PMarking Rubric:

Set A -

Criteria	Points to meet	Marks
Priority Score Calculation [5]	- Correct use of **priority_score = demand × rating** to calculate scores for each book.	**2**
	- **Handling all input elements:** Ensure all elements in arr1 and arr2 are considered, and the result matches expected output.	**3**
Heap Construction [7]	- Correct creation of an empty **MaxHeap** before insertion begins.	**1**
	- Proper **insertion** logic is implemented to maintain max-heap properties during insertions.	**3**
	- Ensure the **swim()** method correctly maintains heap structure during insertion.	**3**
Top 3 Extraction [3]	- Correct use of **extract** method to retrieve top 3 priority scores.	**2**
	- Ensure the extracted elements are in descending order based on priority scores.	**1**
Total:		**15**

Set B -

Criteria	Points to meet	Marks
Priority Score Calculation [5]	- Correct use of **overall_rating = difficulty × satisfaction** to calculate scores for each course.	**2**
	- **Handling all input elements:** Ensure all elements in arr1 and arr2 are considered, and the result matches expected output.	**3**
Heap	- Correct creation of an empty **MinHeap** before insertion begins.	**1**

Construction [3]	- Proper use of pre-implemented **`insert`** and **`swim`** methods	**2**
Top 3 Extraction [7]	- Correct implementation of **`extract()`** and **`sink()`** method to retrieve the top 3 courses.	**3 + 3**
	- Ensure the extracted elements are in ascending order based on ratings.	**1**
Total:		**15**

Tentative Solution:

Set A (Python):

```
def parentIndex(index):
    if index == 0:
        return 0
    return (index-1) // 2

def leftIndex(index):
    return (index*2)+1

def rightIndex(index):
    return (index*2)+2

class MaxHeap:
    def __init__(self, capacity):
        self.heapArray = [0] * capacity
        self.capacity = capacity
        self.size = 0

    def insert(self, item):
        self.heapArray[self.size] = item
        self.swim(self.size)
        self.size += 1

    def swim(self, index):
        item, parent_index = self.heapArray[index], parentIndex(index)
        if item > self.heapArray[parent_index]:
            self.heapArray[parent_index], self.heapArray[index] =
self.heapArray[index], self.heapArray[parent_index]
```

```
            self.swim(parent_index)

    def extract(self):
        if self.size == 0:
            return None
        item = self.heapArray[0]
        self.heapArray[0] = self.heapArray[self.size-1]
        self.size -= 1
        self.sink(0)
        self.heapArray[self.size] = None
        return item

    def sink(self, index):
        max_index = index
        item, left_index, right_index = self.heapArray[index],
leftIndex(index), rightIndex(index)

        if left_index < self.size and self.heapArray[left_index] >
self.heapArray[max_index]:
            max_index = left_index

        if right_index < self.size and self.heapArray[right_index] >
self.heapArray[max_index]:
            max_index = right_index

        if self.heapArray[index] < self.heapArray[max_index] and max_index !=
index:
            self.heapArray[index], self.heapArray[max_index] =
self.heapArray[max_index], self.heapArray[index]
            self.sink(max_index)
#-------------------------------------------------------------------------------#
arr1 = [50, 80, 40, 60, 52]
arr2 = [4.5, 4.8, 4.0, 4.2, 4.7]
priority_scores = [0]*len(arr1)
#? Priority Score Calculation:
for i in range(len(priority_scores)):
    priority_scores[i] = int((arr1[i] * arr2[i]).__ceil__())
print("Priority Scores:", priority_scores)
#---------------------------------------#
#? Heap Construction:
heap = MaxHeap(len(priority_scores))
for score in priority_scores:
    heap.insert(score)
print(heap.heapArray)
#---------------------------------------#
```

```
#? Top 3 Extraction:
top3 = [0]*3
for idx in range(len(top3)):
    top3[idx] = heap.extract()
print(f"Top Courses: {', '.join(map(str, top3))}")
#----------------------------------------#
```

Java:

```
class MaxHeap {
    protected int[] heap;
    protected int capacity;
    protected int size;

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

    private static int parentIndex(int index) {
        if (index == 0) {
            return 0;
        }
        return (index - 1) / 2;
    }

    private static int leftIndex(int index) {
        return (index * 2) + 1;
    }

    private static int rightIndex(int index) {
        return (index * 2) + 2;
    }

    public void insert(int item) {
        heap[size] = item;
        swim(size);
        size++;
    }

    private void swim(int index) {
        int item = heap[index];
        int parentIndex = parentIndex(index);
        if (item > heap[parentIndex]) {
```

```
            int temp = heap[parentIndex];
            heap[parentIndex] = heap[index];
            heap[index] = temp;
            swim(parentIndex);
        }
    }

    public Integer extractMax() {
        if (size == 0) {
            return null;
        }
        int item = heap[0];
        heap[0] = heap[size - 1];
        size--;
        sink(0);
        heap[size] = 0;
        return item;
    }

    private void sink(int index) {
        int maxIndex = index;
        int leftIndex = leftIndex(index);
        int rightIndex = rightIndex(index);

        if (leftIndex < size && heap[leftIndex] > heap[maxIndex]) {
            maxIndex = leftIndex;
        }

        if (rightIndex < size && heap[rightIndex] > heap[maxIndex]) {
            maxIndex = rightIndex;
        }

        if (heap[index] < heap[maxIndex] && maxIndex != index) {
            int temp = heap[index];
            heap[index] = heap[maxIndex];
            heap[maxIndex] = temp;
            sink(maxIndex);
        }
    }

    public void createHeapFromArray(int[] arr) {
        for (int i = 0; i < arr.length; i++) {
            insert(arr[i]);
        }
    }
```

```
}

public class Main {
    public static void main(String[] args) {
        int[] arr1 = {50, 80, 40, 60, 52};
        double[] arr2 = {4.5, 4.8, 4.0, 4.2, 4.7};
        int[] priority_scores = new int[arr1.length];
        for (int i=0; i<arr1.length; i++) {
            double temp = Math.ceil((double)arr1[i] * arr2[i]);
            priority_scores[i] = (int)temp;
        }
        printArray(priority_scores, priority_scores.length);

        MaxHeap heap = new MaxHeap(priority_scores.length);
        heap.createHeapFromArray(priority_scores);
        printArray(heap.heap, heap.capacity);

        int[] top3 = new int[3];
        for (int i=0; i<3; i++) {
            top3[i] = heap.extractMax();
        }
        printArray(top3, 3);
    }
    public static void printArray(int[] array, int size) {
        System.out.print("[");
        for (int i = 0; i < size; i++) {
            System.out.print(array[i]);
            if (i < size - 1) {
                System.out.print(", ");
            }
        }
        System.out.println("]");
    }
}
```

Set B (Python):

```
def parentIndex(index):
    if index == 0 :
        return 0
    return (index-1) // 2

def leftIndex(index):
```

```
    return (index*2)+1

def rightIndex(index):
    return (index*2)+2

class MinHeap:
    def __init__(self, capacity):
        self.heapArray = [0] * capacity
        self.capacity = capacity
        self.size = 0

    def insert(self, item):
        self.heapArray[self.size] = item
        self.swim(self.size)
        self.size += 1

    def swim(self, index):
        item,parent_index = self.heapArray[index],parentIndex(index)
        if item < self.heapArray[parent_index]:
            self.heapArray[parent_index], self.heapArray[index] =
self.heapArray[index],self.heapArray[parent_index]
            self.swim(parent_index)

    def extract(self):
        if self.size == 0:
            return None
        item = self.heapArray[0]
        self.heapArray[0] = self.heapArray[self.size-1]
        self.size -= 1
        self.sink(0)
        self.heapArray[self.size] = None
        return item

    def sink(self, index):
        min_index = index
        item, left_index, right_index = self.heapArray[index],
leftIndex(index), rightIndex(index)

        if left_index < self.size and self.heapArray[left_index] <
self.heapArray[min_index]:
            min_index = left_index

        if right_index < self.size and self.heapArray[right_index] <
self.heapArray[min_index]:
            min_index = right_index
```

```
        if self.heapArray[index] > self.heapArray[min_index] and min_index !=
index:
            self.heapArray[index], self.heapArray[min_index] =
self.heapArray[min_index],  self.heapArray[index]
            self.sink(min_index)
#-------------------------------------------------------------------------#
arr1 = [7, 6, 5, 9, 8]
arr2 = [6, 8, 7, 5, 4]
overall_ratings = [0]*len(arr1)
#? Priority Score Calculation:
for i in range(len(overall_ratings)):
    overall_ratings[i] = arr1[i] * arr2[i]
print("Overall Ratings:", overall_ratings)
#---------------------------------------#
#? Heap Construction:
heap = MinHeap(len(overall_ratings))
for rating in overall_ratings:
    heap.insert(rating)
print(heap.heapArray)
#---------------------------------------#
#? Top 3 Extraction:
top3 = [0]*3
for idx in range(len(top3)):
    top3[idx] = heap.extract()
print(f"Top Courses: {', '.join(map(str, top3))}")
#---------------------------------------#
```

Java:

```
class MinHeap {
    protected int[] heap;
    protected int capacity;
    protected int size;

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

    private static int parentIndex(int index) {
        if (index == 0) {
            return 0;
        }
```

```
        return (index - 1) / 2;
    }

    private static int leftIndex(int index) {
        return (index * 2) + 1;
    }

    private static int rightIndex(int index) {
        return (index * 2) + 2;
    }

    public void insert(int item) {
        heap[size] = item;
        swim(size);
        size++;
    }

    private void swim(int index) {
        int item = heap[index];
        int parentIndex = parentIndex(index);
        if (item < heap[parentIndex]) {
            int temp = heap[parentIndex];
            heap[parentIndex] = heap[index];
            heap[index] = temp;
            swim(parentIndex);
        }
    }

    public Integer extractMin() {
        if (size == 0) {
            return null;
        }
        int item = heap[0];
        heap[0] = heap[size - 1];
        size--;
        sink(0);
        heap[size] = 0;
        return item;
    }

    private void sink(int index) {
        int minIndex = index;
        int leftIndex = leftIndex(index);
        int rightIndex = rightIndex(index);
```

```
        if (leftIndex < size && heap[leftIndex] < heap[minIndex]) {
            minIndex = leftIndex;
        }

        if (rightIndex < size && heap[rightIndex] < heap[minIndex]) {
            minIndex = rightIndex;
        }

        if (heap[index] > heap[minIndex] && minIndex != index) {
            int temp = heap[index];
            heap[index] = heap[minIndex];
            heap[minIndex] = temp;
            sink(minIndex);
        }
    }

    public void createHeapFromArray(int[] arr) {
        for (int i = 0; i < arr.length; i++) {
            insert(arr[i]);
        }
    }
}

public class Main {
    public static void main(String[] args) {
        int[] arr1 = {7, 6, 5, 9, 8};
        int[] arr2 = {6, 8, 7, 5, 4};
        int[] overall_ratings = new int[arr1.length];
        for (int i=0; i<arr1.length; i++) {
            overall_ratings[i] = arr1[i] * arr2[i];
        }
        printArray(overall_ratings, overall_ratings.length);

        MinHeap heap = new MinHeap(overall_ratings.length);
        heap.createHeapFromArray(overall_ratings);
        printArray(heap.heap, heap.capacity);

        int[] top3 = new int[3];
        for (int i=0; i<3; i++) {
            top3[i] = heap.extractMin();
        }
        printArray(top3, 3);
    }
    public static void printArray(int[] array, int size) {
        System.out.print("[");
```

```
        for (int i = 0; i < size; i++) {
            System.out.print(array[i]);
            if (i < size - 1) {
                System.out.print(", ");
            }
        }
        System.out.println("]");
    }
}
```