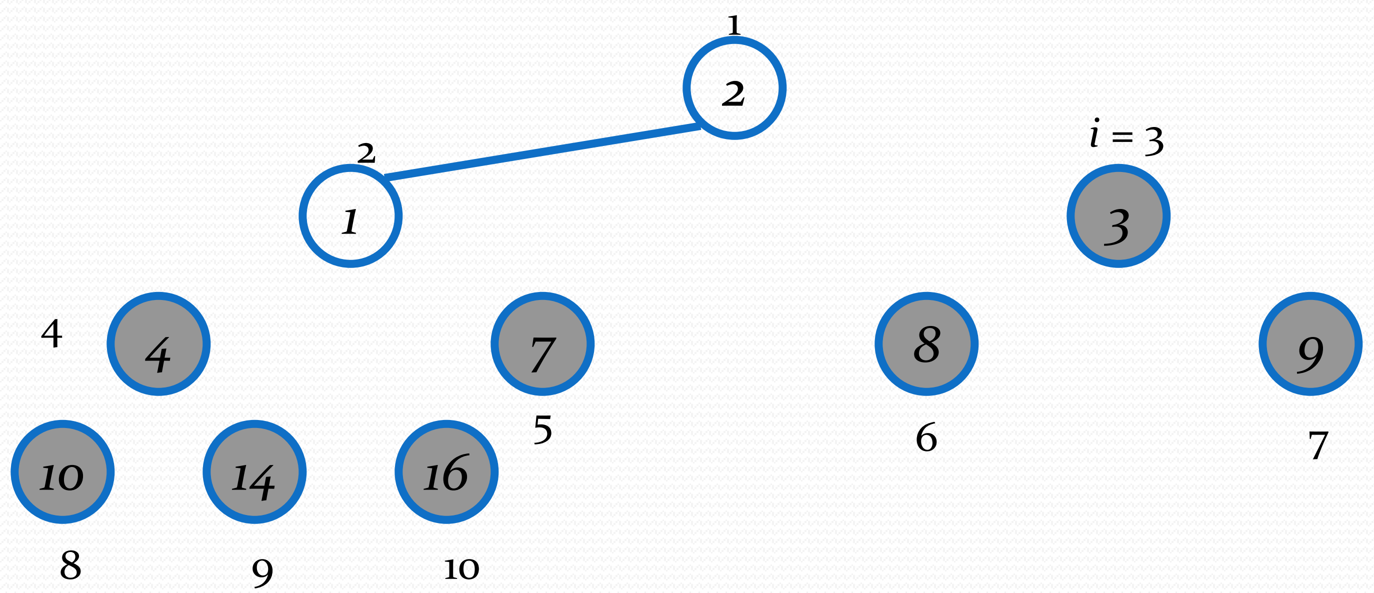
HeapSort() Example

- A = {3, 2, 1, **4**, **7**, **8**, **9**, **10**, **14**, **16**}



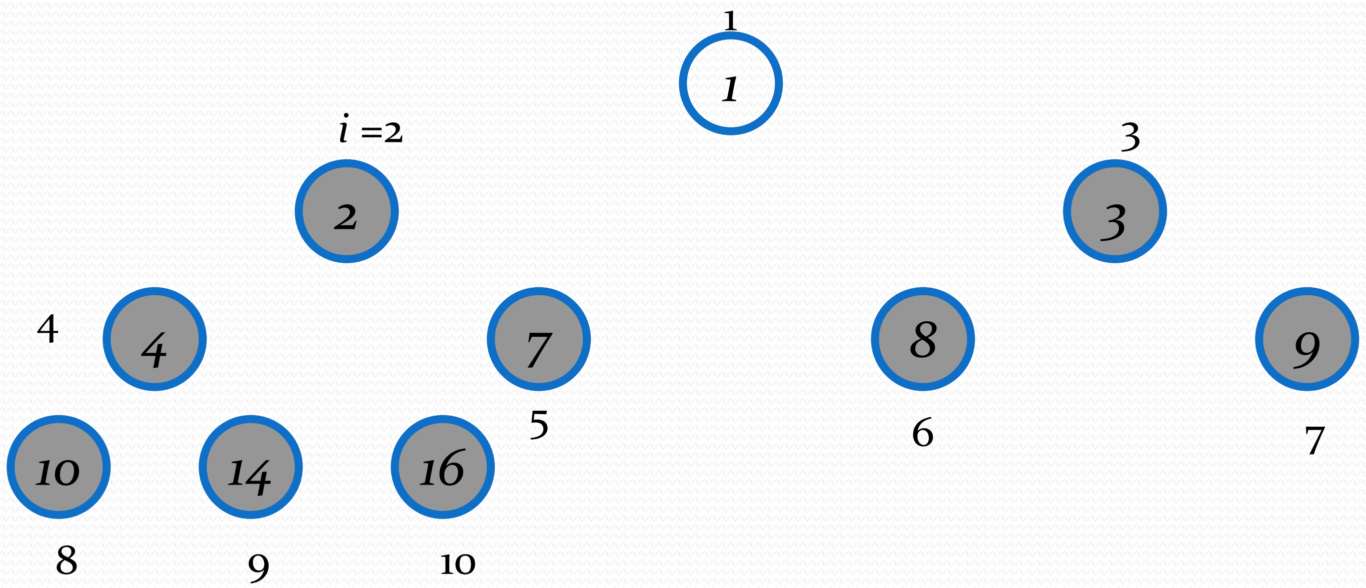
HeapSort() Example

- A = {2, 1, **3**, **4**, **7**, **8**, **9**, **10**, **14**, **16**}



HeapSort() Example

- A = {1, **2**, **3**, **4**, **7**, **8**, **9**, **10**, **14**, **16**} >>orederd



Heap Sort pseducode

```
Heapsort(A as array)
    BuildHeap(A)
    for i = n  to 1
        swap(A[1], A[i])
        n = n - 1
        Heapify(A, 1)

BuildHeap(A as array)
    n = elements_in(A)
    for i = floor(n/2) to 1
        Heapify(A,i)
```

```
Heapify(A as array, i as int)
    left = 2i
    right = 2i+1

    if (left <= n) and (A[left] > A[i])
        max = left
    else
        max = i

    if (right<=n) and (A[right] > A[max])
        max = right

    if (max != i)
        swap(A[i], A[max])
        Heapify(A, max)
```

2-Min heap :

min-heap Definition:

is a complete binary tree in which the value in each internal node is lower than or equal to the values in the children of that node.

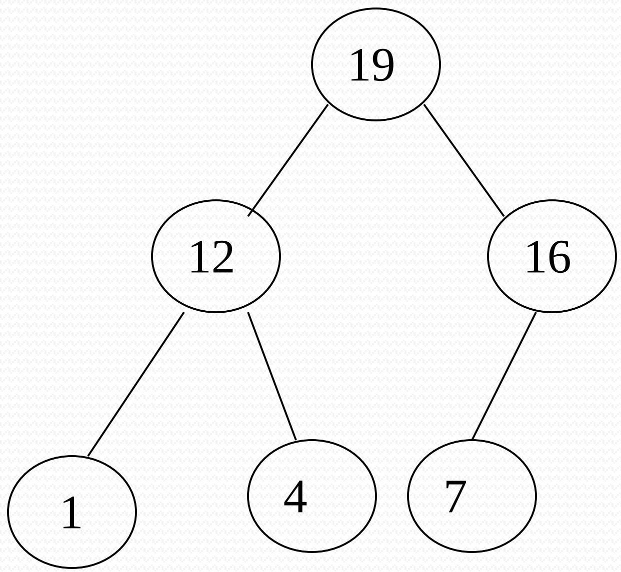
Min-heap property:

- The key of a node is <= than the keys of its children.

Min heap Operation

- A heap can be stored as an array *A*.
  - Root of tree is A[0]
  - Left child of A[i] = A[2i+1]
  - Right child of A[i] = A[2i + 2]
  - Parent of A[i] = A[(i/2)-1]

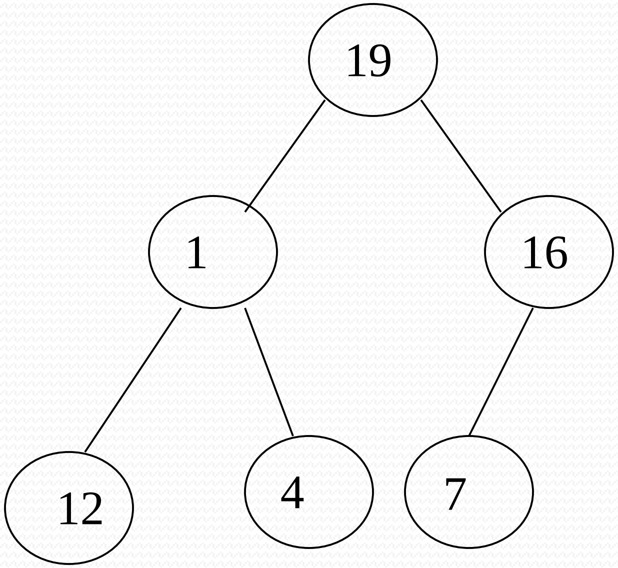
Min Heap



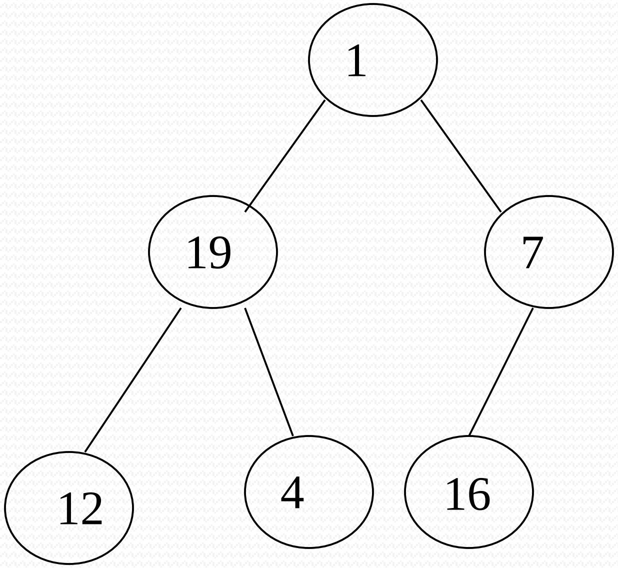
19	12	16	1	4	7

Array A

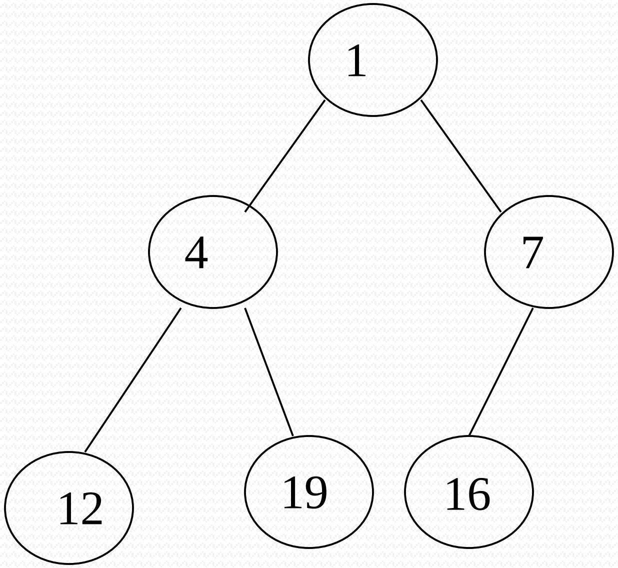
Min Heap phase 1



Min Heap phase 1

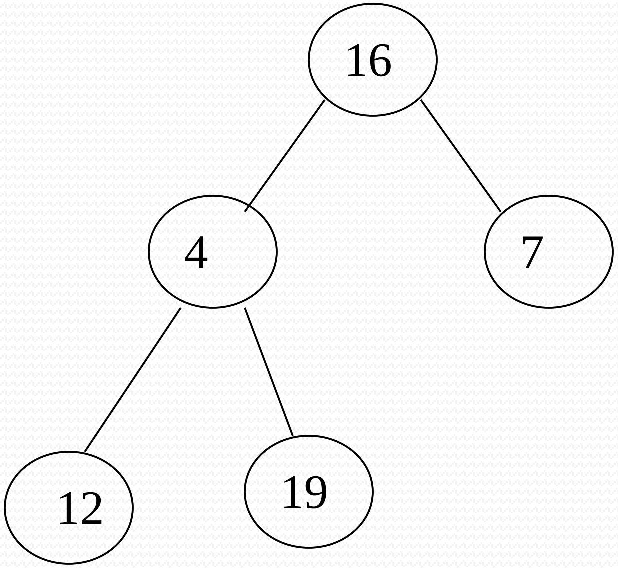


Min Heap phase 1

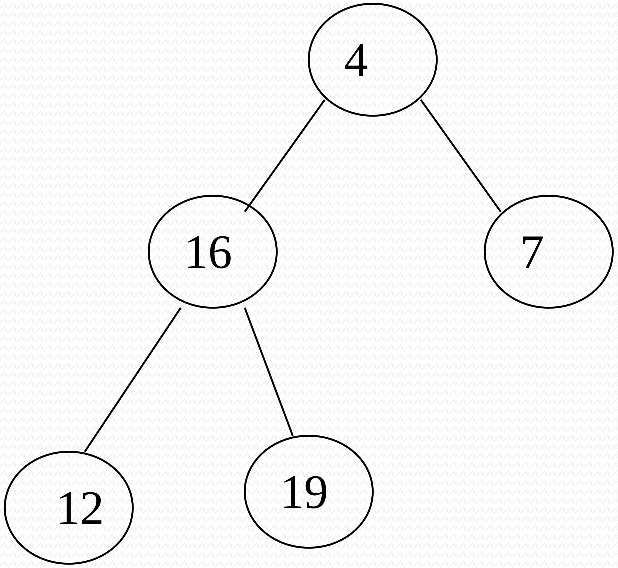


1,

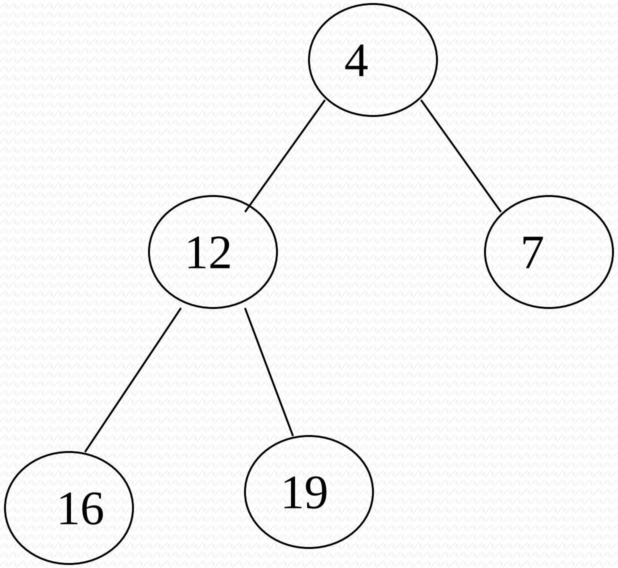
Min Heap phase 2



Min Heap phase 2

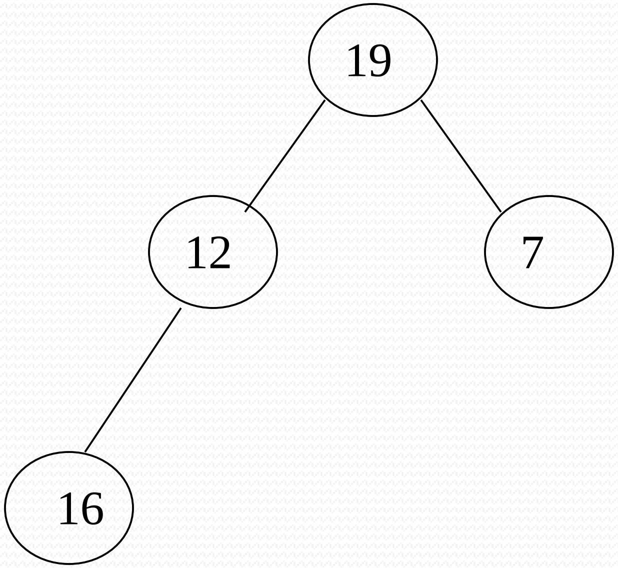


Min Heap phase 2

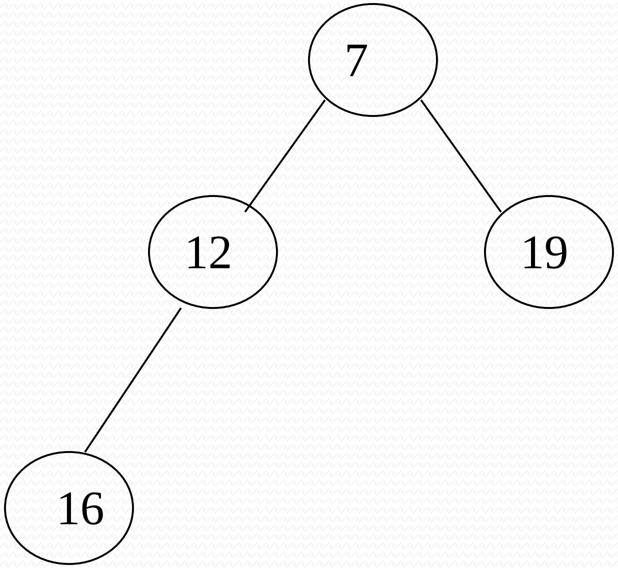


1,4

Min Heap phase 3

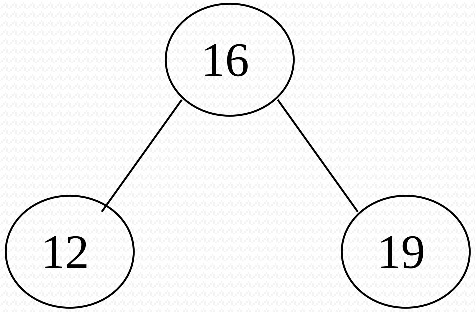


Min Heap phase 3

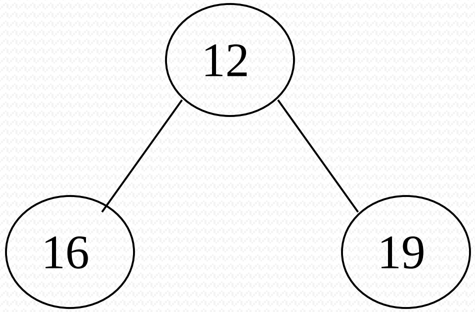


1,4,7

Min Heap phase 4

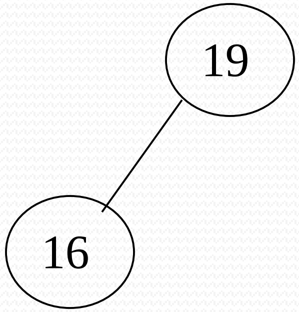


Min Heap phase 4

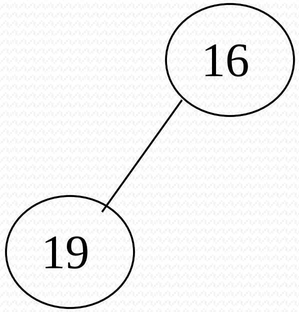


1,4,7,12

Min Heap phase 5

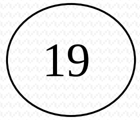


Min Heap phase 5



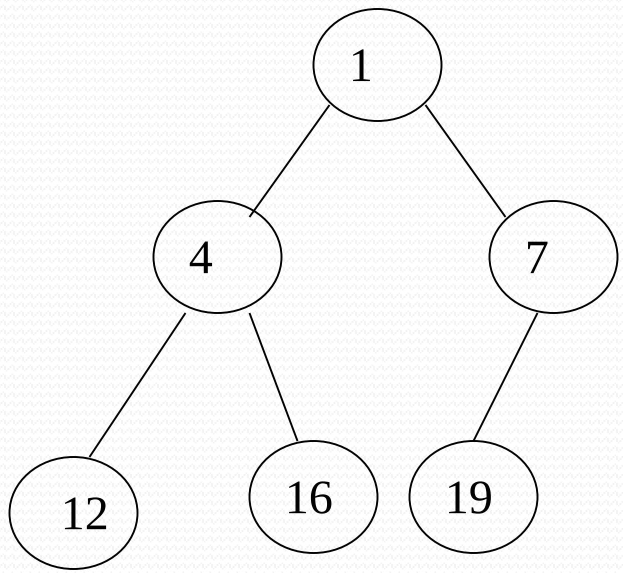
1,4,7,12,16

Min Heap phase 7



1,4,7,12,16,19

Min heap final tree



1	4	7	12	16	19

Array A

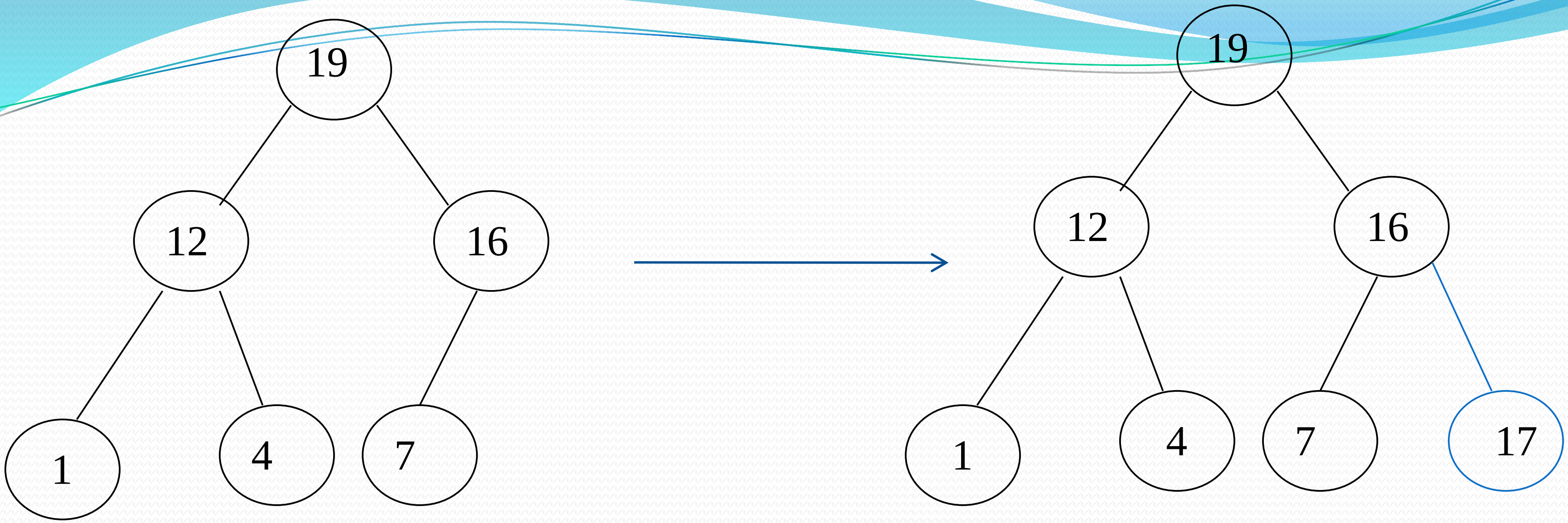
Insertion

- 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.

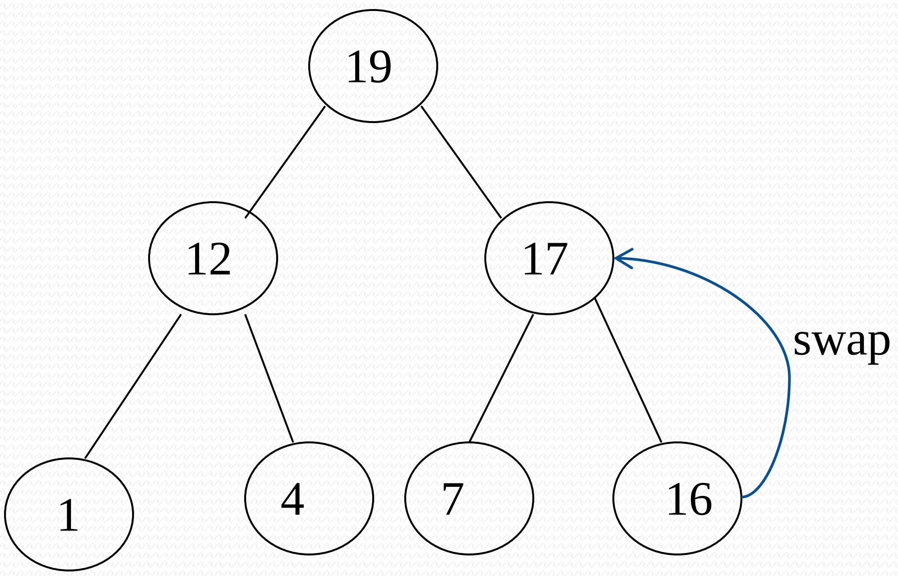
OR

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.



Insert 17



Percolate up to maintain the heap property

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.