SET A

```python
import numpy as np
class MinHeap:
    def __init__(self, capacity):
        self.task_ids = [None] * capacity
        self.priorities = [None] * capacity
        self.capacity = capacity
        self.size = 0

    def insert(self, task_id, priority):
        if self.size == self.capacity:
            raise Exception("Heap is full")
        self.task_ids[self.size] = task_id
        self.priorities[self.size] = priority
        self.swim(self.size)
        self.size += 1

    def swim(self, index):
        while index > 0:
            parent = (index - 1) // 2
            if self.priorities[index] < self.priorities[parent]:
                self.task_ids[index], self.task_ids[parent] = self.task_ids[parent], self.task_ids[index]
                self.priorities[index], self.priorities[parent] = self.priorities[parent], self.priorities[index]
                index = parent
            else:
                break
```

```python
def process_tasks(tasks, priorities, low_priority, high_priority, capacity):
    min_heap = MinHeap(capacity)
    result = np.empty(capacity, dtype=int)
    count = 0
    for i in range(len(tasks)):
        if low_priority <= priorities[i] <= high_priority:
            min_heap.insert(tasks[i], priorities[i])
    while min_heap.size > 0 and count < capacity:
        result[count] = min_heap.extract_min()
        count += 1
    return 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

```python
import numpy as np

class MaxHeap:
    def __init__(self, capacity):
        self.task_ids = [None] * capacity
        self.priorities = [None] * capacity
        self.capacity = capacity
        self.size = 0

    def insert(self, task_id, priority):
        if self.size == self.capacity:
            raise Exception("Heap is full")
        self.task_ids[self.size] = task_id
        self.priorities[self.size] = priority
        self.swim(self.size)
        self.size += 1

    def swim(self, index):
        while index > 0:
            parent = (index - 1) // 2
            if self.priorities[index] > self.priorities[parent]:
                self.task_ids[index], self.task_ids[parent] = self.task_ids[parent], self.task_ids[index]
                self.priorities[index], self.priorities[parent] = self.priorities[parent], self.priorities[index]
                index = parent
            else:
                break
```

```python
def process_tasks(tasks, priorities, low_priority, high_priority, capacity):
    max_heap = MaxHeap(capacity)
    result = np.empty(capacity, dtype=int)
    count = 0
    for i in range(len(tasks)):
        if low_priority <= priorities[i] <= high_priority:
            max_heap.insert(tasks[i], priorities[i])
    while max_heap.size > 0 and count < capacity:
        result[count] = max_heap.extract_max()
        count += 1
    return 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.