SET A

```
import java.util.Arrays;
public class MinHeap {
    private Integer[] taskIds;
    private Integer[] priorities;
    private int capacity;
    private int size;

    public MinHeap(int capacity) {
        this.taskIds = new Integer[capacity];
        this.priorities = new Integer[capacity];
        this.capacity = capacity;
        this.size = 0;
    }

    public void insert(int taskID, int priority) {
        if (size == capacity) {
            throw new RuntimeException("Heap is full");
        }
        taskIds[size] = taskID;
        priorities[size] = priority;
        swim(size);
        size++;
    }

    private void swim(int index) {
        while (index > 0) {
            int parent = (index - 1) / 2;
            if (priorities[index] < priorities[parent]) {
                int tempId = taskIds[index];
                taskIds[index] = taskIds[parent];
                taskIds[parent] = tempId;

                int tempPriority = priorities[index];
                priorities[index] = priorities[parent];
                priorities[parent] = tempPriority;

                index = parent;
            } else {
                break;
```

```
        }
    }
  }
public static int[] processTasks(int[] tasks, int[] priorities, int lowPriority, int highPriority, int
capacity) {
    MinHeap minHeap = new MinHeap(capacity);
    int[] result = new int[capacity];
    int count = 0;
    for (int i = 0; i < tasks.length; i++) {
      if (lowPriority <= priorities[i] && priorities[i] <= highPriority) {
        minHeap.insert(tasks[i], priorities[i]);
      }
    }
    while (minHeap.size > 0 && count < capacity) {
      result[count] = minHeap.extractMin();
      count++;
    }
    return Arrays.copyOf(result, count);
  }
}
```

Rubric

- *2.5 Marks:* Construct the MinHeap class
- *2.5 Marks:* **insert()** -> Checks for proper addition of tasks, priority including the condition to handle a full heap.
- *1 Marks:* **swim()** -> Correctly calculated parent index
- *2 Marks:* **swim()** -> Check that the min-heap order is kept during insertions by swapping indexes correctly for both tasks and priority.
- *2 Marks:* **process_tasks()** -> MinHeap is correctly initialized with the given capacity.
- *3 Marks:* **process_tasks()** -> Check task within the range is inserted.
- 2 Mark: **process_tasks()** -> Extract the minimum value from the heap and insert it in the result array.
-

SET B

```
import java.util.Arrays;
public class MaxHeap {
    private Integer[] taskIds;
    private Integer[] priorities;
    private int capacity;
    private int size;

    public MaxHeap(int capacity) {
        this.taskIds = new Integer[capacity];
        this.priorities = new Integer[capacity];
        this.capacity = capacity;
        this.size = 0;
    }

    public void insert(int taskID, int priority) {
        if (size == capacity) {
            throw new RuntimeException("Heap is full");
        }
        taskIds[size] = taskID;
        priorities[size] = priority;
        swim(size);
        size++;
    }

    private void swim(int index) {
        while (index > 0) {
            int parent = (index - 1) / 2;
            if (priorities[index] > priorities[parent]) {
                int tempId = taskIds[index];
                taskIds[index] = taskIds[parent];
                taskIds[parent] = tempId;

                int tempPriority = priorities[index];
                priorities[index] = priorities[parent];
                priorities[parent] = tempPriority;

                index = parent;
            } else {
                break;
            }
```

```
        }
    }
public static int[] processTasks(int[] tasks, int[] priorities, int lowPriority, int highPriority, int
capacity) {
        MaxHeap maxHeap = new MaxHeap(capacity);
        int[] result = new int[capacity];
        int count = 0;
        for (int i = 0; i < tasks.length; i++) {
            if (lowPriority <= priorities[i] && priorities[i] <= highPriority) {
                maxHeap.insert(tasks[i], priorities[i]);
            }
        }
        while (maxHeap.size > 0 && count < capacity) {
            result[count] = maxHeap.extractMax();
            count++;
        }
        return Arrays.copyOf(result, count);
    }
}
```

Rubric

- *2.5 Marks:* Construct the MaxHeap class
- *2.5 Marks:* **insert()** -> Checks for proper addition of tasks, priority including the condition to handle a full heap.
- *1 Marks:* **swim()** -> Correctly calculated parent index
- *2 Marks:* **swim()** -> Check that the max-heap order is kept during insertions by swapping indexes correctly for both tasks and priority.
- *2 Marks:* **process_tasks()** -> MaxHeap is correctly initialized with the given capacity.
- *3 Marks:* **process_tasks()** -> Check task within the range is inserted.
- 2 Mark: **process_tasks()** -> Extract the maximum value from the heap and insert it in the result array