Set A

Python

```
def create_heap_from_array(self, arr):
    for i in range(len(arr)):
        self.insert(arr[i])



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

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

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

    if self.heap[index] < self.heap[max_index] and max_index != index:
        self.heap[index], self.heap[max_index] = self.heap[max_index], self.heap[index]
        self.sink(max_index)




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



def SUM_of_k_max_elements(self, k):
    sum=0
    for i in range(k):
        sum+=self.extractMax()

    return sum
########################################
array = [11, 15, 8, 2, 31, 23]
k=3
heap = MaxHeap(6)
heap.create_heap_from_array(array)
print("Sum =",heap.SUM_of_k_max_elements(k))
```

Java

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

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

public MaxHeap Sum_of_k_max_elements(int k) {
        int sum = 0 ;

        for (int i = 1; i <= k; i++) {
            sum*=extractMax();
        }

        return sum;
    }

public static void main(String[] args) {
        int[] array = new int[]{11, 15, 8, 2, 31, 23};
        int k = 3;
        MaxHeap heap = new MaxHeap(6);
        heap.createHeapFromArray(array);
        System.out.println("sum = "+heap.Sum_of_k_max_elements);
    }
```

Set B

Python

```
  def create_heap_from_array(self, arr):
      for i in range(len(arr)):
          self.insert(arr[i])



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

      #! Check Left Child
      if left_index < self.size and self.heap[left_index] < self.heap[min_index]:
          min_index = left_index
      #! Check right child
      if right_index < self.size and self.heap[right_index] < self.heap[min_index]:
          min_index = right_index
      if self.heap[index] > self.heap[min_index] and min_index != index:
          self.heap[index], self.heap[min_index] = self.heap[min_index],  self.heap[index]
          self.sink(min_index)



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

def Product_of_k_min_elements(self, k):
    product=1
    for i in range(k):
        product*=self.extractMin()
    return product
#####################################
array = [11, 15, 8, 2, 31, 23]
k=3
heap = MinHeap(6)
heap.create_heap_from_array(array)
print("Product =",heap.Product_of_k_min_elements(k))
```

Java

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

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

public MinHeap Product_of_k_min_elements(int k) {
        int product = 1 ;
        for (int i = 1; i < =k; i++) {
            product*=extractMin();
        }
        return product;
    }

public static void main(String[] args) {
        int[] array = new int[]{11, 15, 8, 2, 31, 23};
        int k = 3;
        MinHeap heap = new MinHeap(6);
        heap.createHeapFromArray(array);
        System.out.println("Product =
"+heap.Product_of_k_min_elements);
    }
```

RUBRIC

SN	Criteria	Marks
1	Finding out which Heap to Use	1
2	Creating a heap from an array (loop/function)	3
3	Sink Function	3
4	Extract Max/Min Function	3
5	Create a function to find out the sum/product	1
6	Find out the result (sum/product) properly	3
7	Return & print the result(sum/product)	1
	Total:	15

Note*: There are multiple ways to solve this problem, and appropriate marks can be given for each approach based on its correctness and efficiency.