DET102 Data Structures and Algorithms

Lecture 10: Heap

Queue and Priority Queue

- Queue
  - First in, first out
  - based on the entering order
- Priority Queue
  - Item with the highest priority
  - based on the priority value
  - implementation
    - binary search tree (how to balance)
    - heap

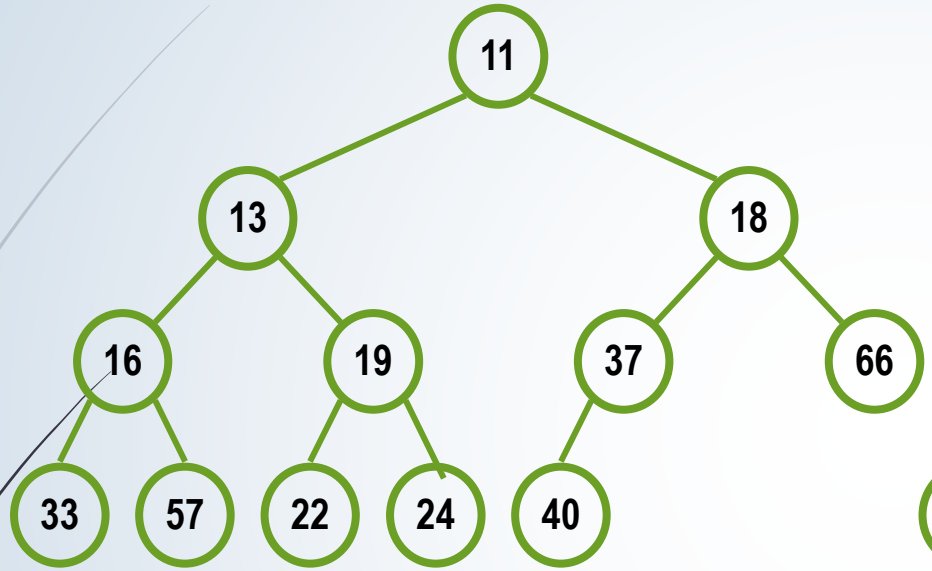
Heap

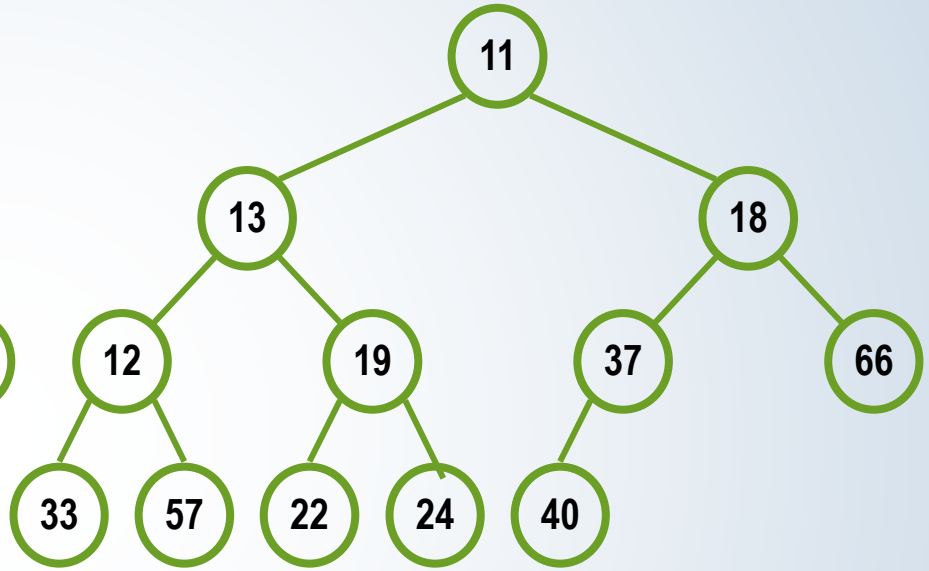
The tree is completely filled on all levels with a possible exception where the lowest level is filled from left to right.

- Structure Property
  - A complete (or simple complete) binary tree.
- Heap-order Property
  - The data in the node is greater/smaller than the data in all the descendants of the node.
  - Max Heap: max value is stored in the root
  - Min Heap: min value is stored in the root.

In the remaining slides, we will study *Min Heap*.

Examples





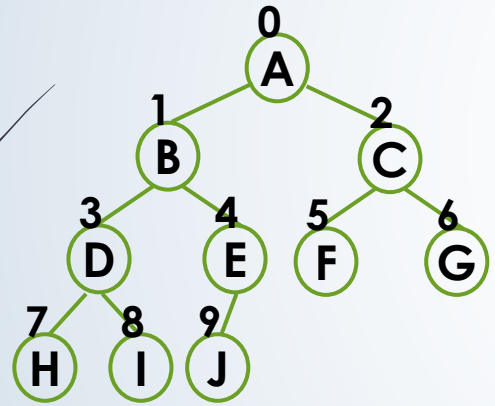
Which one is a min heap ?

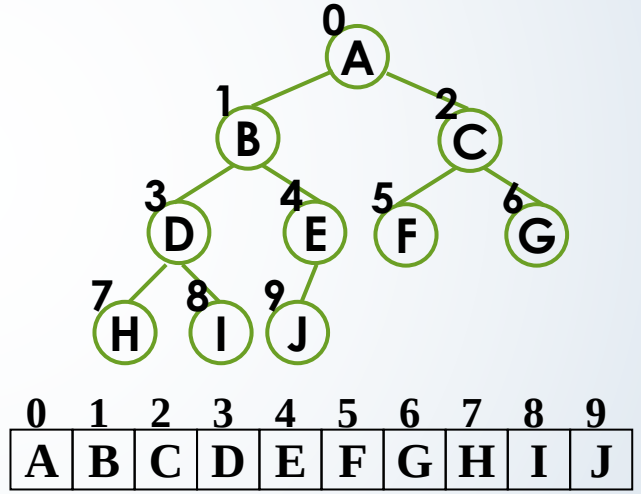
Would you choose array implementation or linked implementation ?

Array Representation of Binary Tree

Array Representation of Binary Tree

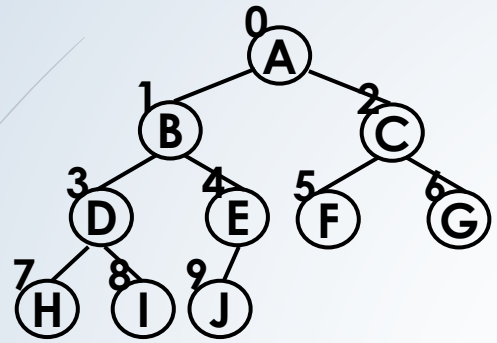
A numbering scheme:

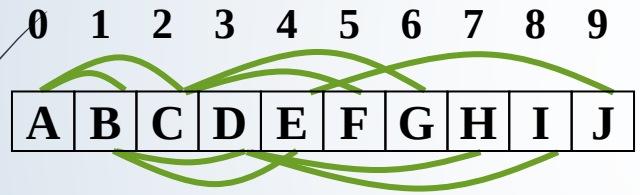




We can **represent binary trees using array** by applying this numbering scheme.

Array Representation of Binary Tree





Children of a node at slot ***i***:

Left(i) = 2i+1
Right(i) = 2i+2

Parent of a node at slot ***i***:

Parent(i) = ⌊ (i-1)/2 ⌋

⌊x⌋: **"Floor" The greatest integer less than x**

⌈x⌉: **"Ceiling" The least integer greater than x**

For any slot ***i***,

If ***i*** is odd: it represents a left son.

If ***i*** is even (but not zero): it represents a right son.

The node at the right of the represented node of ***i*** (if any), is at ***i***+1.

The node at the left of the represented node of ***i*** (if any), is at ***i***-1.

$\lfloor \rfloor$ is the floor function.

Array Implementation

- When we say heap, we involve only Min Heap.
- Notations
  - Length[H]: the number of elements in an array H which is the length
  - Heap-size[H]: the total number of elements stored in the heap.
  - If heap-size[H]< length[H], then array H may contain valid elements but these elements are not in the heap.

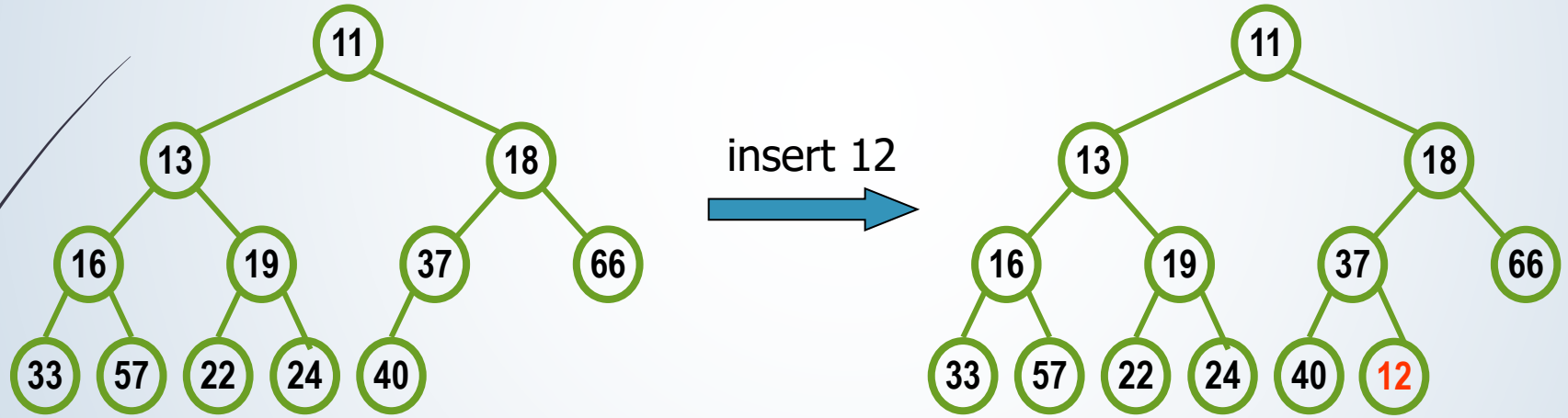
Parent(k) is at $H[\lfloor (k-1)/2 \rfloor]$,	if $k \neq 0$
Lchild(k) is at H[2k+1],	if $2k+1 \leq n-1$
Rchild(k) is at H[2k+2],	if $2k+2 \leq n-1$

Insert

To insert an item

Step 1: Put the item at the first empty slot in the array

Step 2: Adjust the heap to satisfy heap-order property (float up)

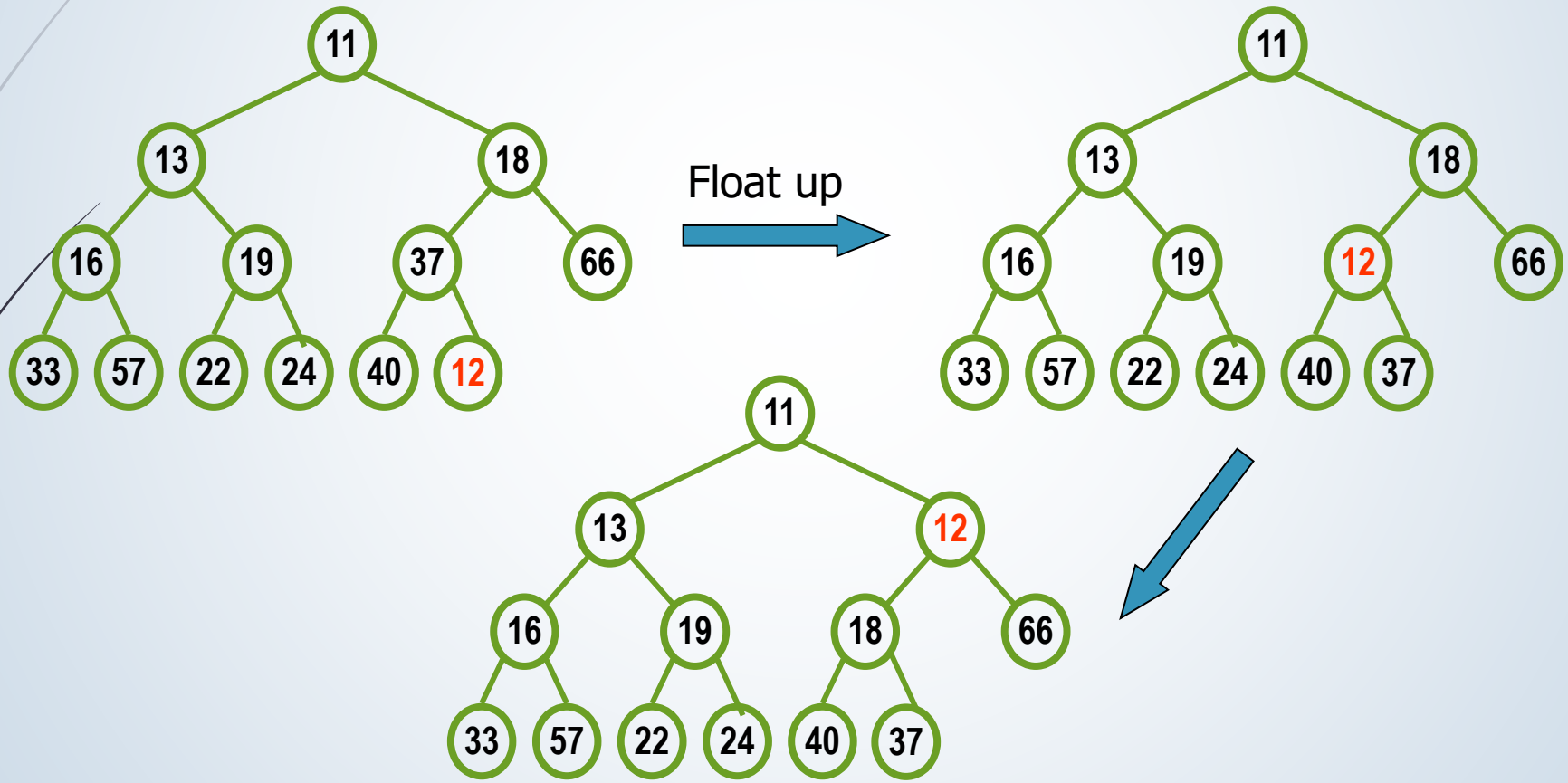


Insert

To insert an item

Step 1: Put the item at the first empty slot in the array

Step 2: Adjust the heap to satisfy heap-order property (float up)

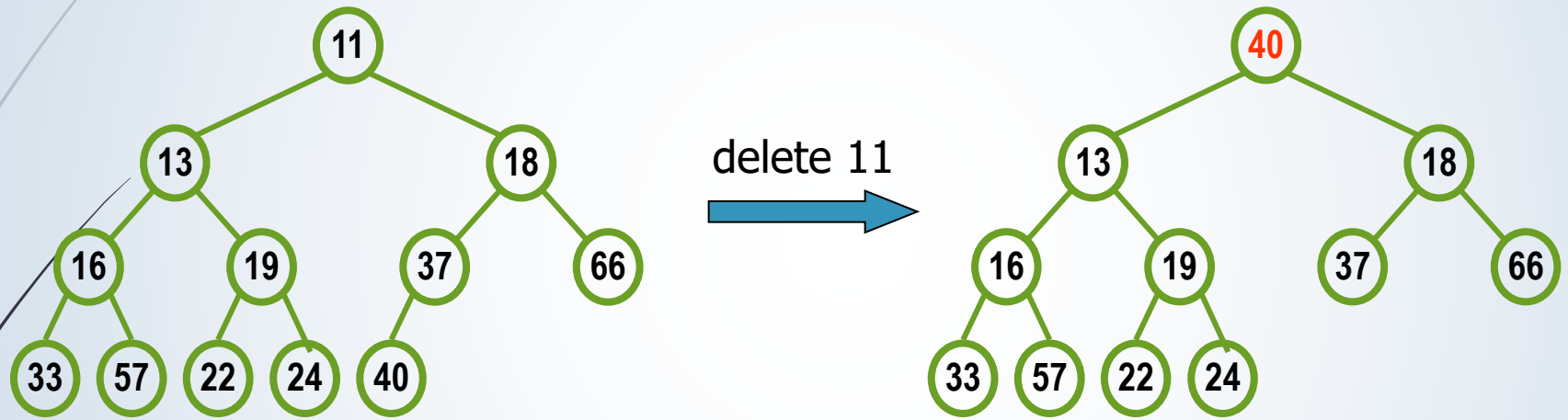


Delete

To delete an item

Step 1: Copy the last data to the root

Step 2: Adjust the heap to satisfy heap-order property (sink)

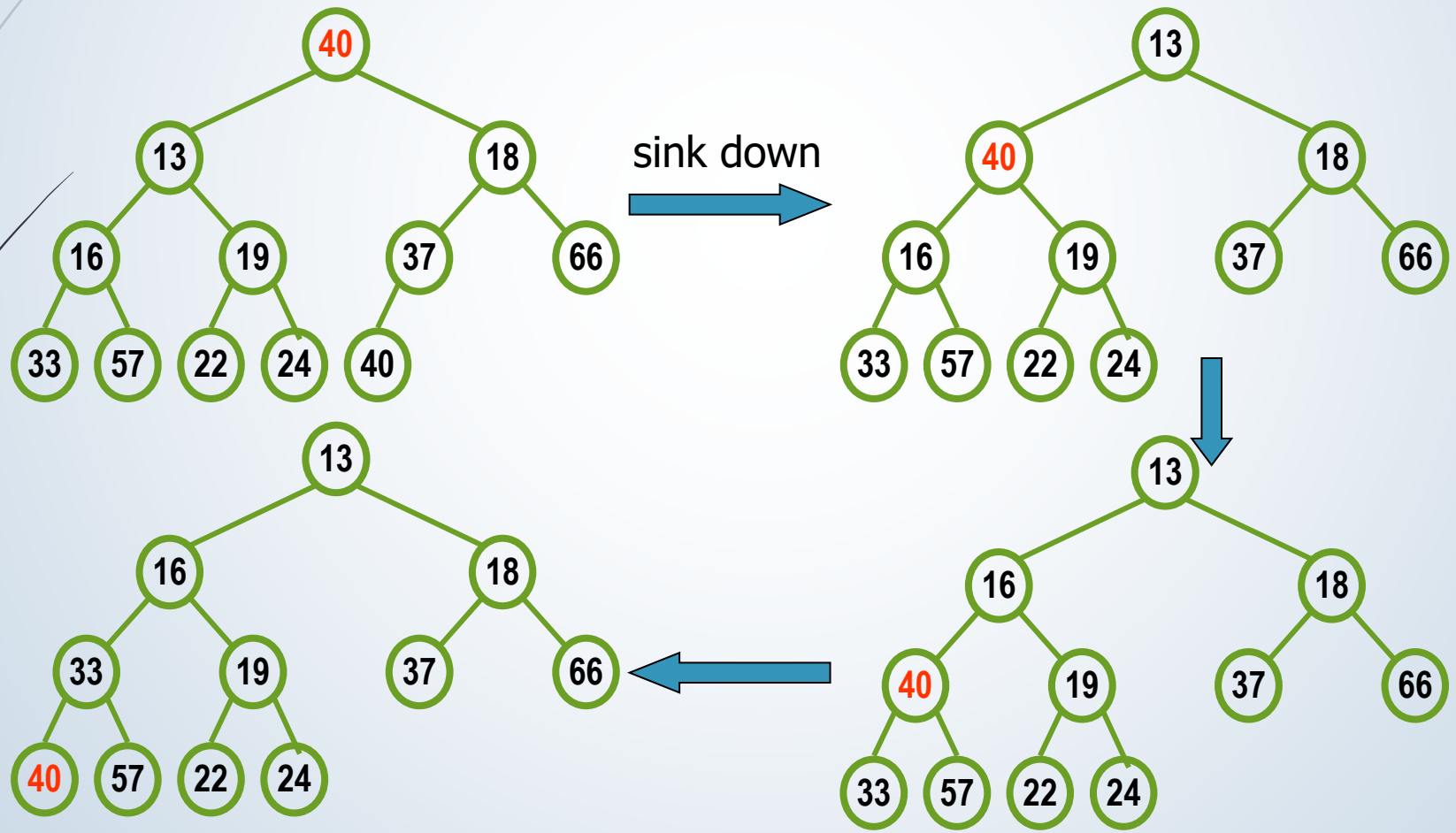


Delete

To delete an item

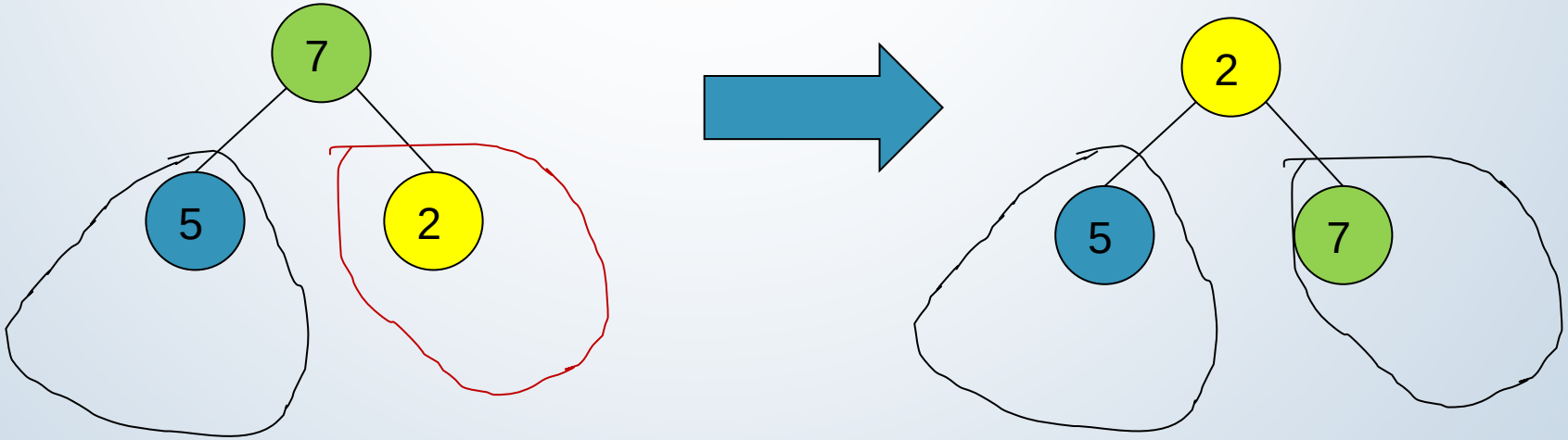
Step 1: Copy the last data to the root

Step 2: Adjust the heap to satisfy heap-order property (sink)



Heapify

- To preserve the heap order, the value of all children of H[i] must be greater than or equal to the value of H[i].
- We assume that binary trees rooted at left and right children of H[i] are (min) heaps, but H[i] may violate the heap property. In this case, we push the value at H[i] down the heap until the tree rooted at H[i] is a heap.



Heapify

- Lemma: The heap order is preserved when Heapify is invoked recursively.
- Analysis
  - The children subtrees have at most size of 2n/3

    $T(n) \leq T(2n/3) + \Theta(1)$
  - Applying Master Theorem

    $T(n) = O(\log n)$

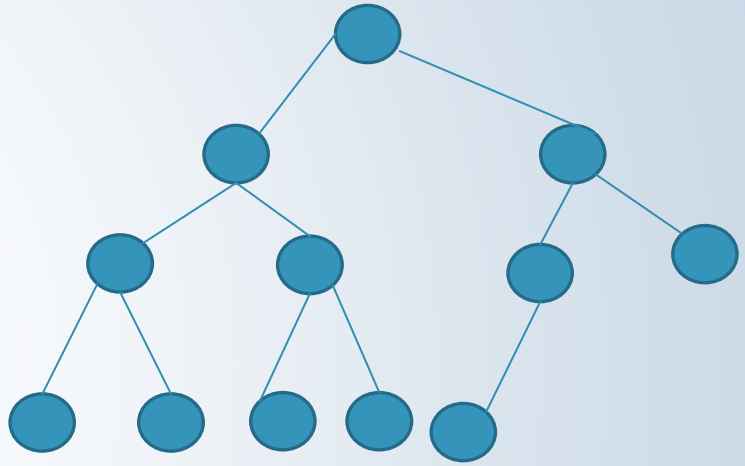
Build a Heap

- Step 1: Initialization

  set heap-size[H]←length[H]
- Step 2: fixing

  for i← [ ] down to 1

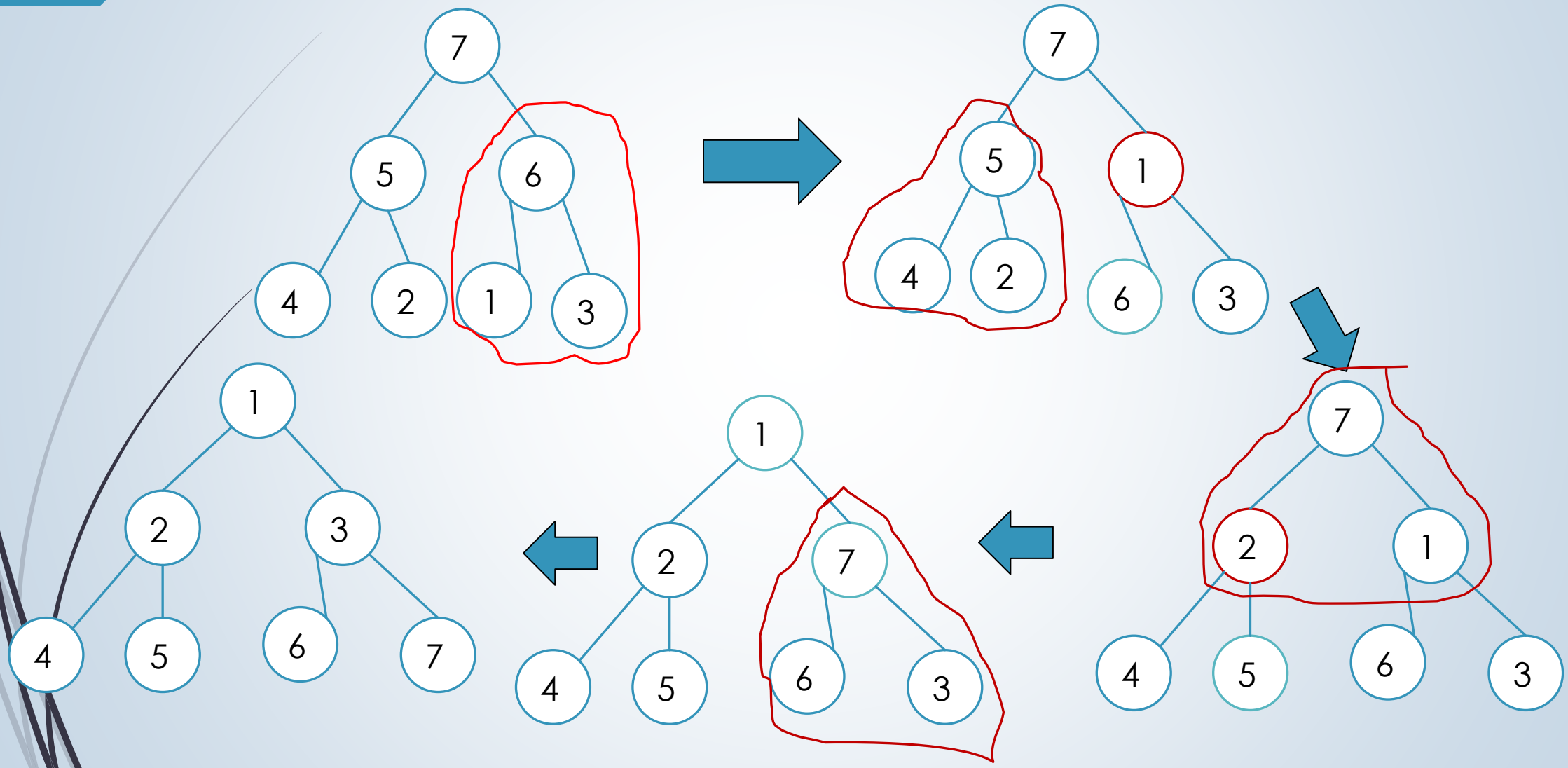
  call to Heapify(H,i)



Guess which value can be the starting value of i ?

Min-heap

Suppose the input numbers are 7,5,6,4,2,1,3



Heapsort

- The heap sort is accomplished by using a Build-heap for maintaining a (min) heap, and Heapify for fixing the heap.

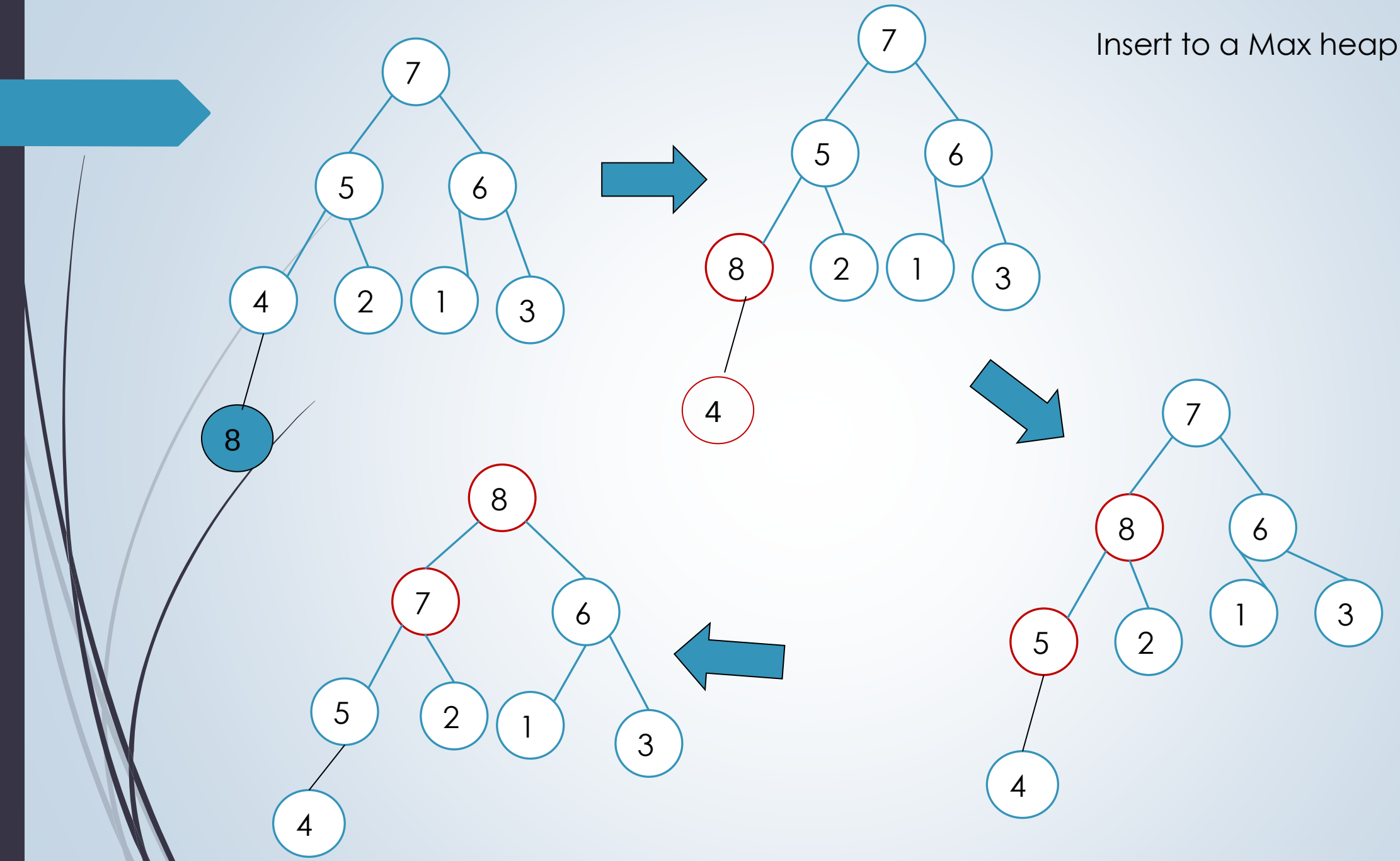
```
Step 1: Building
    call to Build-heap(H)
Step 2: loop, exchanging root, and fixing the heap
    for i←length[H] down to 1
    Swap H[0] and H[i]
    Set heap-size[H] ←heap-size[H]-1
    Call to Heapify(H,0)
Step 3: return at the point of call
     return
```

Exercise

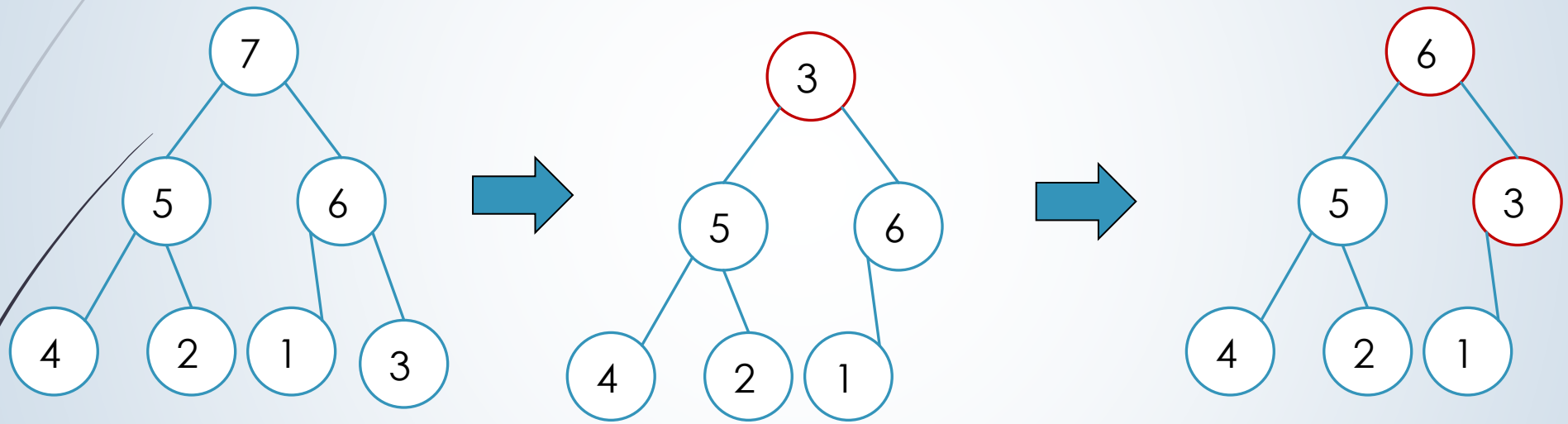
- Insert the following numbers into an empty Heap (Min-Heap)
  - 11, 13, 7, 5, 22, 3, 19, 1, 2
- Given an array of data, how to build a min-heap ?
  - 3,9,7,6,1,4,2,5

How about the max heap?

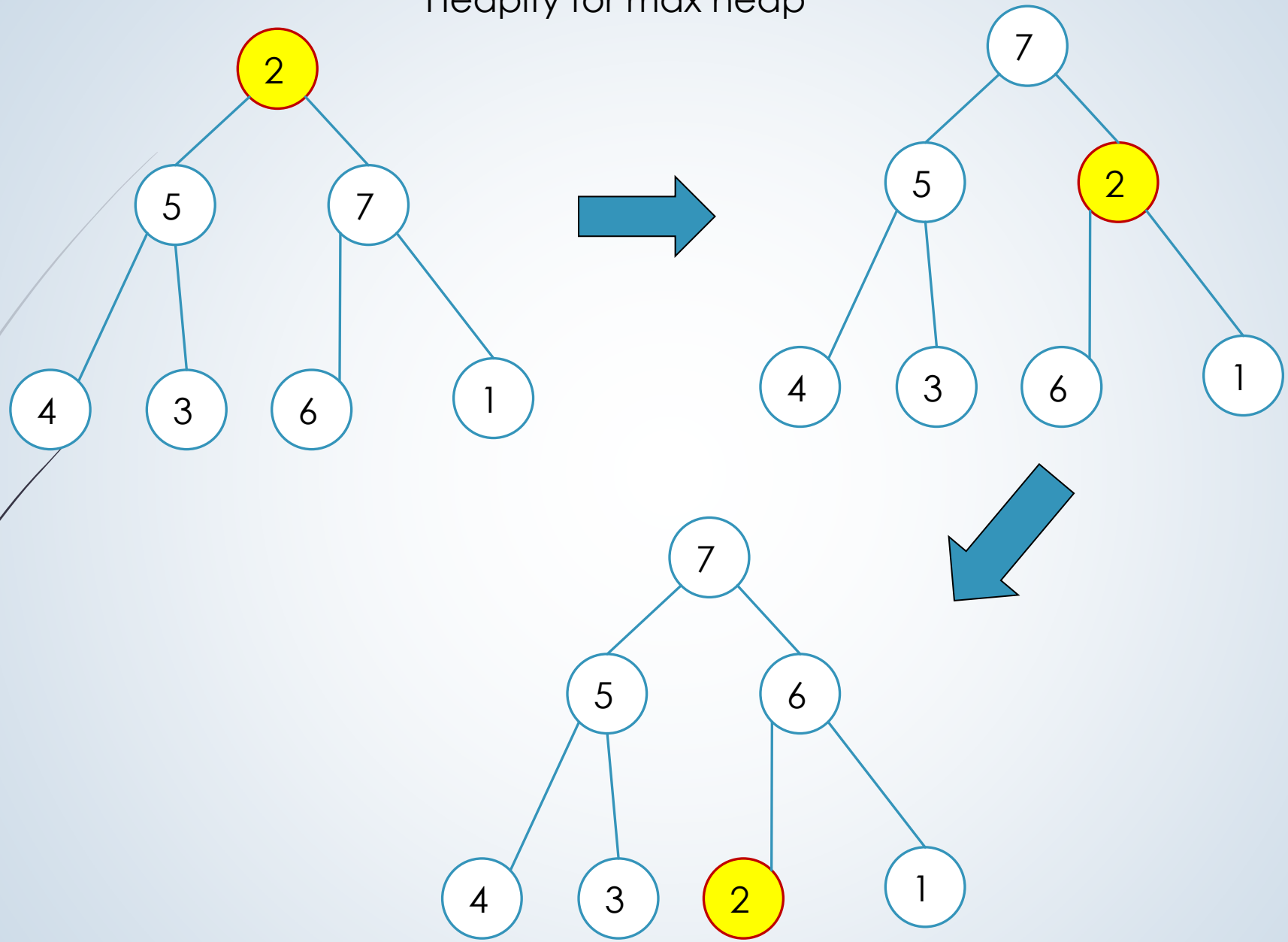
Insert to a Max heap



Delete from a Max Heap

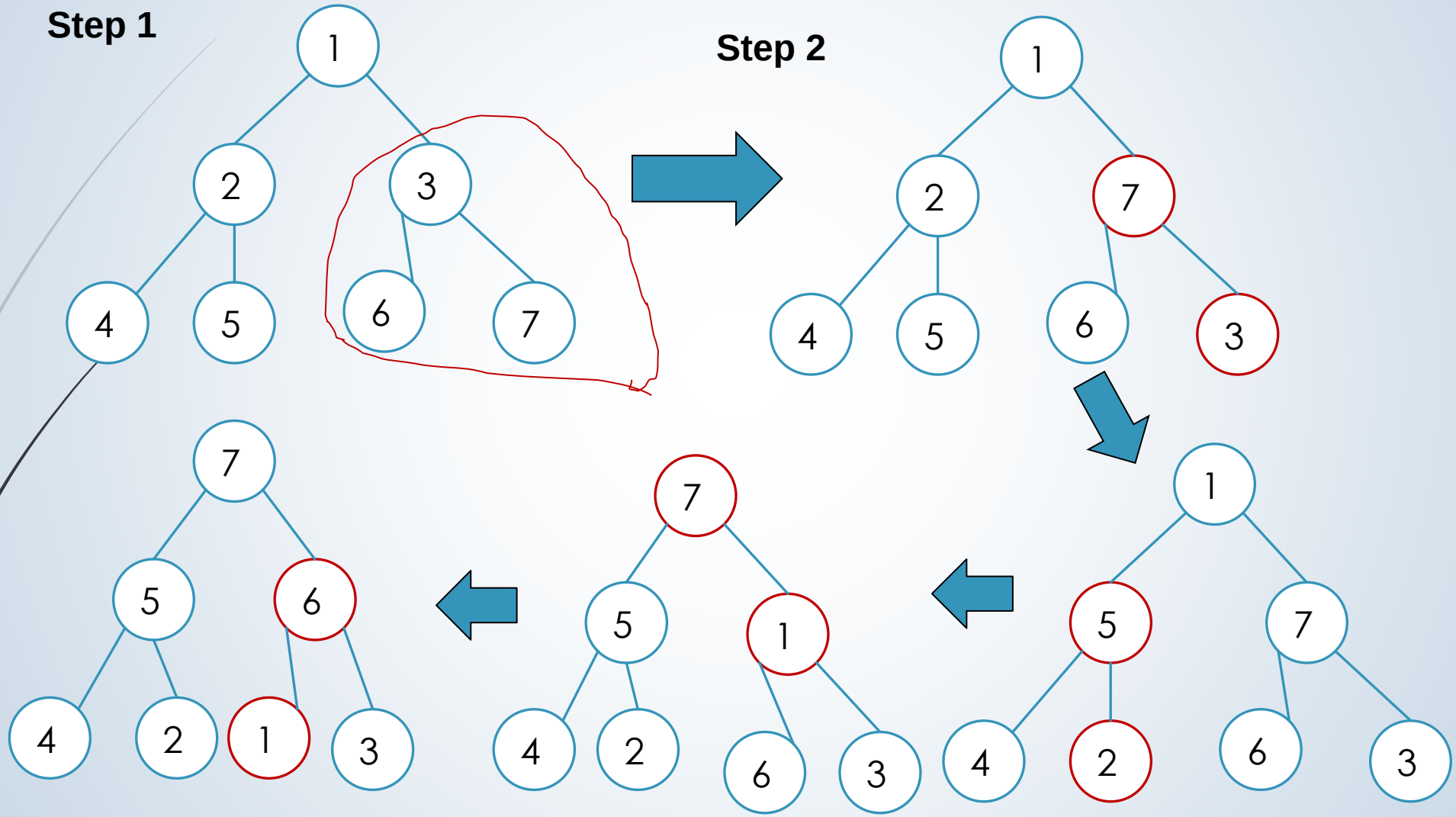


Heapify for max heap

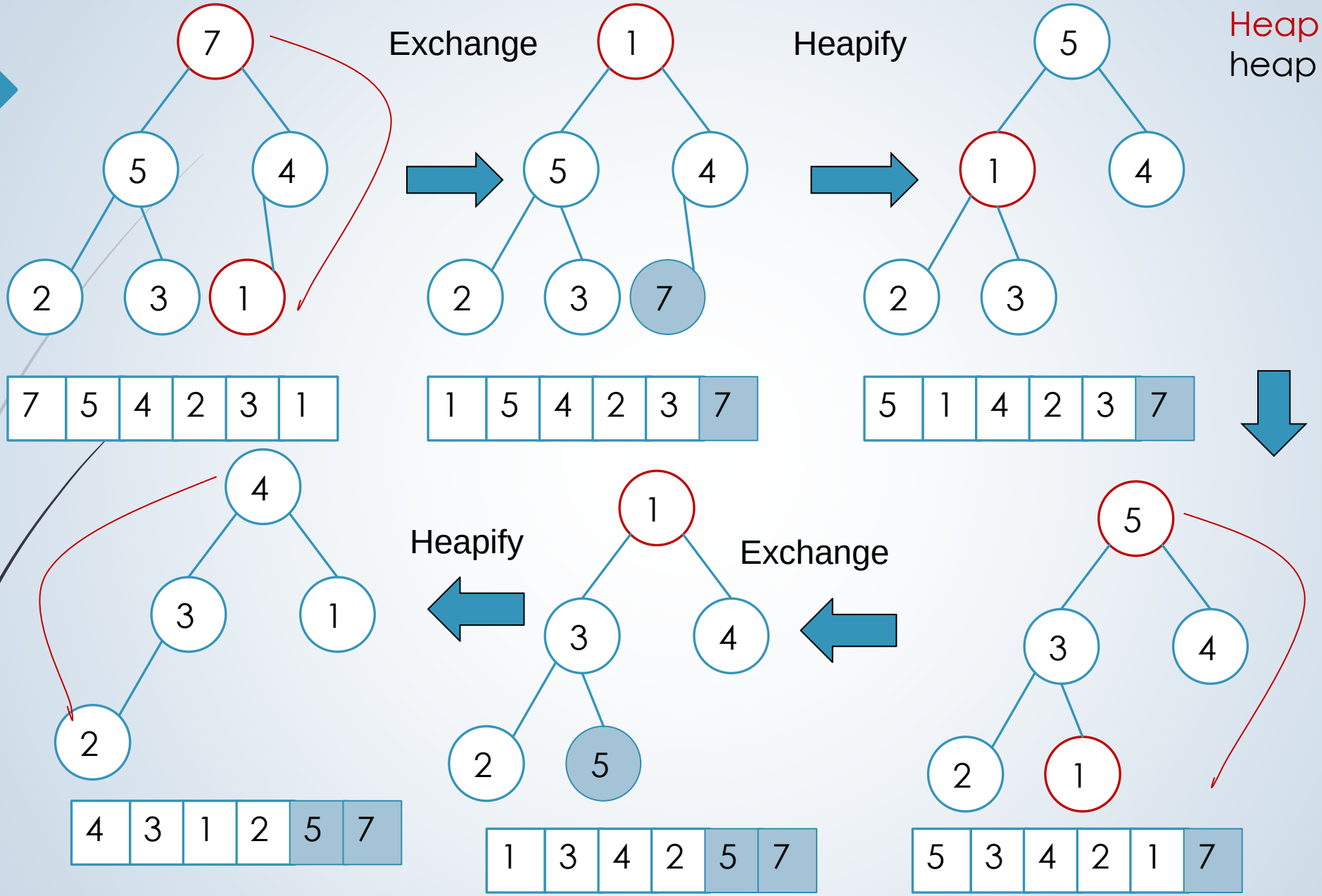


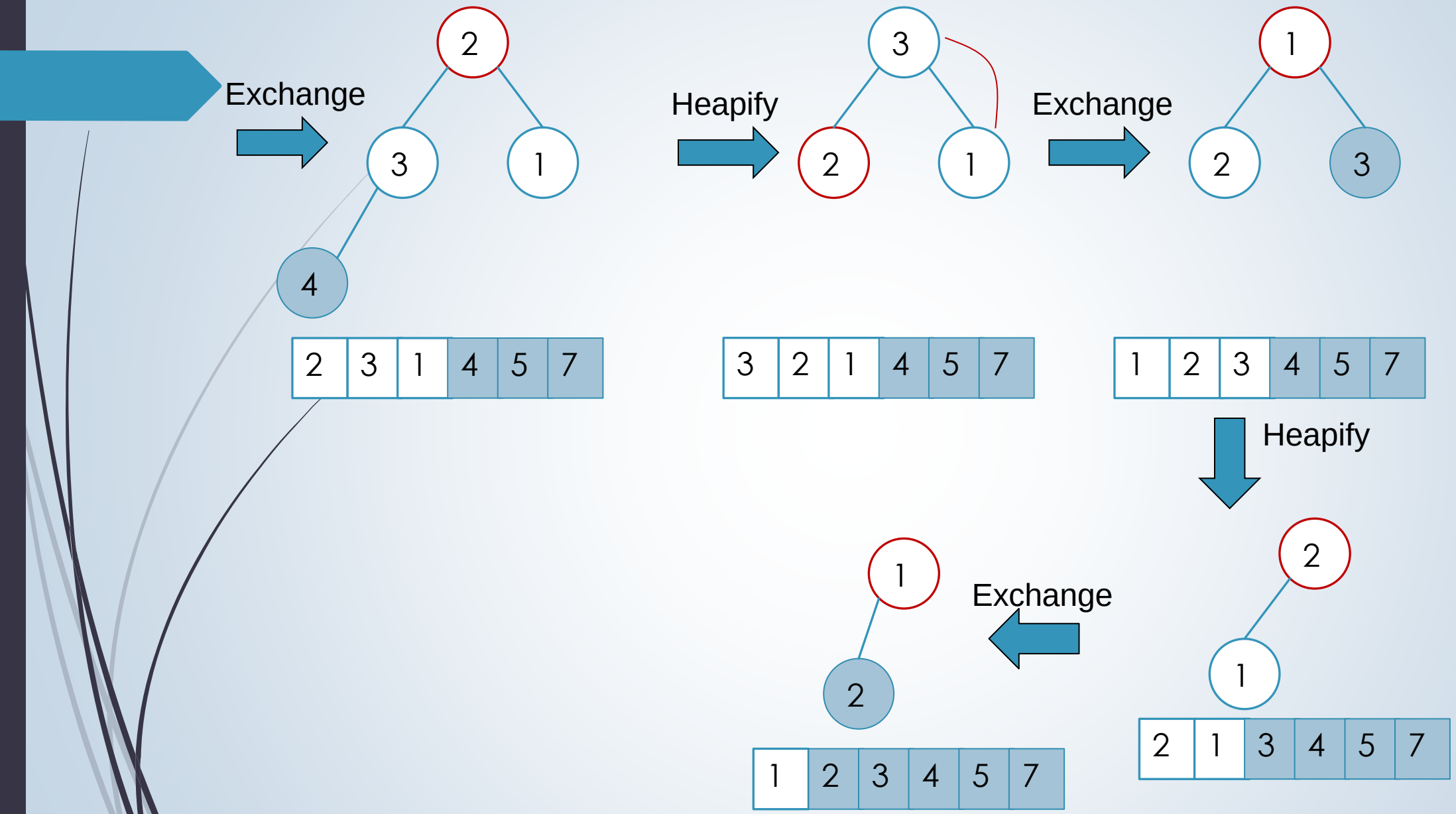
Build a max heap

Suppose the input numbers are 1,2,3,4,5,6,7



Heap sort using Max heap




Exchange
2
3
1
4
2 3 1 4 5 7
Heapify
3
2
1
3 2 1 4 5 7
Exchange
1
2
3
1 2 3 4 5 7
Heapify
2
1
2 1 3 4 5 7
Exchange
1
2
1 2 3 4 5 7

Heap Sort

- Algorithm:
  - Use a heap (max or min)
  - Every time, select the root item, adjust the heap.
  - The output sequence is sorted.
- Complexity:
  - O(nlogn)

Application of Heap

- Priority Queue
  - A priority queue is a data structure which maintains a set S of elements, each of with an associated value (key), and supports the following operations:
    - insert(S, k): insert an element k into the set S
    - extractMax(S): emove and return the element of S with the largest key

Complexity

- Heap
  - Heapify: O(log n)
  - Insert: O(log n)
  - Delete: O(log n)
- Priority Queue
  - insert: O(log n)
  - extractMax: O(log n)

Priority Queue in STL

main.cpp

```cpp
#include <iostream>
#include<queue>
using namespace std;

int main()
{
    priority_queue<int> PQ;
    PQ.push(1);
    PQ.push(8);
    PQ.push(3);
    PQ.push(5);

    cout<<PQ.top()<<" ";
    PQ.pop();

    cout<<PQ.top()<<" ";
    PQ.pop();

    PQ.push(11);

    cout<<PQ.top()<<" ";
    PQ.pop();

    cout<<PQ.top()<<" ";
    PQ.pop();

    return 0;
}
```

Output:
8 5 11 3

If the data type of the elements in priority queue is int, max key has the highest priority by default.

push(): insert a new element to PQ
pop(): remove one element from the top.
top(): return the top element which is the one with highest priority (max key)