# Lab 8
# Binary Heap

Quang D. C.
dungcamquang@tdtu.edu.vn

November 22, 2021

**Note**

In this lab, we will learn another data structure called Binary Heap and apply it for Heap Sort.

## Part I
# Classwork

*In this part, lecturer will:*

- Summarize the theory related to this lab.
- Instruct the lesson in this lab to the students.
- Explain the sample implementations.

*Responsibility of the students in this part:*

- Students practice sample exercises with solutions.
- During these part, students may ask any question that they don't understand or make mistakes. Lecturers can guide students, or do general guidance for the whole class if the errors are common.

## 1. What is Binary Heap?

A Heap is a special Tree-based data structure in which the tree is a **complete binary tree**. Generally, Heaps can be of two types:

- **Max-Heap:** $A[parent(i)] \geq A[i]$
- **Min-Heap:** $A[parent(i)] \leq A[i]$

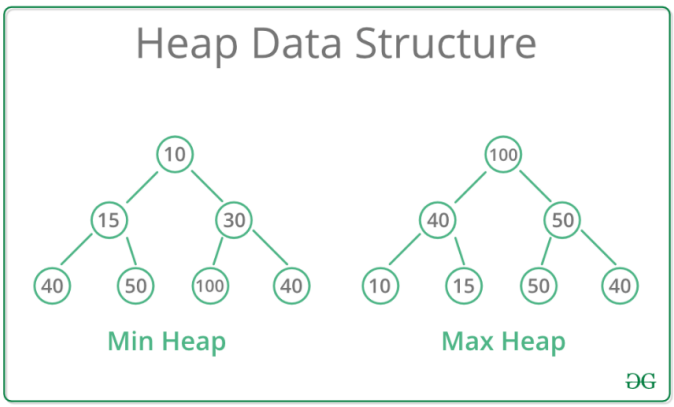
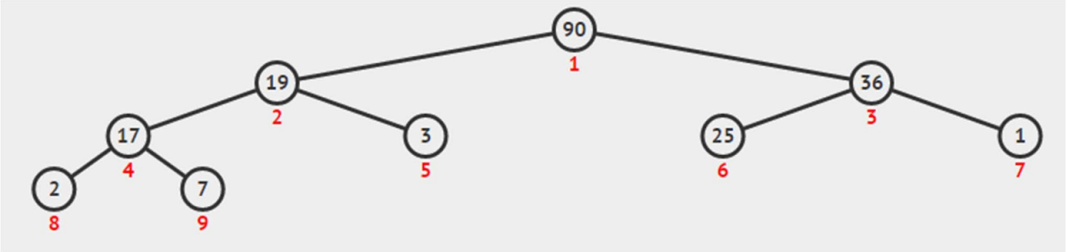



Figure 1: Binary Heap [1]

For storing a **complete binary tree**, we can use an array.




| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| - | 90 | 19 | 36 | 17 | 3 | 25 | 1 | 2 | 7 | - | - |

Figure 2: Using an array to store Binary Heap

Index 0 can be used or not, and the corresponding parent-children relation formulas may be different.

| | **Root at index 0** | **Root at index 1** |
|---|---|---|
| Parent(i) | (i-1)/2 | i/2 |
| Left(i) | (2*i)+1 | 2*i |
| Right(i) | (2*i)+2 | (2*i)+1 |

We will implement Max-heap by using an array that starts at index 1.
Two basic operations of Max-heap are:

- Insert
- Extract Max

[1] https://www.geeksforgeeks.org/binary-heap/

## 2. Construct a Heap class

Here is the code for initializing a heap instance:

```
public class MaxHeap{
    int[] heap;
    int heapSize;
    int maxSize; //maximum size to initialize an heap array

    public MaxHeap(int capity){
        heapSize = 0;
        this.maxSize = capity + 1;
        heap = new int[maxSize];
        heap[0] = -1;
    }
}
```

You need some methods to access the parent and childs index.

```
private int parent(int i){
    return i/2;
}

private int left(int i){
    //thinking and filling
}

private int right(int i){
    //thinking and filling
}
```

And the method to help us swap two values in an array.

```
private void swap(int i, int j){
    int temp = heap[i];
    heap[i] = heap[j];
    heap[j] = temp;
}
```

After finishing the helper methods, we continue to implement the method to insert a value to a heap.

## 3. Insert

We have three steps to do:

1. Increase heap size by 1
2. Add a new key at the heap size position.
3. If the new key is smaller than its parent, then we don't need to do anything. If not, **shift it up**.

This is the code of insertion:

```
public void insert(int key){
    if(heapSize == maxSize){
```

```
        throw new NoSuchElementException("Overflow Exception"); // Remember to import java.util.NoSuchElementException;
    }
    heapSize += 1;
    heap[heapSize] = key;

    shiftUp(heapSize);
}
```

How to "shift up"? This is the answer:

```
private void shiftUp(int i){
    while(i > 1 && heap[parent(i)] < heap[i]){
        swap(parent(i), i); //this method you have defined before
        i = parent(i);
    }
}
```

## 4. Extract Max

Extract max as known as delete the maximum element (the root of Max-Heap). Insertion needs a shift up method. On the contrary, deletion needs a **shift down** method.

```
public int extractMax(){
    if(heapSize == 0){
        throw new NoSuchElementException("Underflow Exception");
    }
    int max = heap[1];
    heap[1] = heap[heapSize];
    heapSize -= 1;
    shiftDown(1);
    return max;
}
```

This is the code for **shifting down**:

```
private void shiftDown(int i){
    while(i <= heapSize){
        int max = heap[i];
        int max_id = i;
        if(left(i) <= heapSize && max < heap[left(i)]){
            max = heap[left(i)];
            max_id = left(i);
        }
        if(right(i) <= heapSize && max < heap[right(i)]){
            max = heap[right(i)];
            max_id = right(i);
        }
        if(max_id != i){
            swap(max_id,i);
            i = max_id;
        }
        else{
            break;
        }
    }
}
```

## 5. Heap Sort

Given the pseudo code:

```
HeapSort(array)
1 BuildHeap(array)
2 for i ←0 →n-1 do
3     A[i] = ExtractMax()
4 return A
```

Implement this method:

```
1 public static void heapSort(int[] arr){
2     //code here
3 }
4 //This function can be implemented in the class that contains the
     main function.
```

# Part II
# Excercise

*Responsibility of the students in this part:*

- Complete all the exercises with the knowledge from **Part I**.
- Ask your lecturer if you have any question.
- Submit your solutions according to your lecturer requirement.

## Exercise 1

Complete the Max-heap class following the instructions in this lab.

## Exercise 2

Implement Min-heap of integers.

## Exercise 3

Sort the following numbers ascending/descending by using Heap Sort:

$$15, 23, 18, 63, 21, 35, 36, 21, 66, 12, 42, 35, 75, 23, 64, 78, 39$$

## Exercise 4

(*) Define the priority queue to queue some people. A person has name and priority number. Given that: higher priority = higher number. Perform these operations:

- Enqueue: (Alex, 3), (Bob, 2), (David, 6), (Susan, 1)
- Dequeue

- Enqueue: (Mike, 5), (Kevin, 4)
- Dequeue
- Dequeue
- Enqueue: (Helen, 0), (Paul, 8), (Iris, 7)
- Dequeue

Show the result of 4 persons will be dequeued.

THE END