Heap Algorithms

- Assume that the heap is to be stored in an array with *size* elements. (A[0..size-1])
- Heapify is called passing it the root of some subtree (index i in the algorithm).
- The algorithm assumes that the left and right **subtrees** of node A[i] are both heaps. However A[i] may have a value smaller that one or both children. Heapify fixes the problem so that the subtree rooted at A[i] is a heap:
- Heapify does not make a heap out of a tree but makes a heap out of the subtree rooted at node i.

```
heapify (A[], int size,  i)      // A is an array; i is the index of a node
{
        left = 2i+1;             //left child of A[i]
        right = 2i+2;            //right child of A[i]

        if ( left < size  && A[left] > A[i])
                largest = left;
        else
                largest = i;


        if ( right < size && A[right] > A[largest]
                largest = right;

        if (largest != i)
        {
                swap (A[i], A[largest]);
                heapafy(A, size, largest);
        }
}
```

Building a Heap

// this builds a heap by calling heapify on each node
// start with the deepest node with children

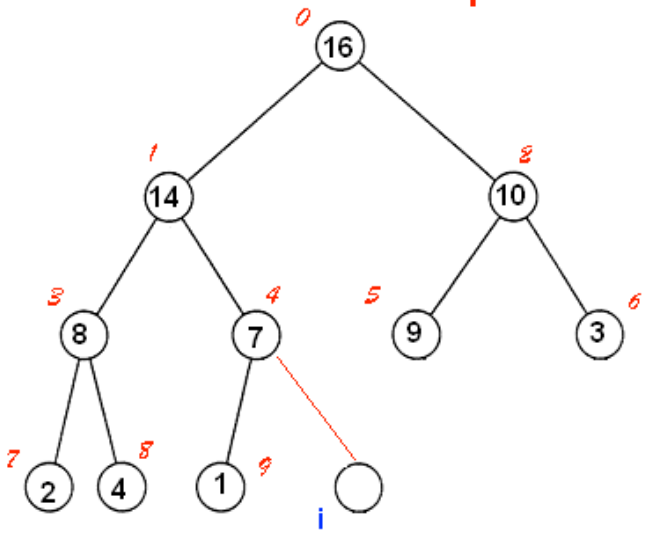
```
buildHeap(A[], int size)  // A[0..size – 1]
{
   for (int i = (size –2)/2 down 0)  // from the first node with children to root
                heapafy(A, size, i);
}
```

Priority Queue Algorithms Using a Heap Implementation

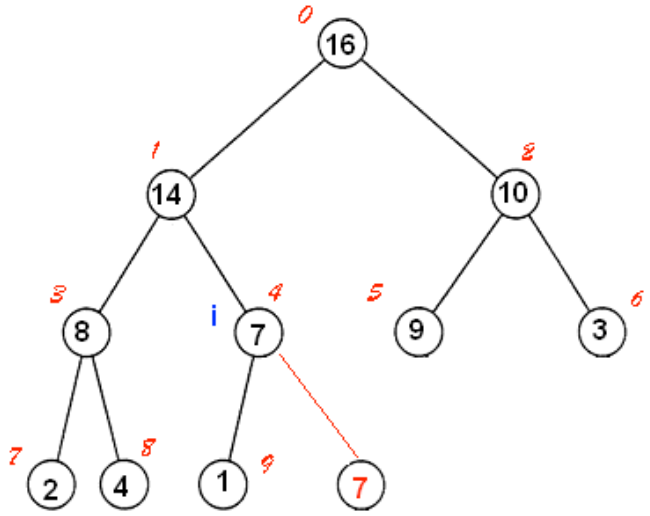
```
insert( A, size, max, key)        // key  holds the priority,
                                  //size is  current number of data in the heap
                                  // max is the   maximum number of data
                                  // that CAN be stored in A
{

     if ( size == max)
          error (“overflow”)


     i = size;


     while (i > 0 && A[ (i-1)/2] < key)   // (i-1)/ 2 is the parent of i
     {
            A[i] = A[(i-1)/2]              // move value in the parent of A[i} down to A[i]
             i= parent (i)
      }

      A[i] = key;  // add new element to tree
      size++;
}
```

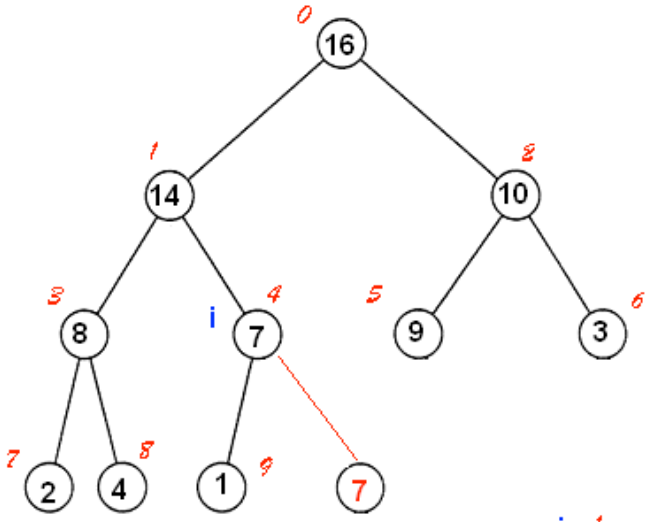
```
remove(A , size)  // returns element with the highest priority
{
          if (size < 1)
                    error (“heap underflow”)

          max = A[0]            // top element

          A[0] = A[size – 1]  // move last element to top
          size--;

          heapafy(A, size, 0)  // adjust the heap
          return max
}
```

Insert 15 into the heap

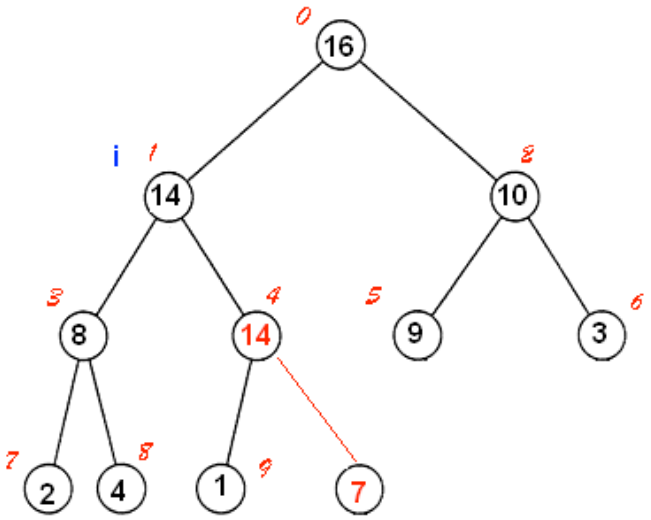


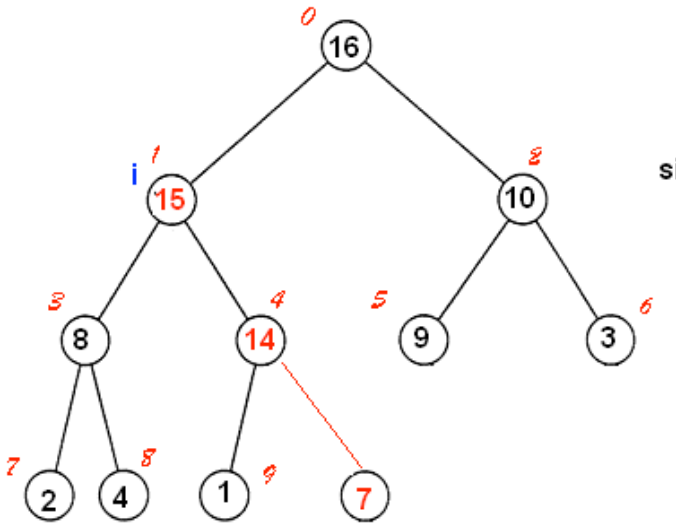
size = 10
set i = size
since i >0 && parent(i) <15
move parent(i) to i
move i to parent(i)





since i >0 && parent(i) <15
move parent(i) to i
move i to parent(i)

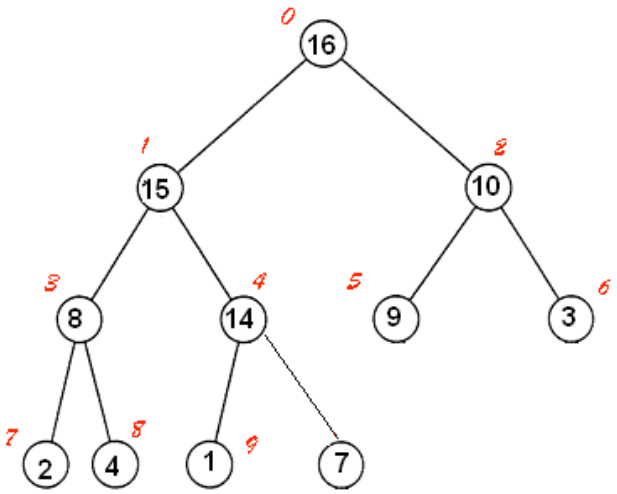




since i >0 but parent(i) [16] > 15

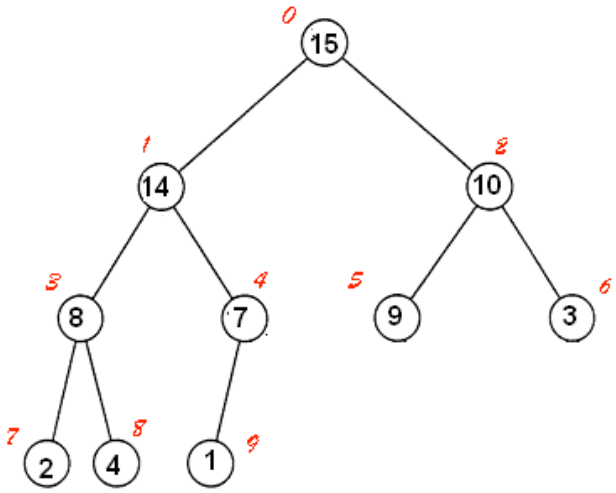
place 15 in node (i)

Remove top element



size = 11
max = A[0] = 16
move last element to A[0]
decrement size
heapafy from the root

max
16



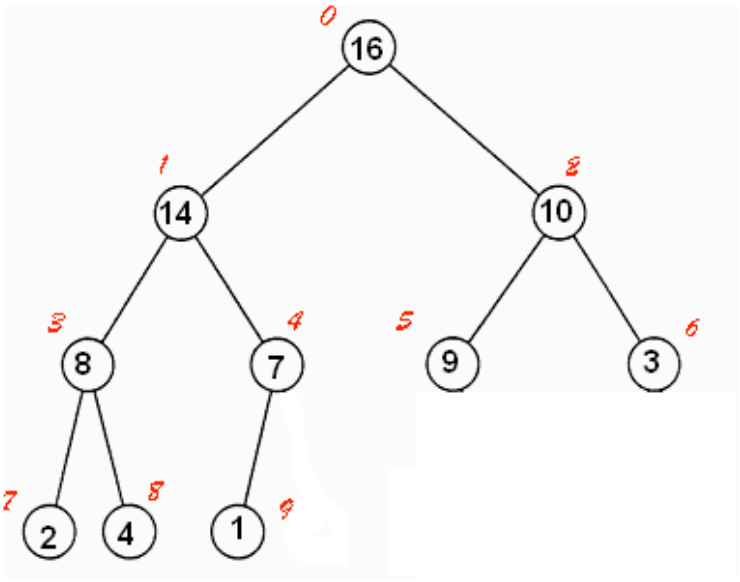
Heapsort

```
void heapsort(A[], int size)
{
	buildHeap(A, size); // build an initial heap from the sata

	   int index = size-1;
	   while(index >= 1) // for each node beginning with  the last leaf
	   {
			//switch the root with the last leaf
			int temp = A[0];
			A[0] = A[index];
			A[index] = temp;
			index--;

			//adjust the heap excluding the leaves with the largest values
			size--;
			heapify(A,size,0);
	   }
}
```

HeapSort

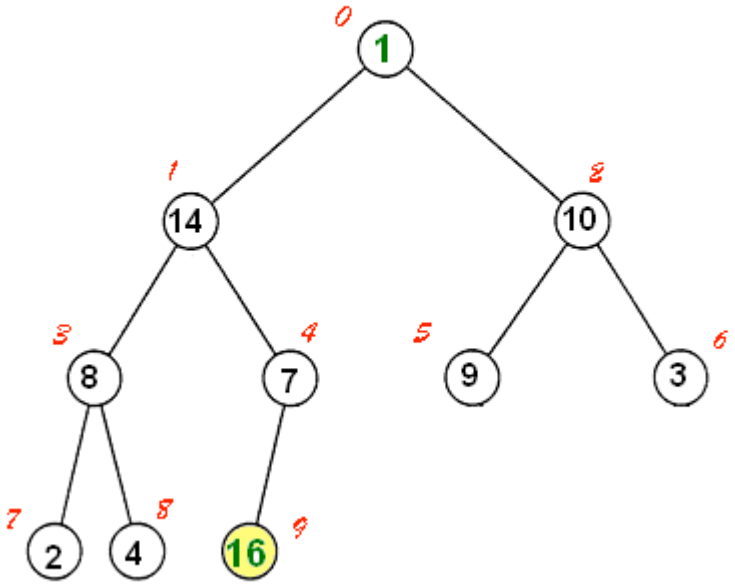
Problem :
Sort the list (array)
16,14,10,8,7,9,3,2,4,1

Build an initial heap



0	16
1	14
2	10
3	8
4	7
5	9
6	3
7	2
8	4
9	1

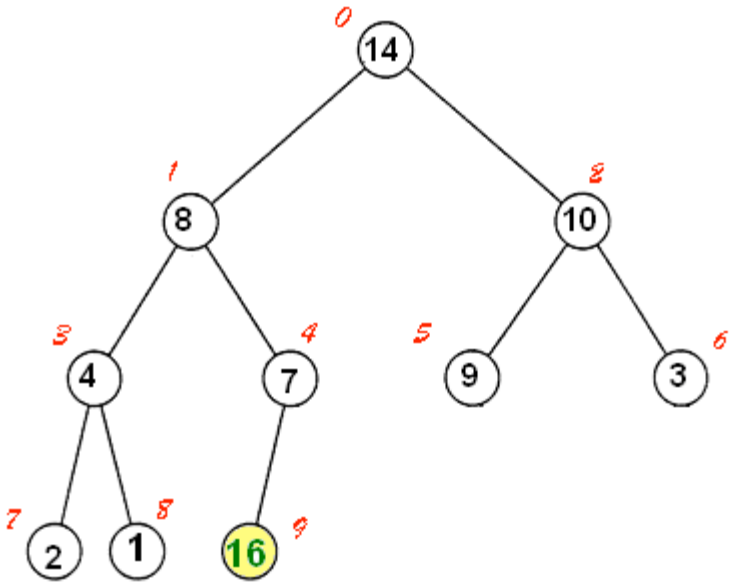
Swap first and last elements



0	1
1	14
2	10
3	8
4	7
5	9
6	3
7	2
8	4
9	16

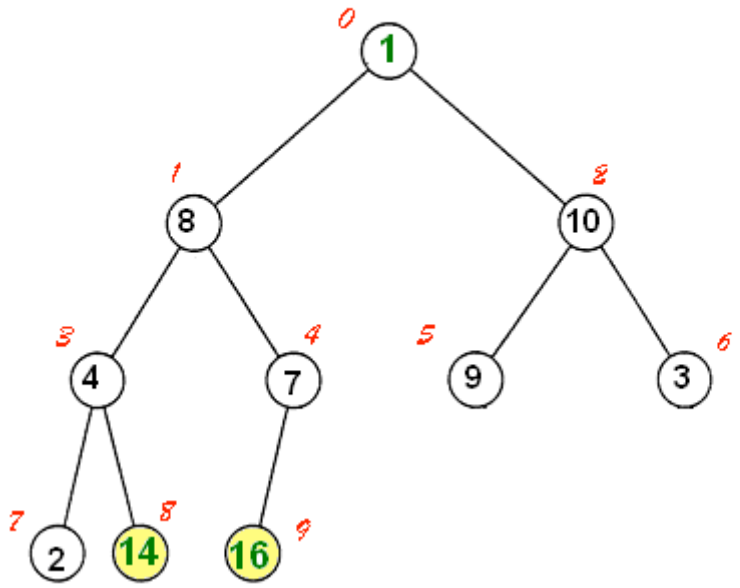
Adjust the heap: Heapafy(S,9,0)

Do not include node 9



0	14
1	8
2	10
3	4
4	7
5	9
6	3
7	2
8	1
9	16

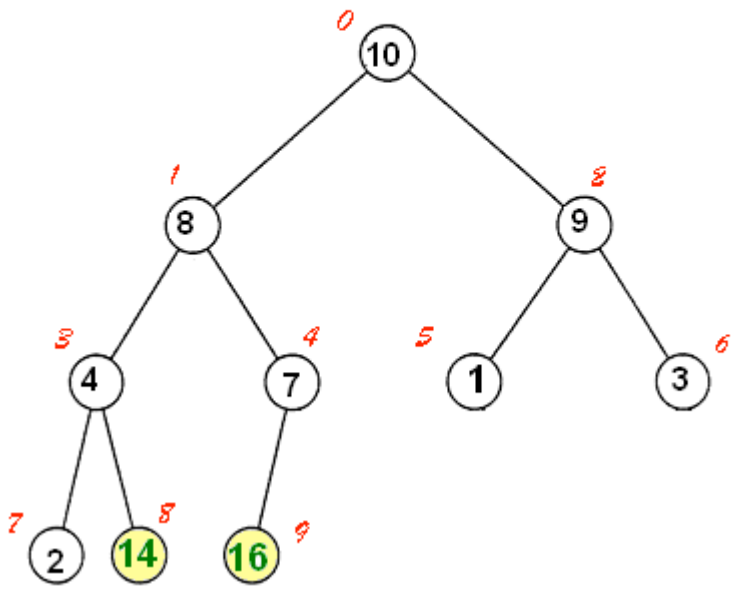
Swap first and last elements



0	1
1	8
2	10
3	4
4	7
5	9
6	3
7	2
8	14
9	16

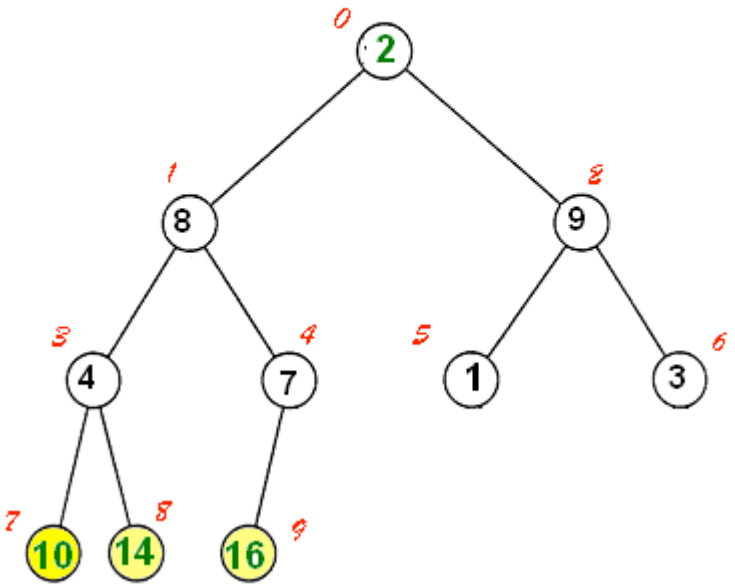
Adjust the heap: Heapafy(S,8,0)

Do not include nodes 9 and 8



0	10
1	8
2	9
3	4
4	7
5	1
6	3
7	2
8	14
9	16

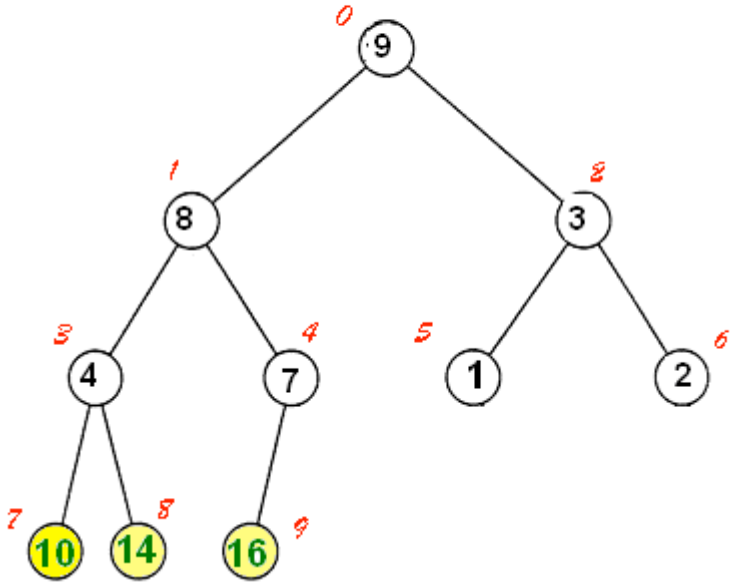
Swap first and last elements



0	2
1	8
2	9
3	4
4	7
5	1
6	3
7	10
8	14
9	16

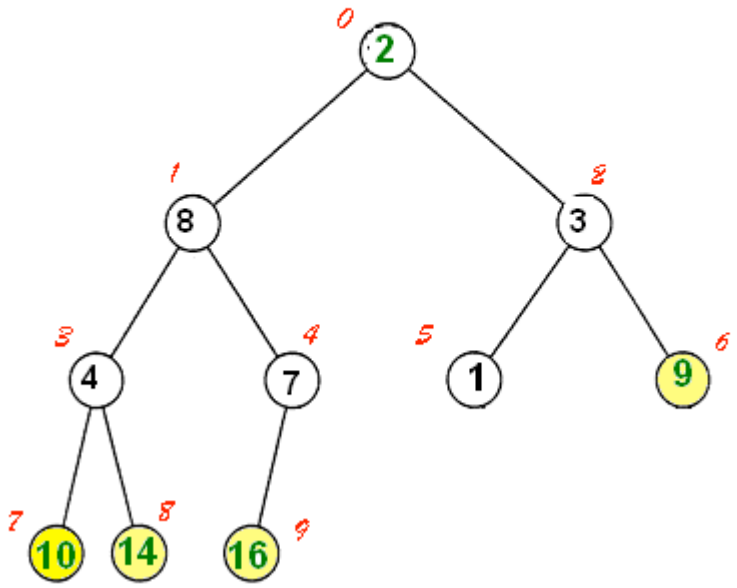
Adjust the heap: Heapafy(S,7,0)

Do not include nodes 9, 8, and 7



0	9
1	8
2	3
3	4
4	7
5	1
6	2
7	10
8	14
9	16

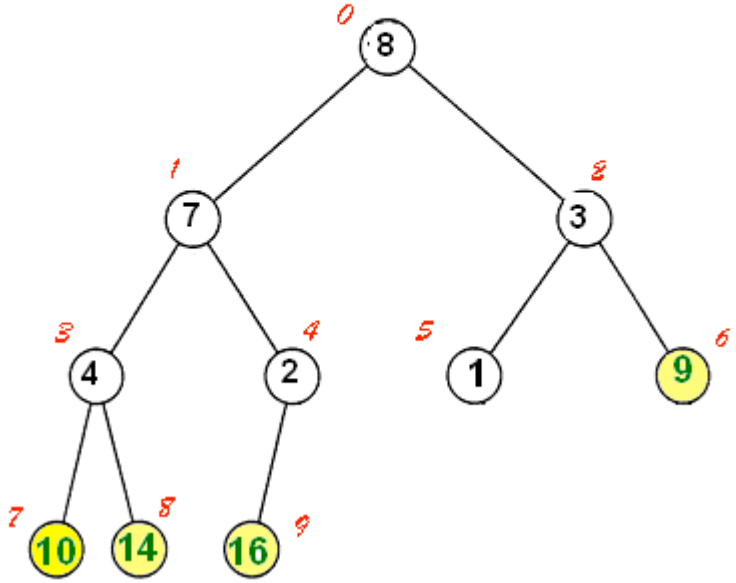
Swap first and last elements



0	2
1	8
2	3
3	4
4	7
5	1
6	9
7	10
8	14
9	16

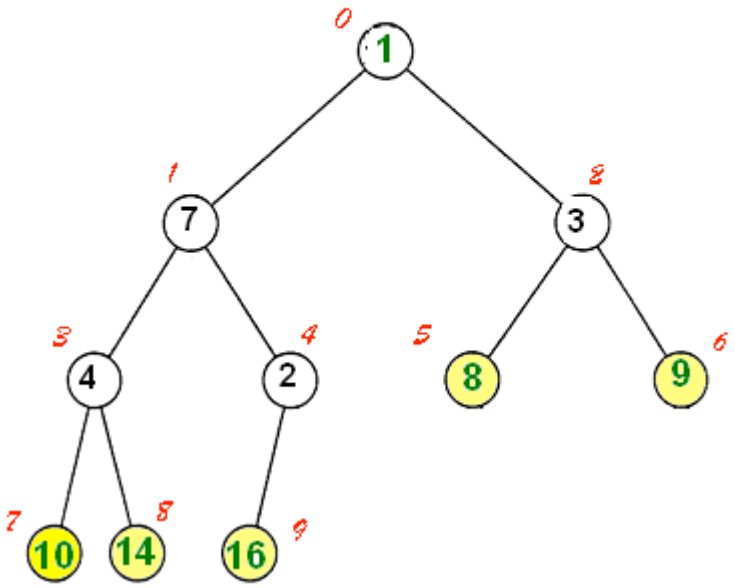
Adjust the heap: Heapafy(S,7,0)

Do not include nodes 9, 8, 7 and 6



0	8
1	7
2	3
3	4
4	2
5	1
6	9
7	10
8	14
9	16

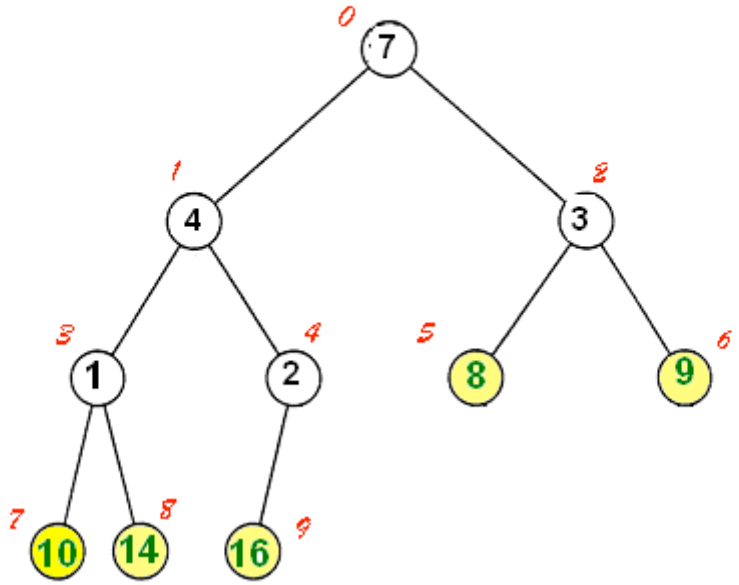
Swap first and last elements



0	1
1	7
2	3
3	4
4	2
5	8
6	9
7	10
8	14
9	16

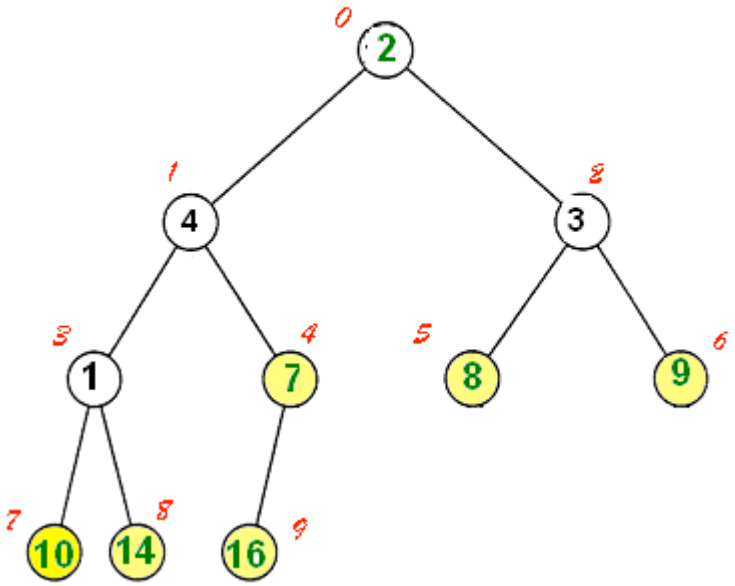
Adjust the heap: Heapafy(S,7,0)

Do not include nodes 9, 8, 7 ,6, and 5



0	7
1	4
2	3
3	1
4	2
5	8
6	9
7	10
8	14
9	16

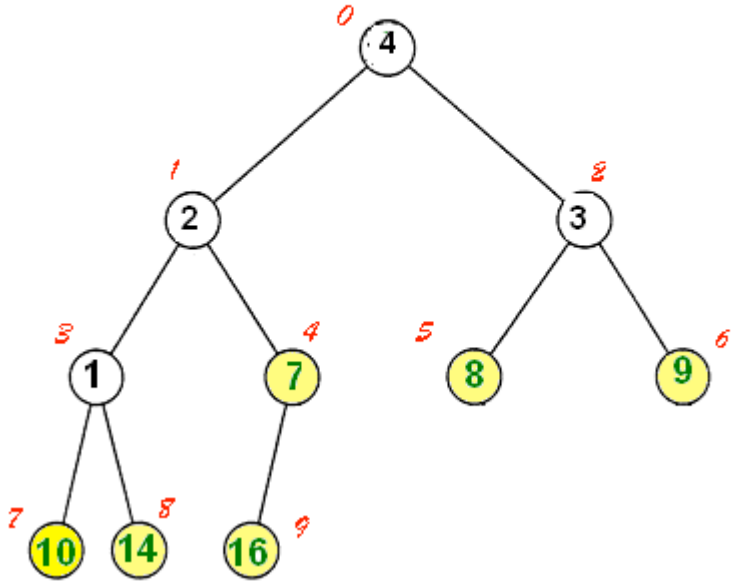
Swap first and last elements



0	2
1	4
2	3
3	1
4	7
5	8
6	9
7	10
8	14
9	16

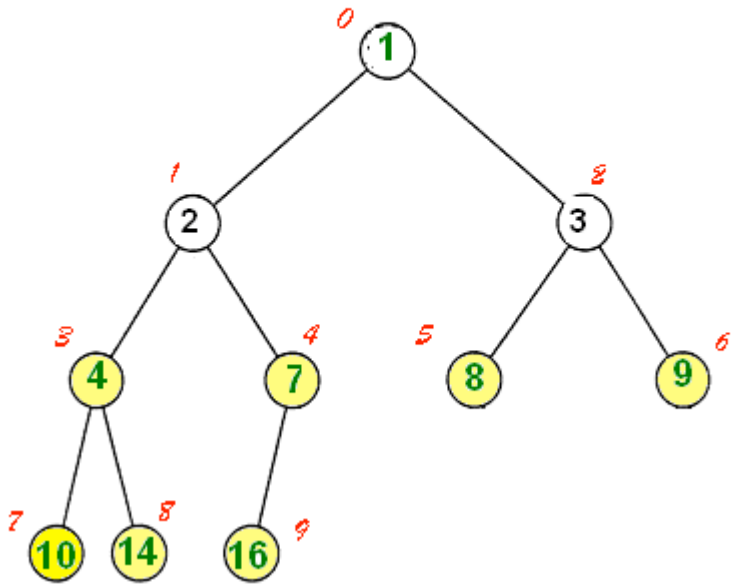
Adjust the heap: Heapafy(S,7,0)

Do not include nodes 9, 8, 7 ,6, 5, and 4



0	4
1	2
2	3
3	1
4	7
5	8
6	9
7	10
8	14
9	16

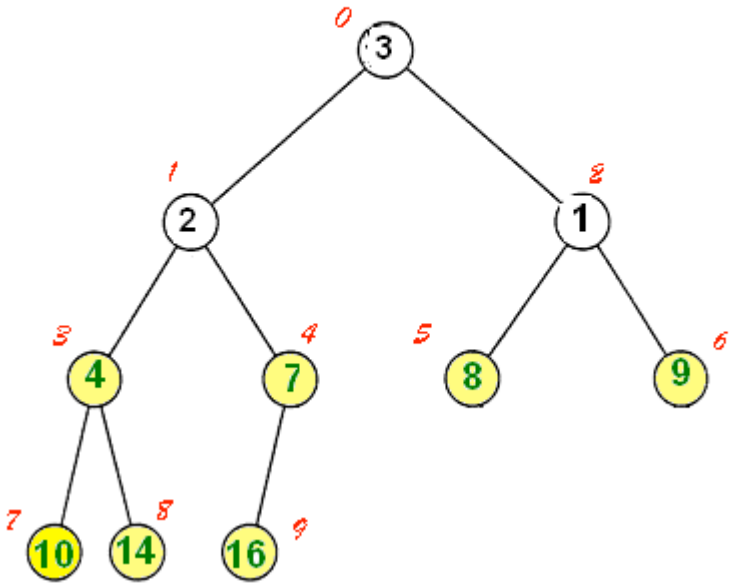
Swap first and last elements



0	1
1	2
2	3
3	4
4	7
5	8
6	9
7	10
8	14
9	16

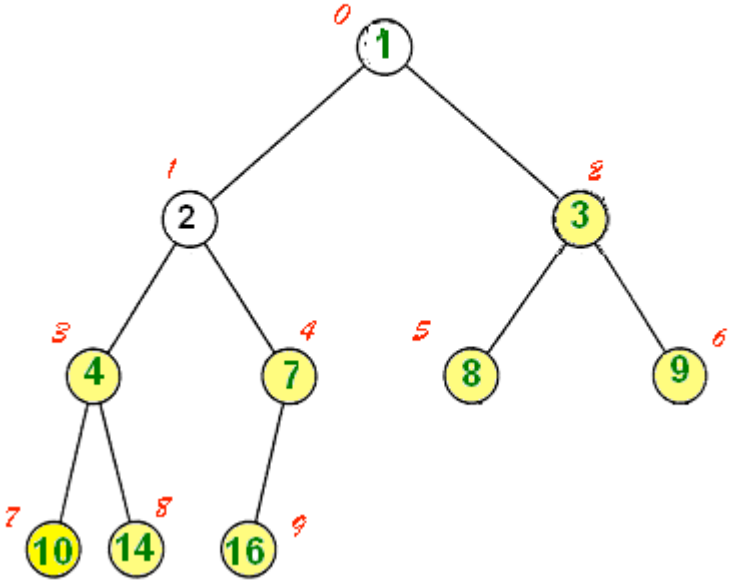
Adjust the heap: Heapafy(S,7,0)

Do not include nodes 9, 8, 7 ,6, 5, 4 and 3



0	3
1	2
2	1
3	4
4	7
5	8
6	9
7	10
8	14
9	16

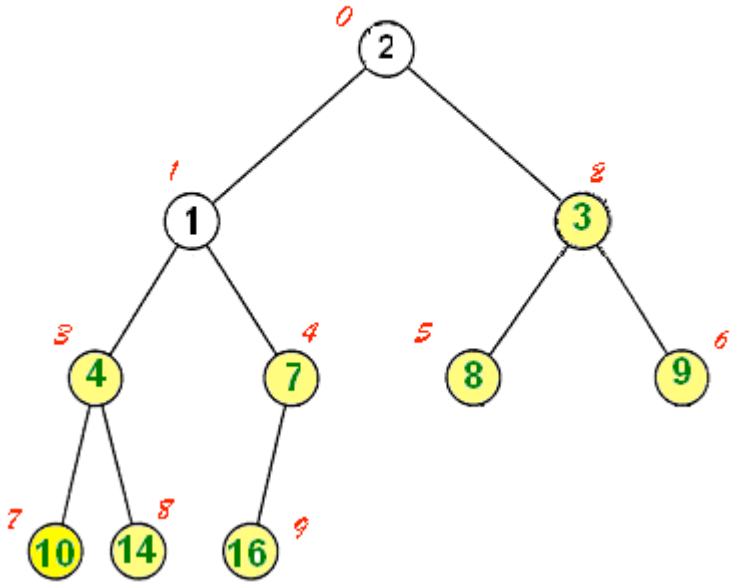
Swap first and last elements



0	1
1	2
2	3
3	4
4	7
5	8
6	9
7	10
8	14
9	16

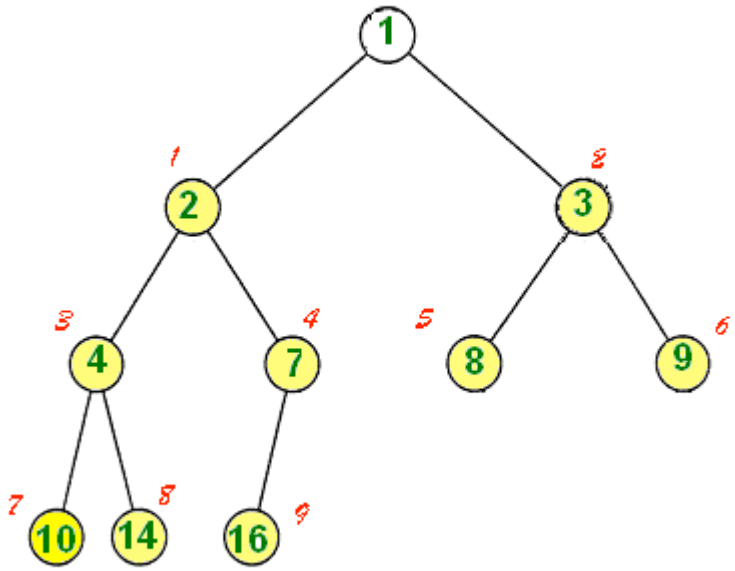
Adjust the heap: Heapafy(S,7,0)

Do not include nodes 9, 8, 7 ,6, 5, 4 , 3, and 2



0	2
1	1
2	3
3	4
4	7
5	8
6	9
7	10
8	14
9	16

Swap first and last elements



0	1
1	2
2	3
3	4
4	7
5	8
6	9
7	10
8	14
9	16

DONE