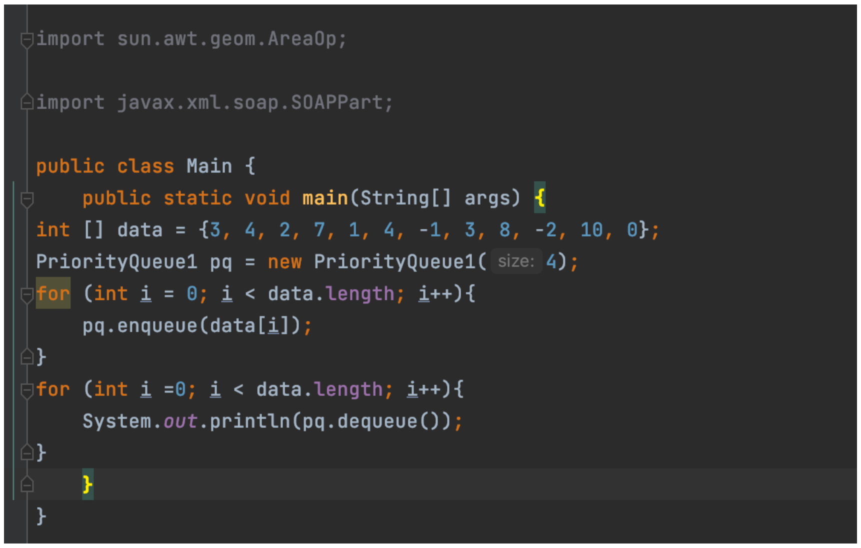
My Anh Huynh
Assignment 3

CS4306 Algorithm Analysis
Fall 2021
Department of Computer Science
Kennesaw State University

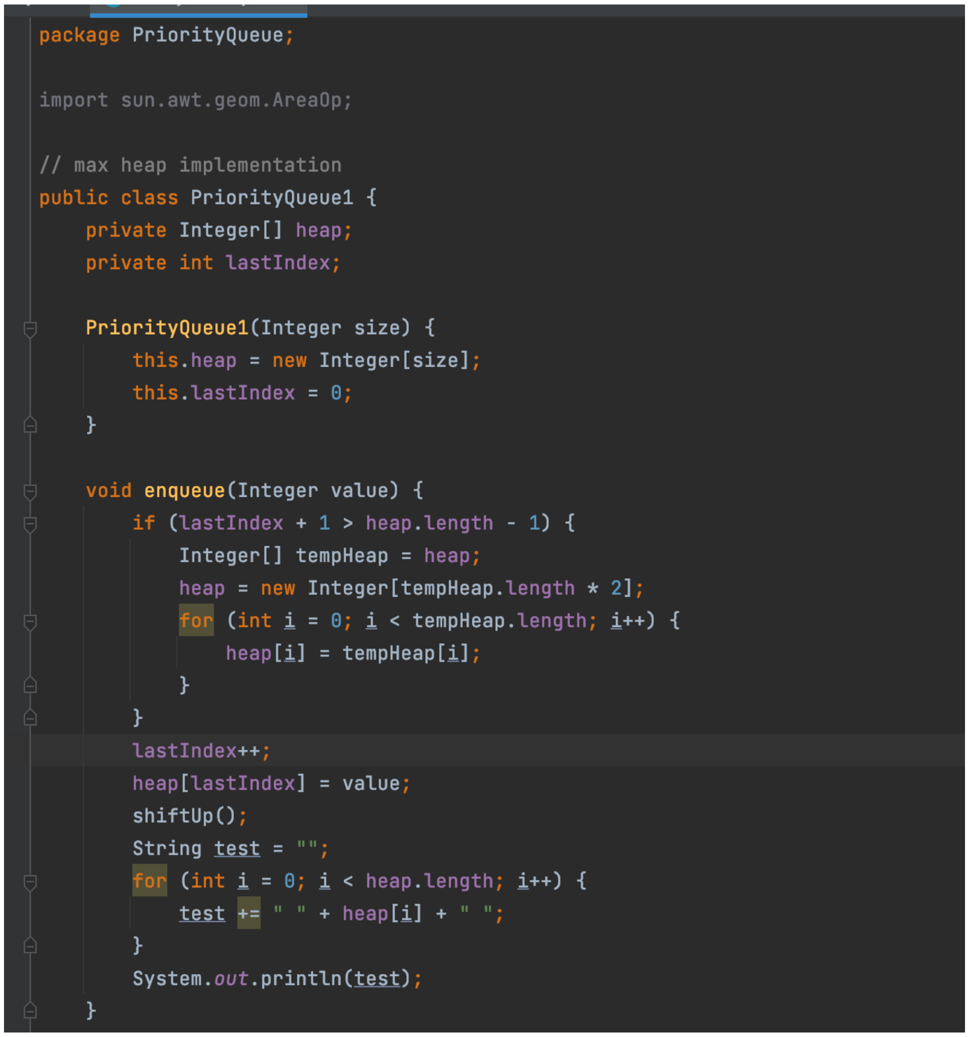
Programming Assignment 3
Due Date: Thursday, September 30, 2021 (by 11:59pm)

Report
My Anh Huynh

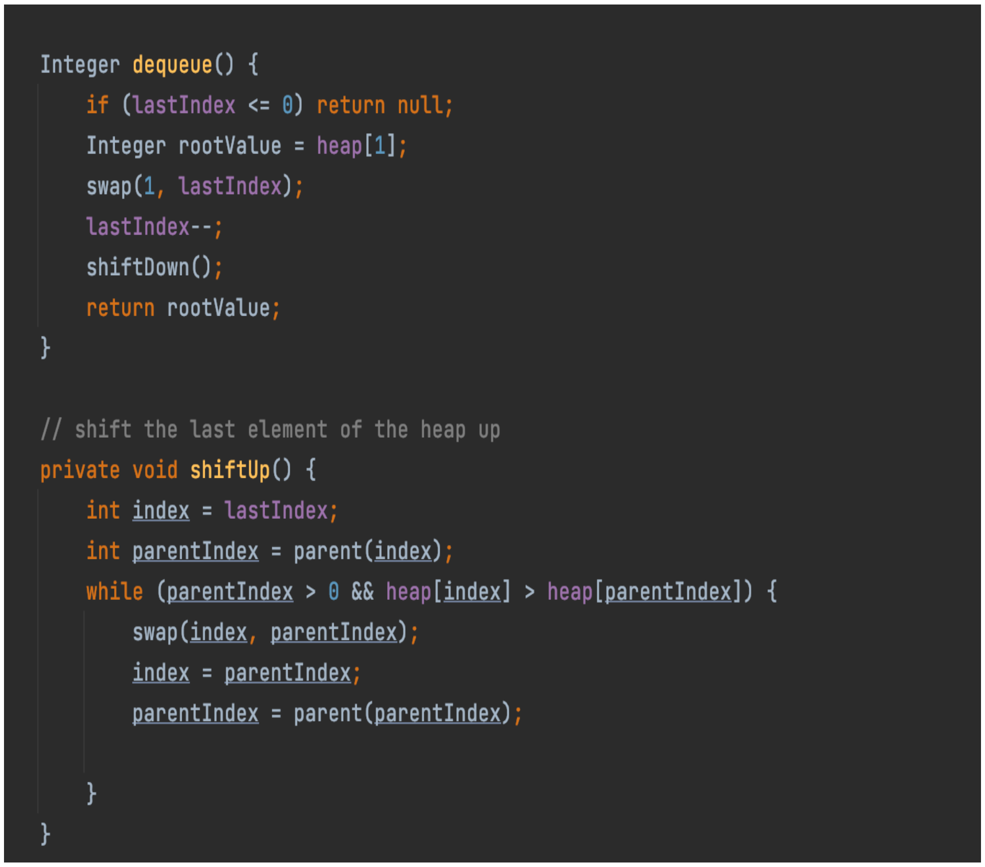
Maintain a Priority Queue Using Heap Data Structure (*Consider the Descending Order Of Elements*)
In this assignment we will work with some concepts such as priority queue and Heap.
A priority queue is a nomal queue where its elements are assigned a specific priority. It could be a larger number higher priority for max heap and smaller number higher priority for min heap.
The reason we could use heap data structure to create a priority queue because heap is a tree- based data structure. Also, we could implement max heap and min heap or min heap where the root of the tree is highest or lowest respectively. Therefore, in priority queue, we needs the highest / lowest priority elements and a heap has the highest/ lowest element which is stored at the root of the tree. Hence, priority queue is implemented using the heap data structure.
We know to create the heap the time complexity is O(logn) but minimum time taken for creating a heap is O(n) using Heapify.
Also, when condering the descending order of Elenments we use Priority Queue with the key : larger number higher priority. So we implefy a max heap.
We could use binary heap based priority queue to do sorting by enqueue all the elements that you want to sort and then dequeue one by one until there is no more element in the priority queue and since the priority queue is implement using binary heap so the time complexity of sorting method will be O(N*logN).
The first n comes from the fact that we have to enqueue and dequeue as many times as the number of elements that we have and the log n comes from the binary heaf shift up and ship down operation.
The Code has a Main class and a PriorityQueue1 class that have methods such as enqueue, dequeue, shiftUp, shiftDown, parent, left, right and swap. Those methods are used to create a max heap from an array and implement priority queue.



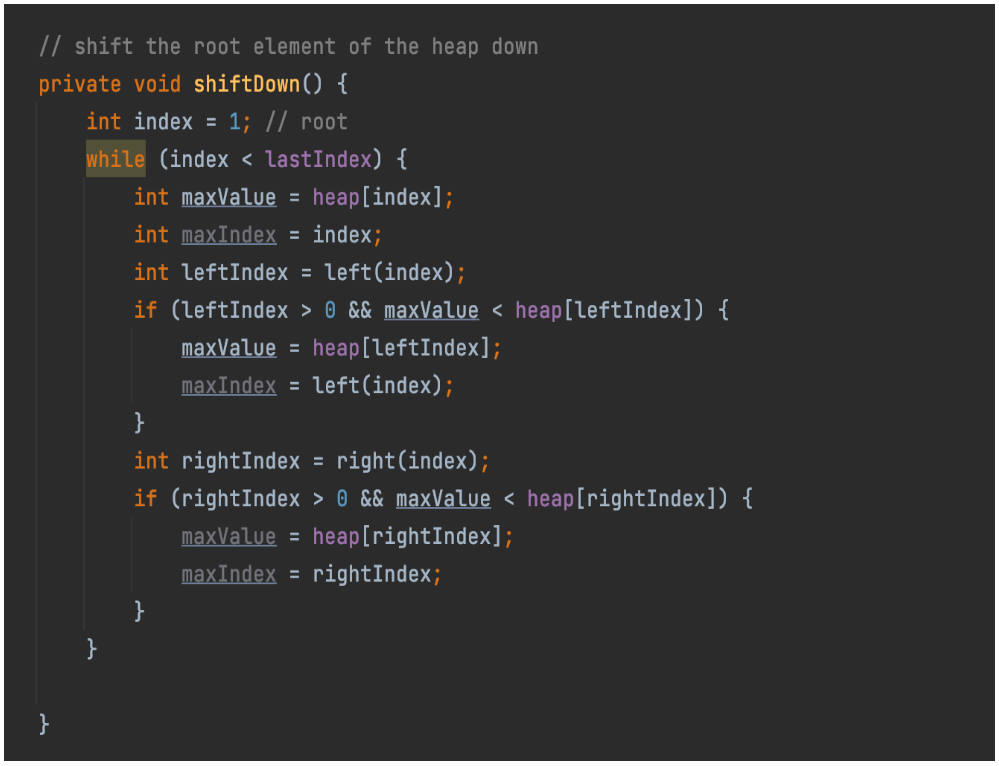
```
import sun.awt.geom.AreaOp;

import javax.xml.soap.SOAPPart;

public class Main {
    public static void main(String[] args) {
int [] data = {3, 4, 2, 7, 1, 4, -1, 3, 8, -2, 10, 0};
PriorityQueue1 pq = new PriorityQueue1( size: 4);
for (int i = 0; i < data.length; i++){
    pq.enqueue(data[i]);
}
for (int i =0; i < data.length; i++){
    System.out.println(pq.dequeue());
}
    }
}
```



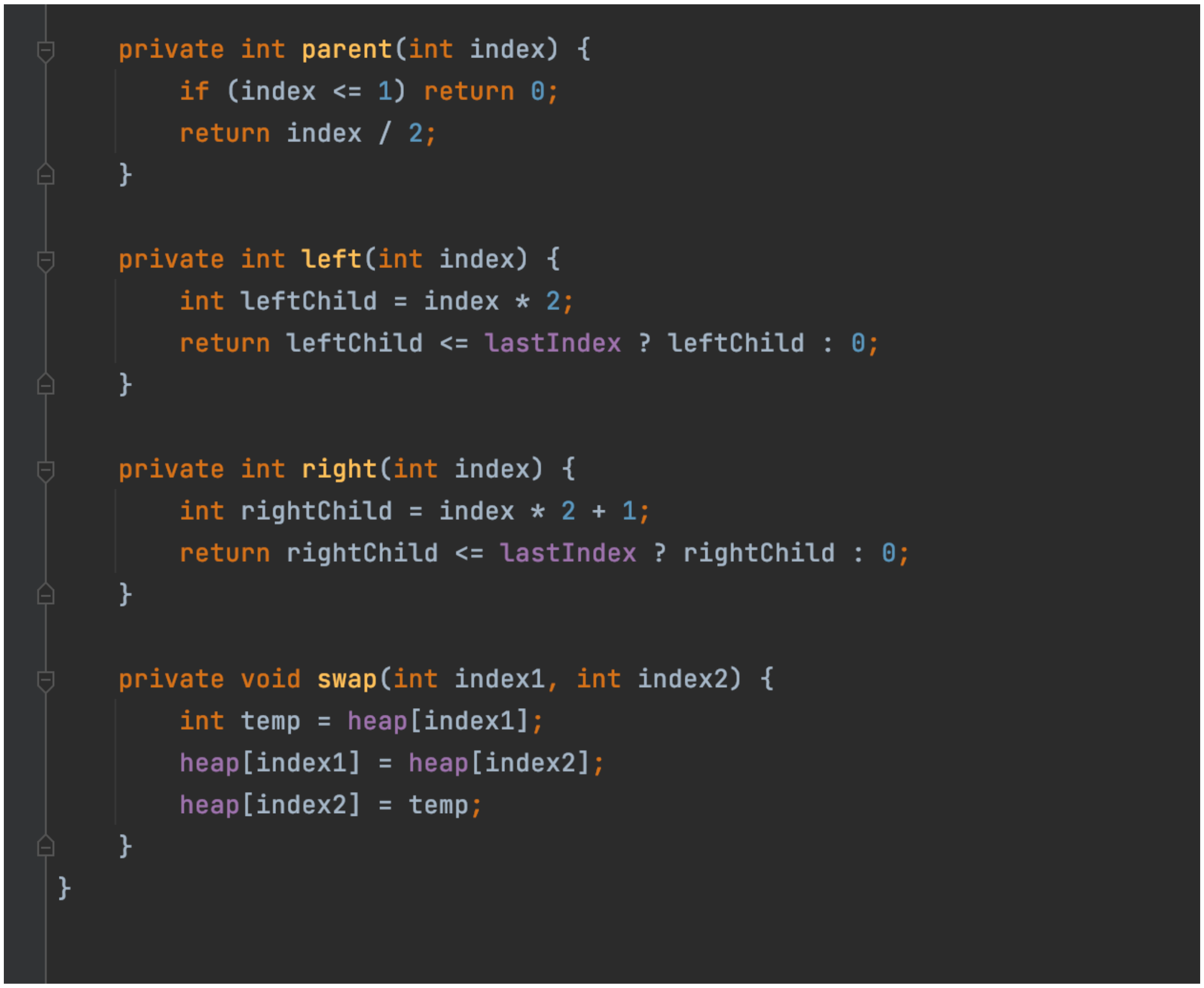
```
package PriorityQueue;

import sun.awt.geom.AreaOp;

// max heap implementation
public class PriorityQueue1 {
    private Integer[] heap;
    private int lastIndex;

    PriorityQueue1(Integer size) {
        this.heap = new Integer[size];
        this.lastIndex = 0;
    }

    void enqueue(Integer value) {
        if (lastIndex + 1 > heap.length - 1) {
            Integer[] tempHeap = heap;
            heap = new Integer[tempHeap.length * 2];
            for (int i = 0; i < tempHeap.length; i++) {
                heap[i] = tempHeap[i];
            }
        }
        lastIndex++;
        heap[lastIndex] = value;
        shiftUp();
        String test = "";
        for (int i = 0; i < heap.length; i++) {
            test += " " + heap[i] + " ";
        }
        System.out.println(test);
    }
```



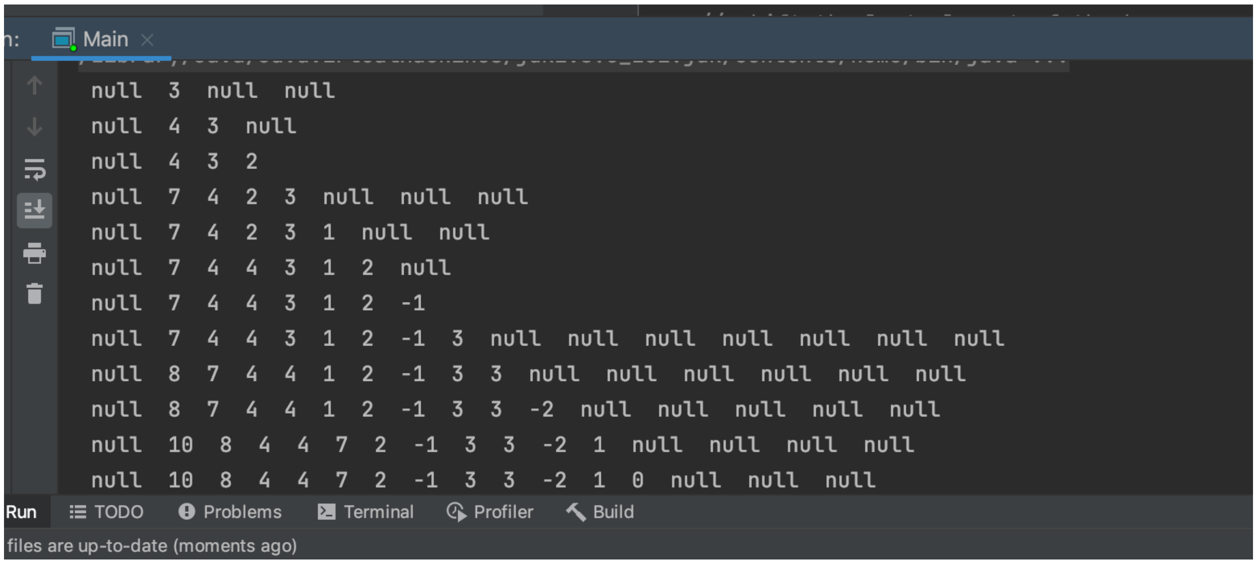
```java
Integer dequeue() {
    if (lastIndex <= 0) return null;
    Integer rootValue = heap[1];
    swap(1, lastIndex);
    lastIndex--;
    shiftDown();
    return rootValue;
}

// shift the last element of the heap up
private void shiftUp() {
    int index = lastIndex;
    int parentIndex = parent(index);
    while (parentIndex > 0 && heap[index] > heap[parentIndex]) {
        swap(index, parentIndex);
        index = parentIndex;
        parentIndex = parent(parentIndex);

    }
}
```



```java
// shift the root element of the heap down
private void shiftDown() {
    int index = 1; // root
    while (index < lastIndex) {
        int maxValue = heap[index];
        int maxIndex = index;
        int leftIndex = left(index);
        if (leftIndex > 0 && maxValue < heap[leftIndex]) {
            maxValue = heap[leftIndex];
            maxIndex = left(index);
        }
        int rightIndex = right(index);
        if (rightIndex > 0 && maxValue < heap[rightIndex]) {
            maxValue = heap[rightIndex];
            maxIndex = rightIndex;
        }
    }

}
```



```java
    private int parent(int index) {
        if (index <= 1) return 0;
        return index / 2;
    }

    private int left(int index) {
        int leftChild = index * 2;
        return leftChild <= lastIndex ? leftChild : 0;
    }

    private int right(int index) {
        int rightChild = index * 2 + 1;
        return rightChild <= lastIndex ? rightChild : 0;
    }

    private void swap(int index1, int index2) {
        int temp = heap[index1];
        heap[index1] = heap[index2];
        heap[index2] = temp;
    }
}
```



```
null  3  null  null
null  4  3  null
null  4  3  2
null  7  4  2  3  null  null  null
null  7  4  2  3  1  null  null
null  7  4  4  3  1  2  null
null  7  4  4  3  1  2  -1
null  7  4  4  3  1  2  -1  3  null  null  null  null  null  null  null
null  8  7  4  4  1  2  -1  3  3  null  null  null  null  null  null
null  8  7  4  4  1  2  -1  3  3  -2  null  null  null  null  null
null  10  8  4  4  7  2  -1  3  3  -2  1  null  null  null  null
null  10  8  4  4  7  2  -1  3  3  -2  1  0  null  null  null
```