BAB V
SORTED TREE: HEAP TREE

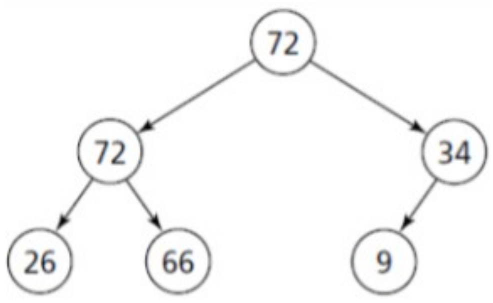
5.1 Tujuan Praktikum

1. Able to understand the concept of Heap Tree
2. Able to implement Heap Tree

5.2 Materi

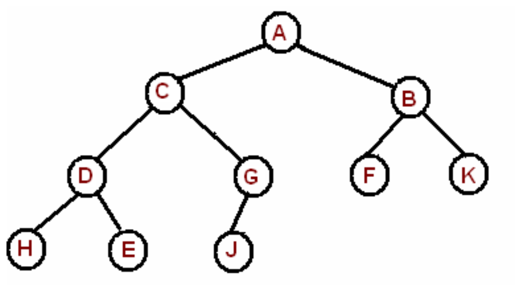
A heap is a binary tree that satisfies the following property:

- Tree must be complete. The point is that each level of the tree should be filled (a parent should have the right leftchild and child), except at the lowest level may not be complete. If the lowest level is not complete then the child should be vacated is rightchild.
- For each element E, all of which are child values of E must be less than or equal with the value of E. Therefore, root will store the maximum value.



5.2.1 Array Implementation

In indexing the array index, heap tree has rules to store value as follows:



0	1	2	3	4	5	6	7	8	9	10
	A	C	B	D	G	F	K	H	E	J

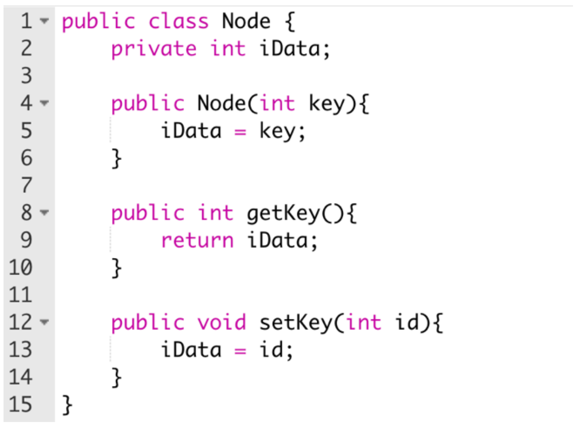
In this case the root is placed on the index "1" to facilitate the calculation of the index. Can beobserved that for a node other than the root applicable,

• For each leftchild is at index 2 * k

• For each rightchild is at index 2 * k + 1

• And the parent is at index k / 2

Heap tree is formed of nodes that have a key value. Therefore, it needs a node class as thefollowing code



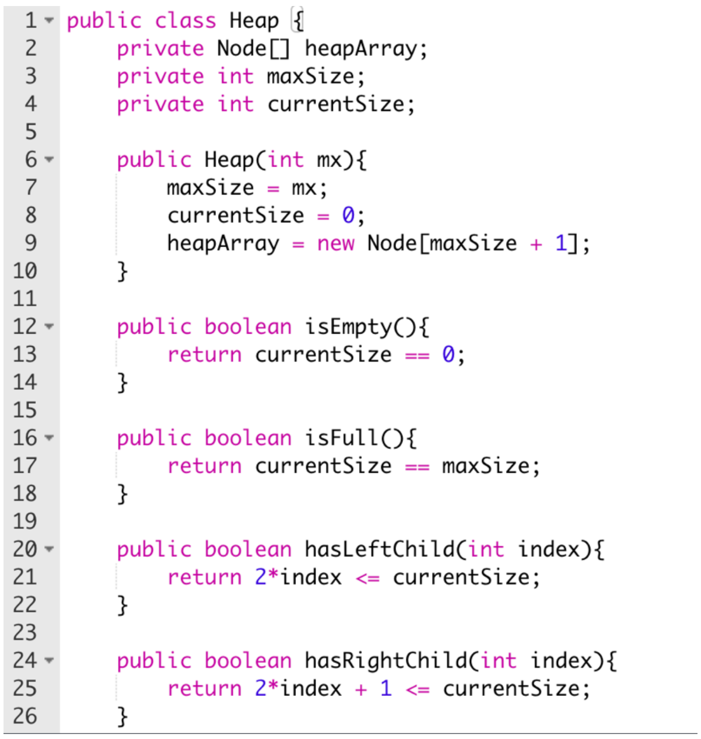
```java
public class Node {
    private int iData;

    public Node(int key){
        iData = key;
    }

    public int getKey(){
        return iData;
    }

    public void setKey(int id){
        iData = id;
    }
}
```

After make the node class, then made a heap class



```java
public class Heap {
    private Node[] heapArray;
    private int maxSize;
    private int currentSize;

    public Heap(int mx){
        maxSize = mx;
        currentSize = 0;
        heapArray = new Node[maxSize + 1];
    }

    public boolean isEmpty(){
        return currentSize == 0;
    }

    public boolean isFull(){
        return currentSize == maxSize;
    }

    public boolean hasLeftChild(int index){
        return 2*index <= currentSize;
    }

    public boolean hasRightChild(int index){
        return 2*index + 1 <= currentSize;
    }
```

Once formed Heap class, of course we need a method to insert, remove, and to keep the heap property is always met.

5.2.2 Insert

The new elements always be added to the heap at the bottom level. Property of the heap is always kept by comparing elements that are added to its parent and then swap any position when the element is larger than its parent. Keep in mind that the purpose of this property is to keep the value of the parent is always greater than the child.

```
    public boolean insert(int key){
        if (isFull()){
            return false;
        }

        Node newNode = new Node(key);
        currentSize++;
        heapArray[currentSize] = newNode;
        trickleUp(currentSize);
        return true;
    }

    public void trickleUp(int index){
        int parent = index/2;
        Node bottom = heapArray[index];

        while (index>1 && heapArray[parent].getKey() < bottom.getKey()){
            heapArray[index] = heapArray[parent];
            index = parent;
            parent = index/2;
        }

        heapArray[index] = bottom;
    }
```

5.2.3 Remove

In heap tree, when its applied to the method remove () then the node that has the largest value is deleted. Then the root will be replaced with the lower-right node. Then the tree heap will keep the property with the existing rule that would bring down the rightmost node to the most suitable position on the heap tree.

```java
    public Node remove(){
        Node root = heapArray[1];
        heapArray[1] = heapArray[currentSize];
        currentSize--;
        trickleDown(1);
        return root;
    }

    public void trickleDown(int index){
        int largerChild;
        Node top = heapArray[index];

        while (hasLeftChild(index)){ // has at least left child
            int leftChild = 2*index;
            int rightChild = leftChild + 1;

            if (hasRightChild(index) &&
                heapArray[rightChild].getKey() > heapArray[leftChild].getKey()){
                largerChild = rightChild;
            }
            else{
                largerChild = leftChild;
            }

            if (top.getKey() >= heapArray[largerChild].getKey()){ // stop looping
                break;
            }

            heapArray[index] = heapArray[largerChild];
            index = largerChild;
        }
        heapArray[index] = top;
    }
```

Delete on this method is delete the max, so if the method is called the key to the root will be lost because the key to the root key must be the greatest / maximum. After the root is removed then the trickle-down method will look for the right key to root replacement at the same time to maintain the heap property remains unfulfilled.

5.2.4 Print

Each value / key from the heap tree stored in an array. Then to display the heap tree, the heap property is used.

```java
    public void displayHeap(){
        System.out.print("heapArray : ");

        for (int i=1 ; i<=currentSize ; i++){
            System.out.print(heapArray[i].getKey() + " ");
        }

        System.out.println("");
    }
}
```

5.3 Assignment

1. Create heap tree for array with size of 15 and data key:

   Key = {78, 3, 9, 10, 23, 77, 34, 86, 90, 100, 20, 66, 94, 63, 97}

2. Create a heap sort function to use heap from this chapter.