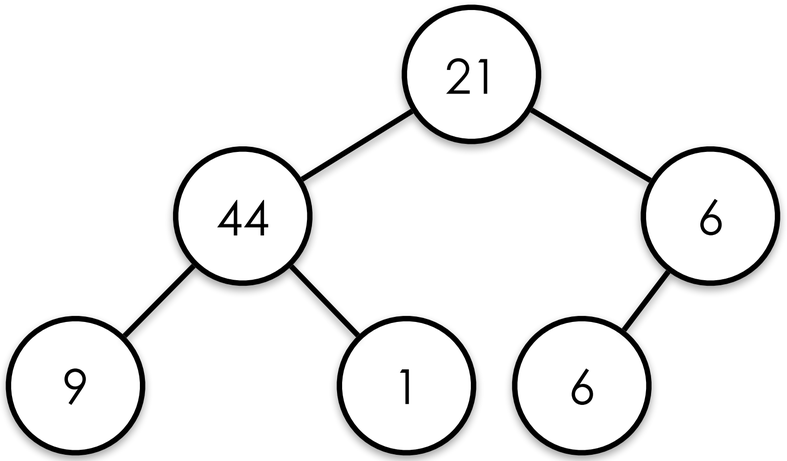
Heap

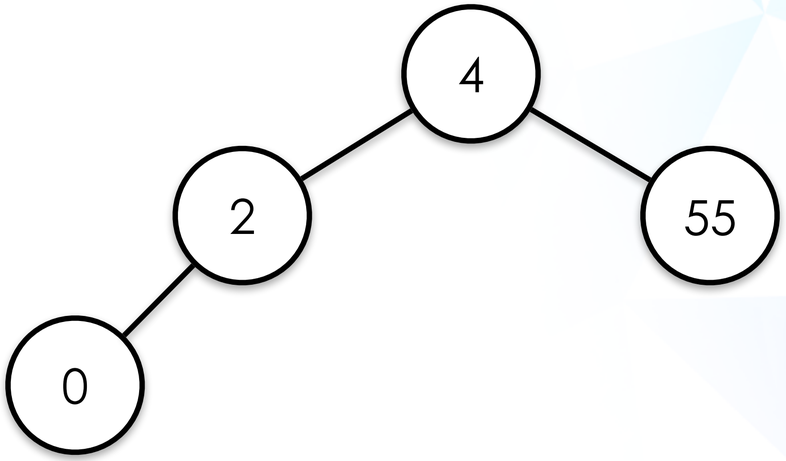
Tan Cher Wah (isstcw@nus.edu.sg)

Complete Binary Tree

A **Complete** binary tree is such that all its levels, except possibly the last level, are fully filled



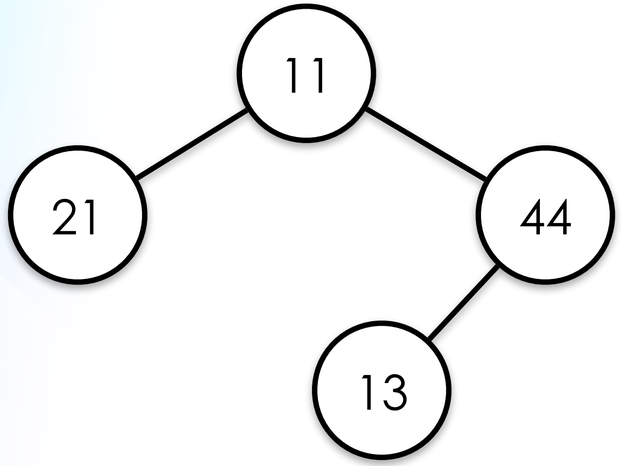
A Complete Binary Tree



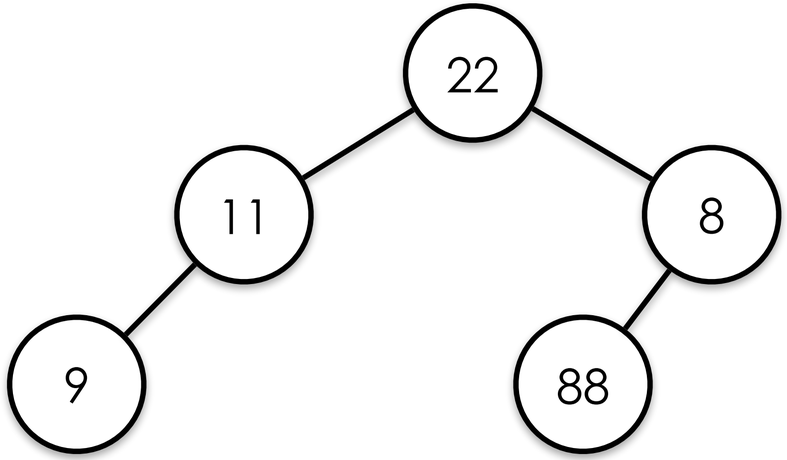
A Complete Binary Tree

Complete Binary Tree

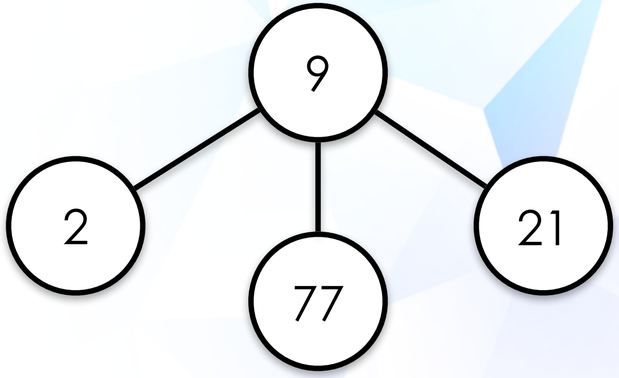
The followings are NOT complete binary trees



NOT Complete



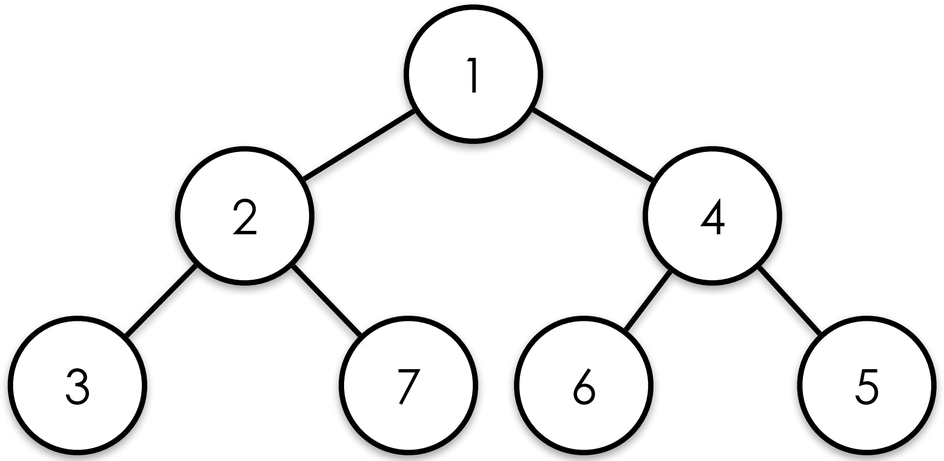
NOT Complete



NOT Binary

Min Heap

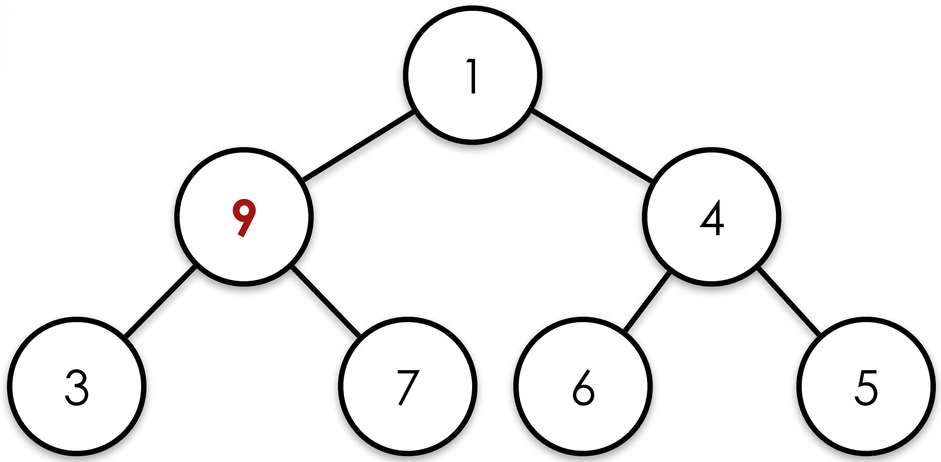
A Min Heap is a **complete binary tree** where each **parent** node has a **key** that is **smaller** than its **child** node(s), and the **root node** has the **smallest key**



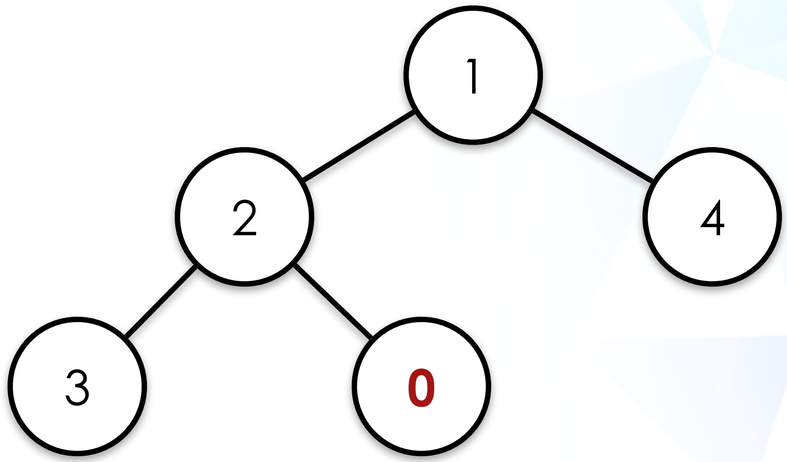
Min Heap

Min Heap

The followings are not Min Heaps



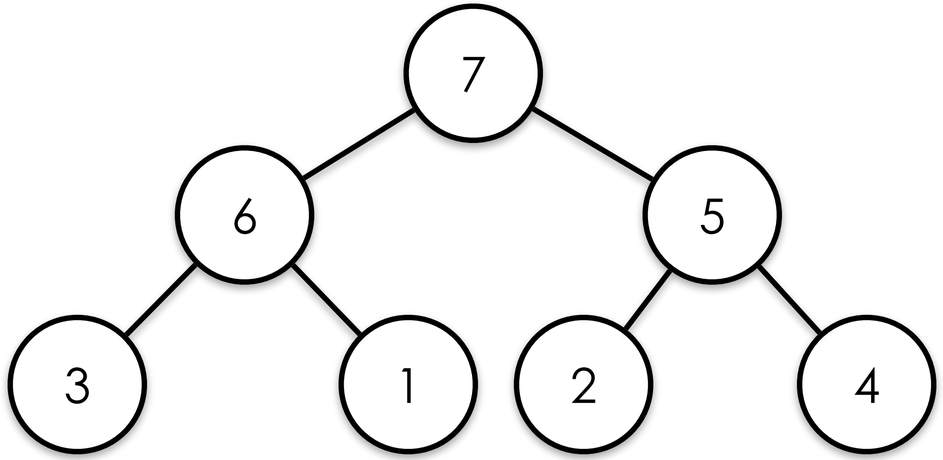
Node with key 9 is greater than child's key



Node with key 0 is smaller than parent's key

Max Heap

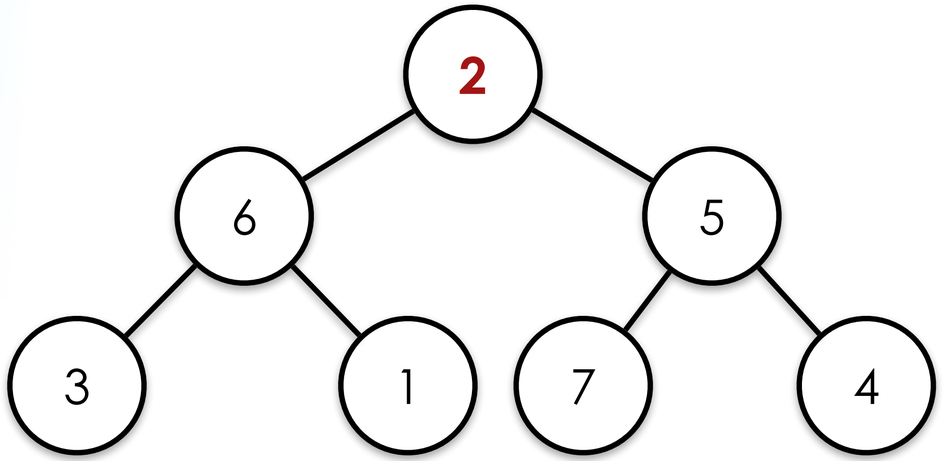
A Max Heap is a **complete binary tree** where each **parent** node has a **key** that is **larger** than its **child** node(s), and the **root node** has the **largest key**



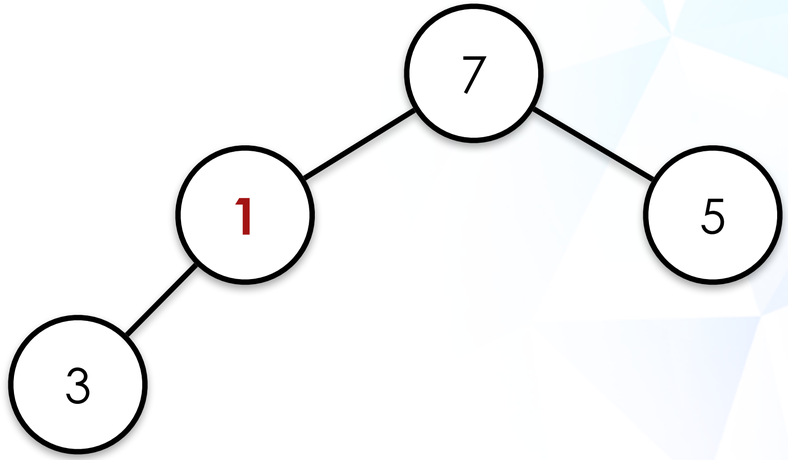
Max Heap

Max Heap

The followings are not Max Heaps



Node with key 2 is smaller than children's keys



Node with key 1 is smaller than its child's key

Common Heap Operations

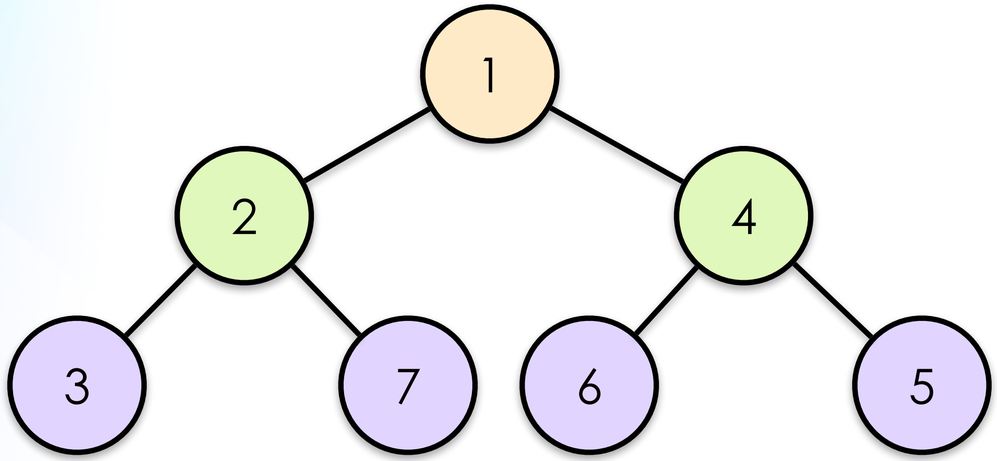
- **GET MIN** OR **MAX** ELEMENT
  - Get root element - **min key** for Min Heap
  - Get root element - **max key** for Max Heap
- **REMOVE MIN** OR **MAX** ELEMENT
  - Remove root element - **min key** for Min Heap
  - Remove root element - **max key** for Max Heap
- **INSERT ELEMENT**
  - Insert a new element while maintaining the property of a Heap

Applications

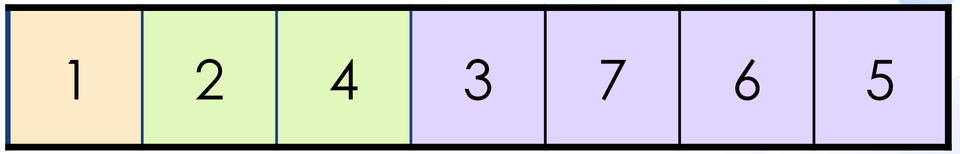
- Priority Queues (e.g. CPU job-scheduling)
  - Give each runnable task a priority number
  - Execute the next runnable task with highest priority
- Sorting (via HeapSort)
  - To order elements in ascending order, use Min Heap
  - To order elements in descending order, use Max Heap

Implementation

Use an Array to implement a Min Heap data structure



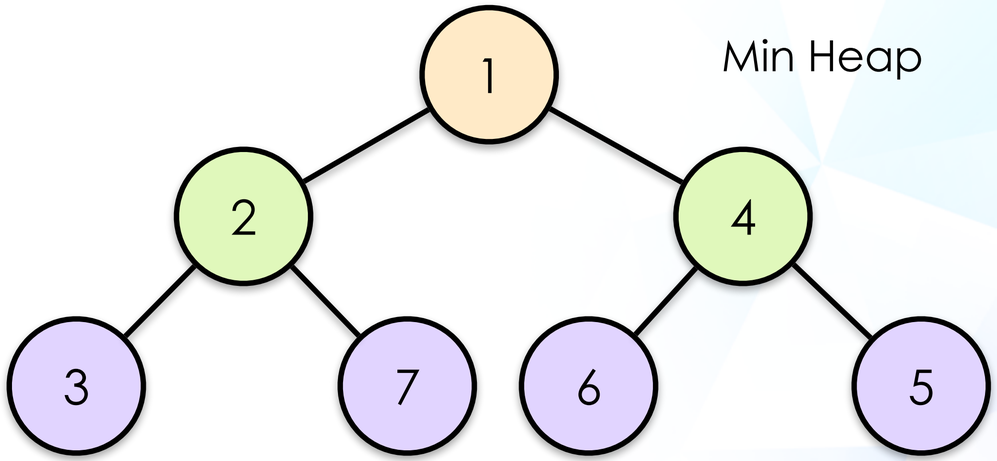
Min Heap

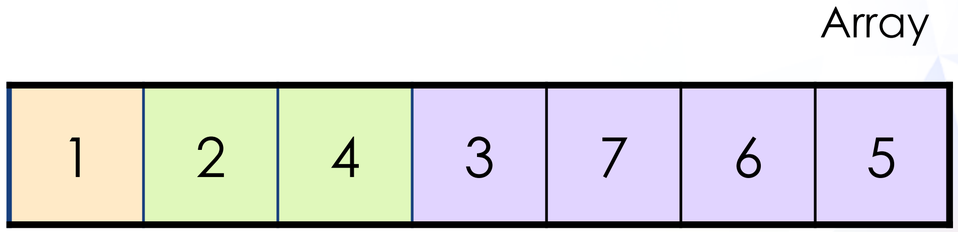


Array

Implementation

- i = index of array
- Parent node: i
  - Left node: 2 * i + 1
  - Right node: 2 * i + 2
- Find a node's parent: (i - 1) / 2

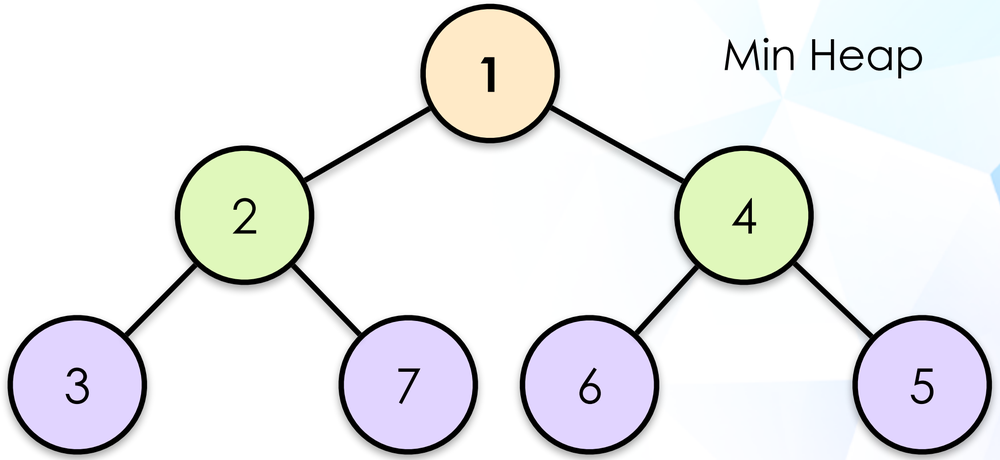




Example

How do we get the child nodes of Array[0]?

- Current array index = 0
- Applying the formula,
  - Left child-node index = 2 * 0 + 1 = 1
  - Right child-node index = 2 * 0 + 2 = 2
- Hence,
  - Left child-node value = Array[1] = 2
  - Right child-node value = Array[2] = 4



Min Heap

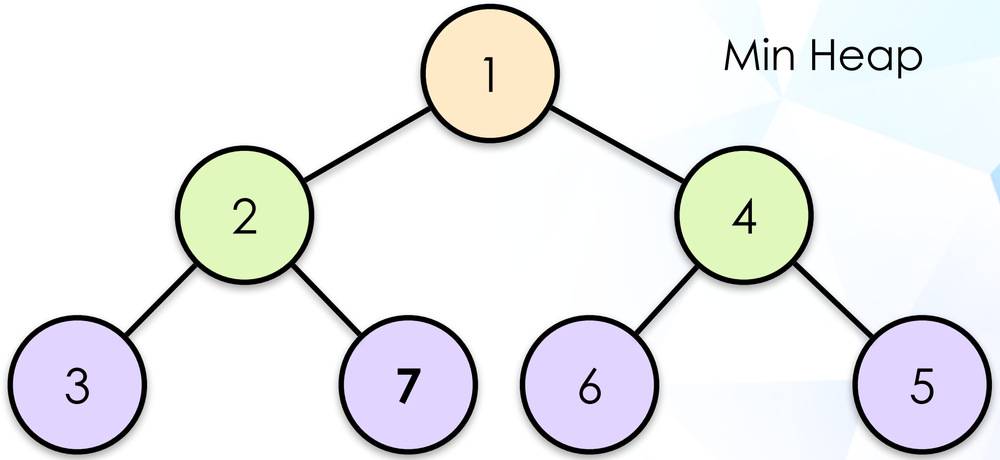
Array

1	2	4	3	7	6	5

Example

How do we get the parent node of Array[4]?

- Current array index = 4
- Applying the formula,
  - Parent node index = (4 - 1) / 2 = 1
  - Parent node value = Array[1] = 2

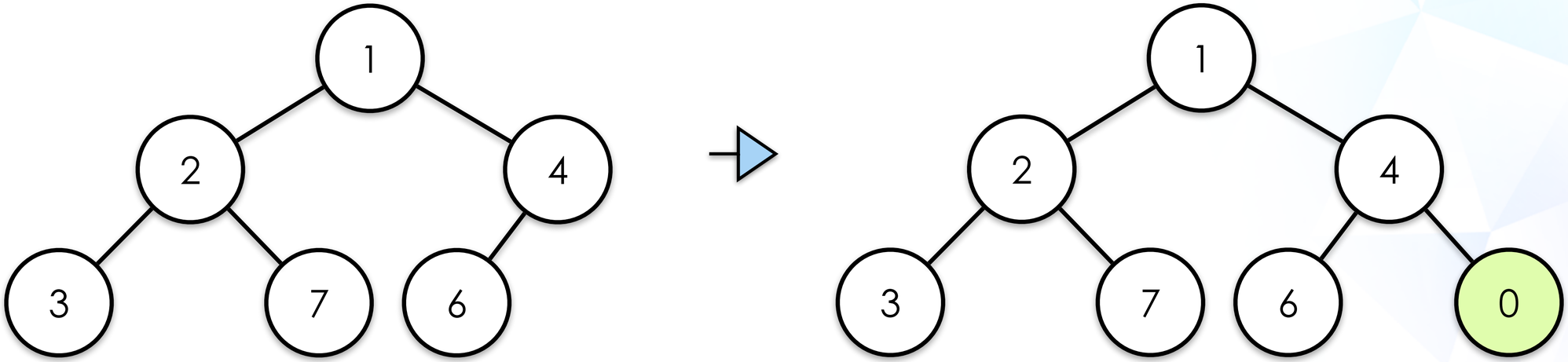


Array

1	2	4	3	**7**	6	5

Insert

To insert a new element with a key of 0, first place it at the end of Heap

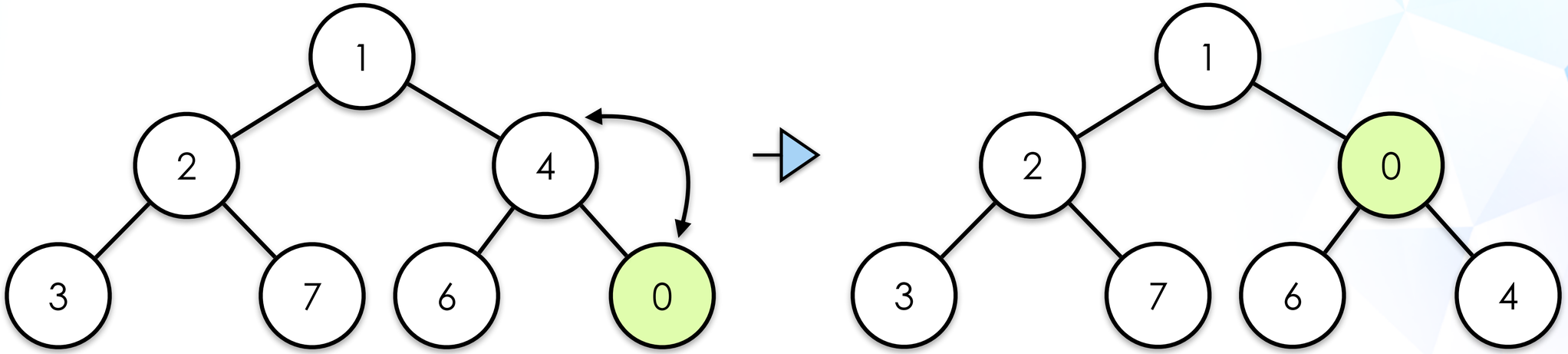


Min Heap

Min Heap

Insert

Compare new element (0) with parent element (4); swap as new element (0) is smaller

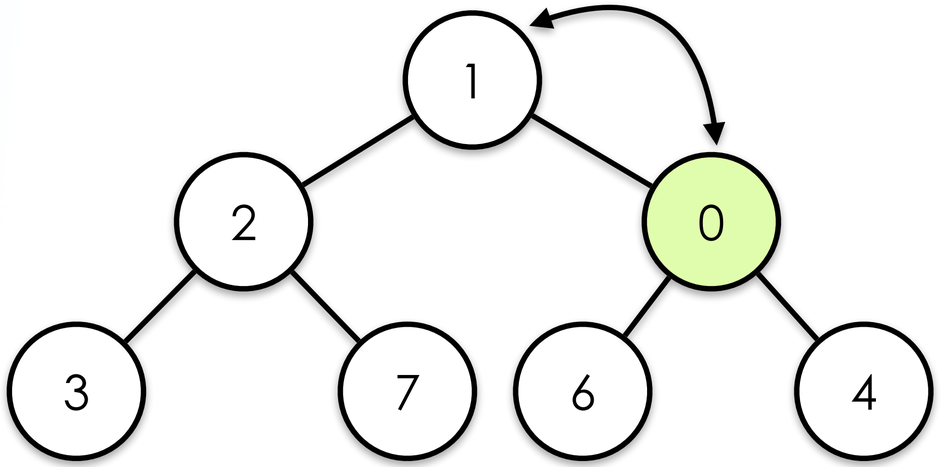


Min Heap

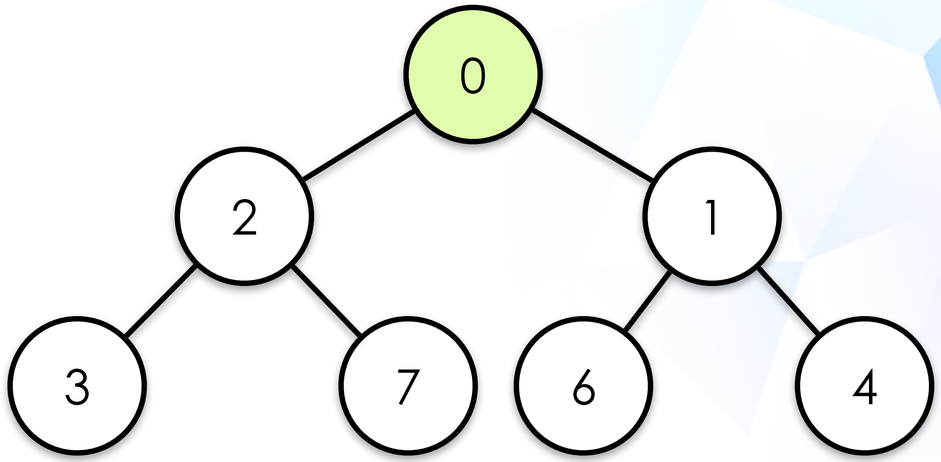
Min Heap

Insert

Swap new element (0) with parent element (1) as new element is smaller; stop as **heap property** is restored



Min Heap



Min Heap

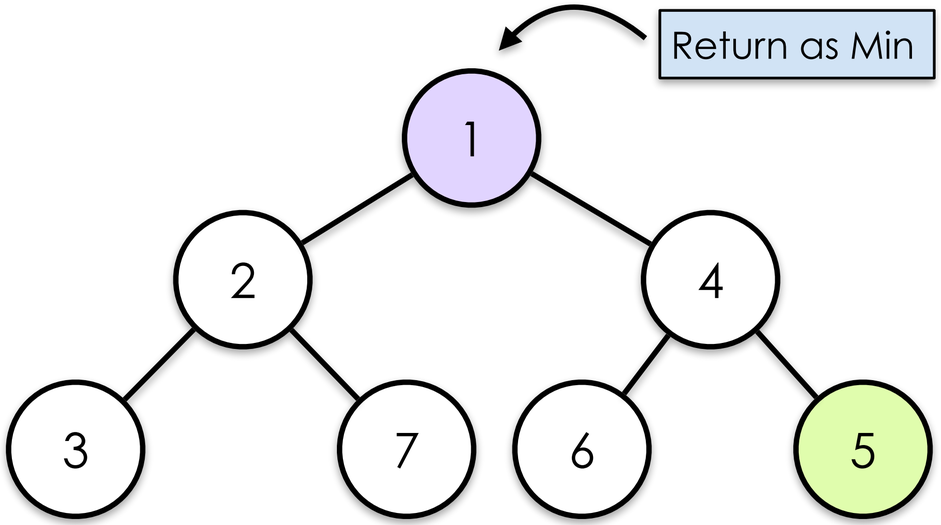
Code Sample

To insert a new node to a Min Heap

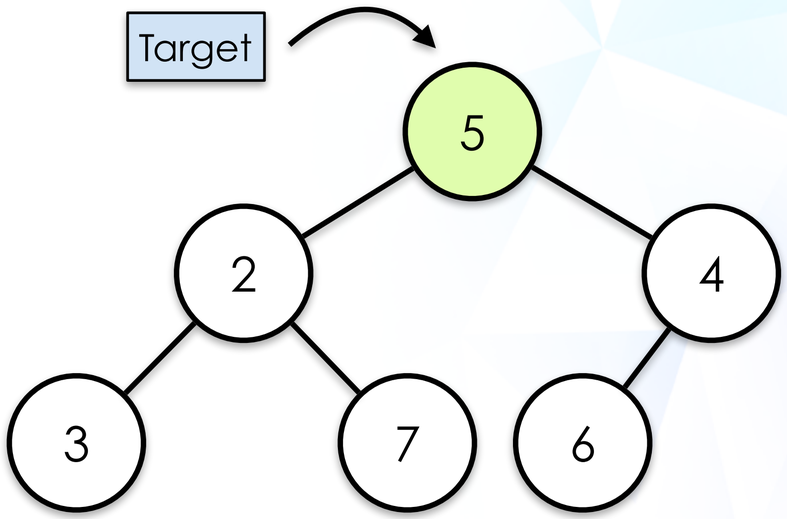
```
void Insert(List<int> heap, int n)
{
    // always add to end of the heap first
    heap.Add(n);

    // heap property is already preserved
    // as there is only 1 item
    if (heap.Count == 1) {
        return;
    }

    // our newly inserted node is at
    // last position
    int child = heap.Count - 1;

    // compare child's value with parent's value
    while (true) {
        int parent = (child - 1) / 2;
        if (parent == child) {
            // we have moved all the way to the top
            break;
        }

        if (heap[parent] > heap[child]) {
            // swap parent and child
            int tmp = heap[parent];
            heap[parent] = heap[child];
            heap[child] = tmp;

            // bubble-up (taking parent's index)
            child = parent;
        }
        else {
           break;
        }
    }
}
```

Remove

To remove the min element from a Min Heap, replace root element (1) with last element (5)



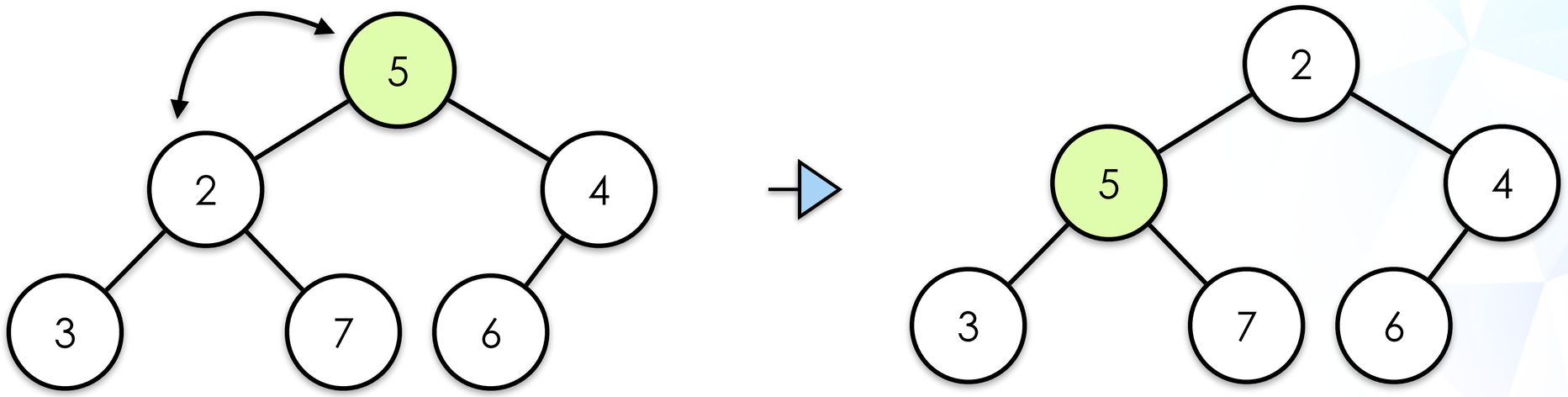
Min Heap



Min Heap

Remove

Child element (2) is the smaller of the two children, swap with target element (5)

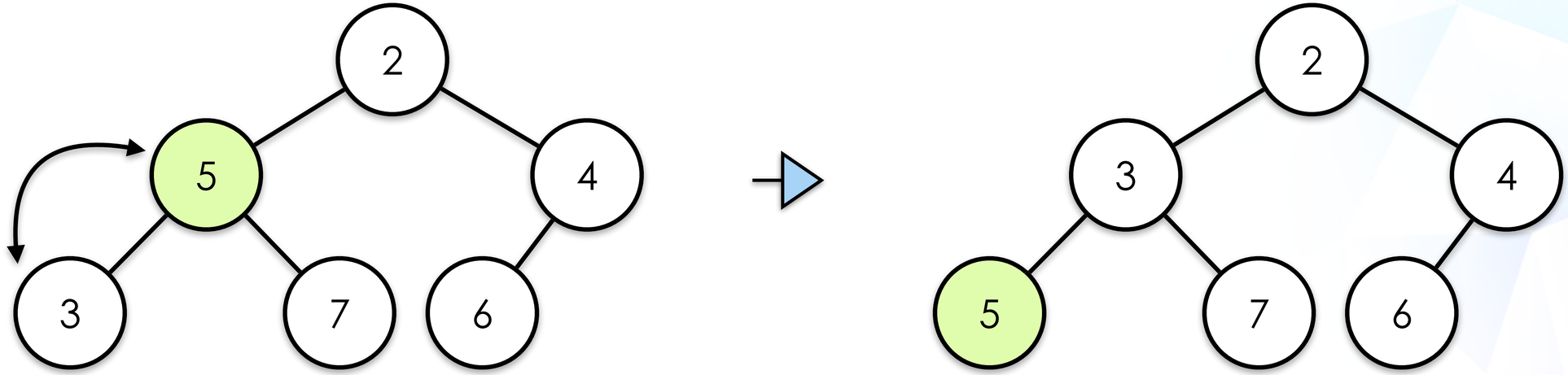


Min Heap

Min Heap

Remove

Child element 3 is the smaller of the two children, swap with target element 5; stop as heap property is restored



Min Heap

Min Heap

THE END