LAB_08

ICS 202

Jalal Ali Zainaddin | 202154790 | 11/03/2023

Task 1.1: buildHeapTopDown(), percolateUp(int index):

From build top to down, we start from the index of the root to the parameter of percolate up until we reach the last leaf node.

In percolate, we check if index is still not at root and compare if the parent is less than child in order to swap them, then go to next child.

Code:

```java
private void buildHeapTopDown()
{
    // Task02
    // we take the root node to start from up to down
    for (int i = 1; i <=size ; i++) {
        percolateUp(i);
    }
}
```

```java
private void percolateUp(int index){
    // Task01

    // check if index is still not at root and compare if the parent is less than child
    if (index > 1 && array[index].compareTo(array[index/2] )< 0)
    {
        // swap between them
        T temp = array[index];
        array[index] = array[index/2];
        array[index/2] = temp;

        // go to the next child
        index = index/2;
        percolateUp(index);
    }
}
```

Output:

```
The original array is: [10, 2, 8, 9, 1, 6, 3, 4, 0, 5]
The min-heap is: [0, 1, 3, 2, 5, 8, 6, 10, 4, 9]
The sorted array is: [0, 1, 2, 3, 4, 5, 6, 8, 9, 10]
```

Task 1.2: buildHeapBottomUp(), percolateDown(int index):

From build down to top, we take the last non-terminal node to start from percolate down, and decrement each time in for loop until we reach to the index of root "1".

In percolate down, we starting to assign the index of a next child node in a variable, then we create a temp variable to store the current child, then we use for loop primarily for two things, to increment the next child index by two times, or by one if the right of the next child exists and is lesser than its sibling. The second one is to compare between right child and its parent, if it's less, swap them.

Code:

```
private void buildHeapBottomUp()
{
    // Task01
    // we take the last non-terminal node to start from down to up
    for (int i = size/2; i >=1; i--) {
        percolateDown(i);
    }
}
```

```
private void percolateDown(int index)
{
    // Task01

    // assign the index of a next child node
    int nextChild;
    // store the key of given child
    T temp = array[index];
    for (; index*2 <= size; index = nextChild){
        // assign it to the index of next child
        nextChild = index*2;
        // we check if the right of the next child exists to compare between the siblings
        if (nextChild != size && array[nextChild + 1].compareTo(array[nextChild])<0)
            //go to right sibling
            nextChild++;
        // compare between right child and its parent, if it's less, swap them
        if (array[nextChild].compareTo(temp) <0)
            array[index] = array[nextChild];
        else
            break;

    }
    // end of swapping
    array[index] = temp;
}
```

Output:

```
The original array is: [10, 2, 8, 9, 1, 6, 3, 4, 0, 5]
The min-heap is: [0, 1, 3, 2, 5, 6, 8, 4, 9, 10]
The sorted array is: [0, 1, 2, 3, 4, 5, 6, 8, 9, 10]
```

Task 2: Design a Hospital class with patients by creating Patient class:

In patient class, each patient has name, emergency level, and arrival order, and by using comparator, we can determine what we should serve first based on emergency level, then arrival order. In hospital class, we created an array of ten patients with different emergency level and arrival order, then we sorted them using BinaryHeap heap sort.

Code:

```java
public class Patient implements Comparable<Patient>{
    // the following attributes
    2 usages
    private String name;
    6 usages
    private int emergencyLevel;
    6 usages
    private int arrivalOrder;

    // a constructor to assign them
    10 usages
    Patient(String n,int el,int ord){
        this.name = n;
        this.emergencyLevel = el;
        this.arrivalOrder = ord;
    }
    // compare to compare between patients emergency and arrival levels
    @Override
    public int compareTo(Patient o) {
        if (this.emergencyLevel > o.emergencyLevel)
            return 1;
        else if (this.emergencyLevel < o.emergencyLevel)
            return -1;
        else{
            if (this.arrivalOrder > o.arrivalOrder)
                return 1;
            else if (this.arrivalOrder < o.arrivalOrder)
                return -1;
            return 0;
        }
    }
    @Override
    public String toString() {
        return "Name: "+ name +", Emergency Level: "+ emergencyLevel + ", ArrivalOrder: " + arrivalOrder;
    }
}
```

```java
public class Hospital {
    no usages
    public static void main(String[] args) {
        Patient[] patients = new Patient[10];
        patients[0] = new Patient( n: "Ali",(int)(Math.random()*5+1),(int)(Math.random()*10+1));
        patients[1] = new Patient( n: "Saleem",(int)(Math.random()*5+1),(int)(Math.random()*10+1));
        patients[2] = new Patient( n: "Jamaal",(int)(Math.random()*5+1),(int)(Math.random()*10+1));
        patients[3] = new Patient( n: "Thamer",(int)(Math.random()*5+1),(int)(Math.random()*10+1));
        patients[4] = new Patient( n: "Muhsin",(int)(Math.random()*5+1),(int)(Math.random()*10+1));
        patients[5] = new Patient( n: "Said",(int)(Math.random()*5+1),(int)(Math.random()*10+1));
        patients[6] = new Patient( n: "Qasim",(int)(Math.random()*5+1),(int)(Math.random()*10+1));
        patients[7] = new Patient( n: "Maryam",(int)(Math.random()*5+1),(int)(Math.random()*10+1));
        patients[8] = new Patient( n: "Ramadhan",(int)(Math.random()*5+1),(int)(Math.random()*10+1));
        patients[9] = new Patient( n: "Zainab",(int)(Math.random()*5+1),(int)(Math.random()*10+1));

        System.out.println("The original order of patients arrival is:");
        for (Patient p: patients)
            System.out.println(p);

        BinaryHeap<Patient> patientsOrder = new BinaryHeap<>();
        for (int i = 0; i < patients.length; i++) {
            patientsOrder.enqueue(patients[i]);
        }

        // to give spacing
        System.out.println();

        System.out.println("The TREATMENT order of patients arrival is:");
        Object[] patientSort = patientsOrder.heapSort();
        for (Object p: patientSort)
            System.out.println(p);
    }
}
```

Output:

```
The original order of patients arrival is:
Name: Ali, Emergency Level: 3, ArrivalOrder: 10
Name: Saleem, Emergency Level: 1, ArrivalOrder: 7
Name: Jamaal, Emergency Level: 2, ArrivalOrder: 2
Name: Thamer, Emergency Level: 2, ArrivalOrder: 6
Name: Muhsin, Emergency Level: 3, ArrivalOrder: 8
Name: Said, Emergency Level: 3, ArrivalOrder: 6
Name: Qasim, Emergency Level: 4, ArrivalOrder: 1
Name: Maryam, Emergency Level: 3, ArrivalOrder: 1
Name: Ramadhan, Emergency Level: 2, ArrivalOrder: 5
Name: Zainab, Emergency Level: 1, ArrivalOrder: 5

The TREATMENT order of patients arrival is:
Name: Zainab, Emergency Level: 1, ArrivalOrder: 5
Name: Saleem, Emergency Level: 1, ArrivalOrder: 7
Name: Jamaal, Emergency Level: 2, ArrivalOrder: 2
Name: Ramadhan, Emergency Level: 2, ArrivalOrder: 5
Name: Thamer, Emergency Level: 2, ArrivalOrder: 6
Name: Maryam, Emergency Level: 3, ArrivalOrder: 1
Name: Said, Emergency Level: 3, ArrivalOrder: 6
Name: Muhsin, Emergency Level: 3, ArrivalOrder: 8
Name: Ali, Emergency Level: 3, ArrivalOrder: 10
Name: Qasim, Emergency Level: 4, ArrivalOrder: 1
```

```
The original order of patients arrival is:
Name: Ali, Emergency Level: 5, ArrivalOrder: 8
Name: Saleem, Emergency Level: 4, ArrivalOrder: 1
Name: Jamaal, Emergency Level: 3, ArrivalOrder: 4
Name: Thamer, Emergency Level: 1, ArrivalOrder: 2
Name: Muhsin, Emergency Level: 3, ArrivalOrder: 2
Name: Said, Emergency Level: 1, ArrivalOrder: 2
Name: Qasim, Emergency Level: 3, ArrivalOrder: 9
Name: Maryam, Emergency Level: 1, ArrivalOrder: 6
Name: Ramadhan, Emergency Level: 2, ArrivalOrder: 9
Name: Zainab, Emergency Level: 3, ArrivalOrder: 9

The TREATMENT order of patients arrival is:
Name: Thamer, Emergency Level: 1, ArrivalOrder: 2
Name: Said, Emergency Level: 1, ArrivalOrder: 2
Name: Maryam, Emergency Level: 1, ArrivalOrder: 6
Name: Ramadhan, Emergency Level: 2, ArrivalOrder: 9
Name: Muhsin, Emergency Level: 3, ArrivalOrder: 2
Name: Jamaal, Emergency Level: 3, ArrivalOrder: 4
Name: Qasim, Emergency Level: 3, ArrivalOrder: 9
Name: Zainab, Emergency Level: 3, ArrivalOrder: 9
Name: Saleem, Emergency Level: 4, ArrivalOrder: 1
Name: Ali, Emergency Level: 5, ArrivalOrder: 8
```