SKILL3

2100032079
C. MANVITH

1.

```
package Skill3;
import java.lang.StringBuilder;
import java.lang.Comparable;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.util.PriorityQueue;

public class P1 {
   public static void main(String args[] ) throws Exception {

      BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
      PriorityQueue<Course> pq = new PriorityQueue<Course>();
      StringBuilder sb = new StringBuilder();
      String[] params = br.readLine().split(" ");
      String[] students = br.readLine().split(" ");
      String[] monks = br.readLine().split(" ");
      int c = Integer.parseInt(params[0]);
      int p = Integer.parseInt(params[1]);
      int n = Integer.parseInt(params[2]);

      for (int i = 0; i < c; i++) {
          int iq = (i >= n)? 0: Integer.parseInt(students[i]);
         Course course = new Course(i+1, iq);
         pq.add(course);
      }

      for (int i = 0; i < p; i++) {
         Course course = pq.poll();
         sb.append(course.idx).append(" ");
         int iq = (i >= monks.length)? 0: Integer.parseInt(monks[i]);
         course.addStudent(iq);
         pq.add(course);
      }

      System.out.println(sb.toString());
   }

   static class Course implements Comparable<Course> {
         final int idx;
```

```
        int students;
        int lastIQ1;
        int lastIQ2;

        public Course(int idx, int iq) {
                this.idx = idx;
                this.lastIQ1 = iq;
                if (iq > 0) {this.students++;}
        }

        public int compareTo(Course c) {
                int x = this.calculateZ() - c.calculateZ();
                if (x == 0) {
                        return this.idx - c.idx;
                }
                return x;
        }

        public int calculateZ() {
                return students * (lastIQ1 + lastIQ2);
        }

        public void addStudent(int iq) {
                this.lastIQ2 = this.lastIQ1;
                this.lastIQ1 = iq;
                this.students++;
        }
   }

}
```

2.

```
package Skill3;


        public class MinHeap {
                int[] harr;
                int heap_size;
                int capacity;

                public MinHeap(int a[], int size)
                {
                        heap_size = size;
                        capacity = size;
                        harr = a;
                        int i = (heap_size - 1) / 2;
```

```
        while (i >= 0) {
                MinHeapify(i);
                i--;
        }
}


void MinHeapify(int i)
{
        int l = left(i);
        int r = right(i);
        int smallest = i;
        if (l < heap_size && harr[l] < harr[i])
                smallest = l;
        if (r < heap_size && harr[r] < harr[smallest])
                smallest = r;
        if (smallest != i) {
                swap(i, smallest);
                MinHeapify(smallest);
        }
}

int parent(int i) { return (i - 1) / 2; }

                int left(int i) { return (2 * i + 1); }

                int right(int i) { return (2 * i + 2); }

                int extractMin()
{
        if (heap_size <= 0)
                return Integer.MAX_VALUE;
        if (heap_size == 1) {
                heap_size--;
                return harr[0];
        }

                                int root = harr[0];
        harr[0] = harr[heap_size - 1];
        heap_size--;
        MinHeapify(0);

        return root;
}

                void insertKey(int k)
{
```

```
        if (heap_size == capacity) {
                System.out.println("Overflow: Could not insertKey");
                return;
        }

                                heap_size++;
        int i = heap_size - 1;
        harr[i] = k;

        while (i != 0 && harr[parent(i)] > harr[i]) {
                swap(i, parent(i));
                i = parent(i);
        }
}

boolean isSizeOne()
{
        return (heap_size == 1);
}

void swap(int x, int y)
{
        int temp = harr[x];
        harr[x] = harr[y];
        harr[y] = temp;
}

static int minCost(int len[], int n)
{
        int cost = 0;
        MinHeap minHeap = new MinHeap(len, n);

                while (!minHeap.isSizeOne()) {
                int min = minHeap.extractMin();
                int sec_min = minHeap.extractMin();

                cost += (min + sec_min);
                minHeap.insertKey(min + sec_min);
        }

        return cost;
}

                public static void main(String args[])
{
        int len[] = { 4, 3, 2, 6 };
        int size = len.length;
```

```
                        System.out.println("Total cost for connecting ropes is " + minCost(len,
size));
                }
        };

3.
package Skill3;

        import java.io.BufferedOutputStream;
        import java.io.DataInputStream; import
        java.io.FileInputStream;
        import java.io.IOException; import
        java.util.Comparator; import
        java.util.PriorityQueue;


        public class TestClass {

         public static void main(String[] args) throws IOException {
         Reader sc = new Reader();
         BufferedOutputStream out = new BufferedOutputStream(System.out);
        StringBuilder sb = new StringBuilder(); int sizeArray = sc.nextInt();
        int query = sc.nextInt(); long sum[] = new long[sizeArray+1];
         PriorityQueue<Integer> minHeap = new PriorityQueue<Integer>(1, new
Comparator<Integer>()
        {
         @Override public int
        compare(Integer o1, Integer o2) {
         return o1 - o2;
         }
         });

         PriorityQueue<Integer> maxHeap = new PriorityQueue<Integer>(1, new
Comparator<Integer>()
        {
         @Override public int
        compare(Integer o1, Integer o2) { return
        o2 - o1;
         }
         });
         long totalSum = 0; for (int i
        = 0; i < sizeArray; i++) { int a
        = sc.nextInt();
         totalSum += a;
        maxHeap.offer(a);
        minHeap.offer(a);
```

```
 }
 sum[0] = totalSum; for (int i = 1; i <= sizeArray; i++) {
sum[i] = getSumAfterKOperation(sum[i - 1], maxHeap, minHeap);
 }

 for (int i = 0; i < query; i++) {
sb.append(sum[sc.nextInt()] + "\n");
 }

 out.write(sb.toString().getBytes());
out.flush();
 }

 public static long getSumAfterKOperation(long prevSum,
 PriorityQueue<Integer> maxheap, PriorityQueue<Integer> minheap) {
int max = maxheap.poll(); int min = minheap.poll();

 maxheap.offer(max - min);
minheap.offer(max - min);

 return prevSum - 2 * min;
 }

 static class Reader { final private int
BUFFER_SIZE = 1 << 16; private
DataInputStream din;
 private byte[] buffer; private int
bufferPointer, bytesRead;

 public Reader() { din = new
DataInputStream(System.in); buffer =
new byte[BUFFER_SIZE]; bufferPointer
= bytesRead = 0;
 }

 public Reader(String file_name) throws IOException { din
= new DataInputStream(new FileInputStream(file_name));
buffer = new byte[BUFFER_SIZE]; bufferPointer = bytesRead =
0;
 }

 public String readLine() throws IOException {
byte[] buf = new byte[64]; // line length
 int cnt = 0, c;
while ((c = read()) != -1) {
 if (c == '\n')
break; buf[cnt++] =
```

```
(byte) c;
 }
 return new String(buf, 0, cnt);
 }

 public int nextInt() throws IOException {
 int ret = 0;
byte c = read();
while (c <= ' ')
c = read();
boolean neg = (c == '-');
 if (neg) c =
read(); do {
ret = ret * 10 + c - '0';
 } while ((c = read()) >= '0' && c <= '9');

 if (neg)
return -ret;
return ret;
 }

 public long nextLong() throws IOException {
 long ret = 0;
byte c = read();
while (c <= ' ')
c = read();
 boolean neg = (c == '-');
if (neg) c = read();
do { ret = ret * 10 + c
- '0';
 }
 while ((c = read()) >= '0' && c <= '9');
 if (neg)
return -ret;
return ret;
 }

 public double nextDouble() throws IOException {
double ret = 0, div = 1;
 byte c = read();
while (c <= ' ')
c = read();
 boolean neg = (c == '-');
if (neg) c = read();

 do { ret = ret *
10 + c - '0';
```

```
 }
 while ((c = read()) >= '0' && c <= '9');

 if (c == '.') {
 while ((c = read()) >= '0' && c <= '9') {
ret += (c - '0') / (div *= 10);
 }
 } if
(neg)
return -ret;
return ret;
 }

 private void fillBuffer() throws IOException { bytesRead =
din.read(buffer, bufferPointer = 0, BUFFER_SIZE); if
(bytesRead == -1) buffer[0] = -1;
 }

 private byte read() throws IOException {
if (bufferPointer == bytesRead)
fillBuffer();
 return buffer[bufferPointer++];
 }

 public void close() throws IOException {
 if (din == null)
return; din.close();
 }
 }
}
```