**Set A**

| Python | Java |
| --- | --- |
| 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)) | 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 | Java |
| --- | --- |
| 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)) | 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.