K= 2

Sample Input 0

```
5 2
-8 2 3 -6 10
```

Sample Output 0

```
✓ ✓ ✓ ✓
-8 0 -6 -6
```

-8 2 3 -6 -10

0 1 2 3 4

K=2

i

K elem.

↳ -ve add qu.

3 4

rest.

prev. ans. ✓

remove i-K+1 ✓

-ve add qu. ✓

$0 < \underline{i-K+1}$

$0 < 2-2+1$

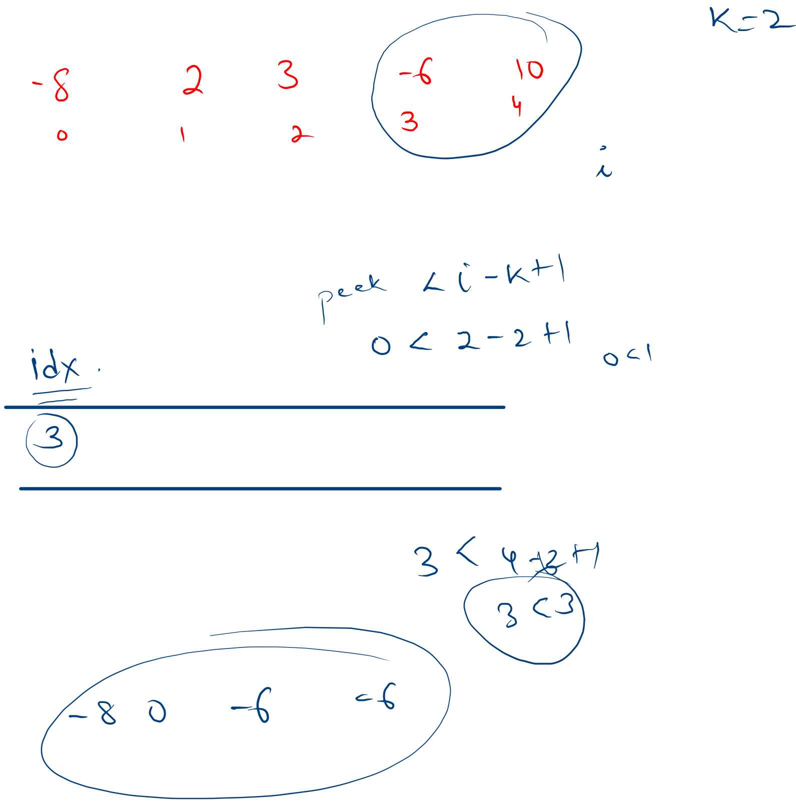
$\boxed{0<1}$

$3 < 4-2+1$

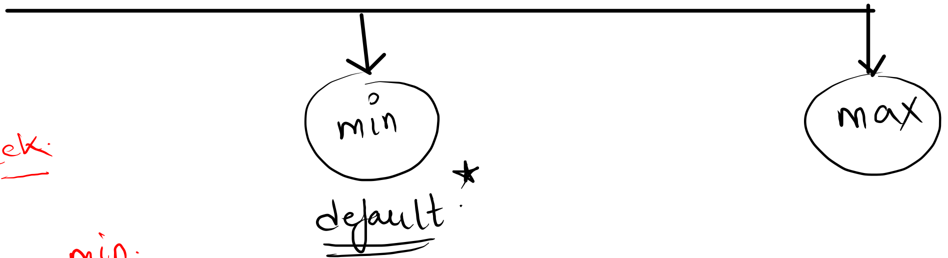
$\boxed{3<3}$ b

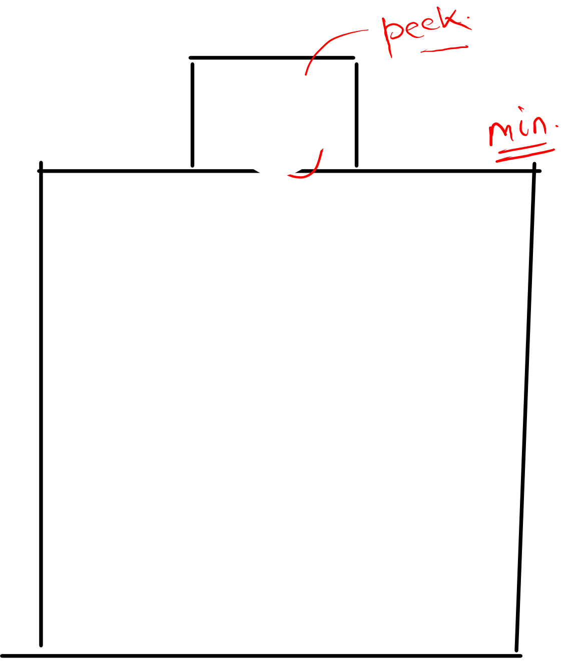
```java
        Queue<Integer> qu = new LinkedList<>();
        //first k ele
        int i = 0;
        while(i < k){
            if(A[i] < 0){
                qu.add(i);
            }
            i++;
        }
        //rest
        while(i < n){
           //1. prev ans
            if(qu.size() == 0){
                System.out.print("0 ");
            }else{
                System.out.print(A[qu.peek()]+" ");
            }
            //2. remove unnecessary
            if(qu.size() != 0 && qu.peek() < i - k + 1){
                qu.remove();
            }
            //3. add -ve
            if(A[i] < 0){
                qu.add(i);
            }
            i++;
        }
        if(qu.size() == 0){
            System.out.print("0 ");
        }else{
            System.out.print(A[qu.peek()]+" ");
        }

    }
}
```



Priority Queue





4 7 5 2 3 9 1 8

remove → 1

remove → 2

remove → 3

remove → 4

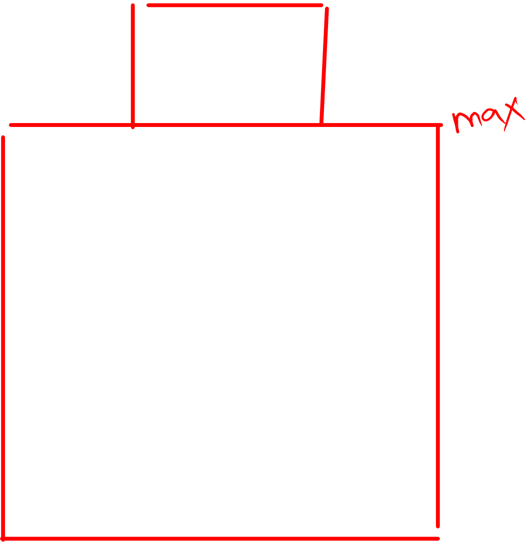
remove → 5

remove → 7

remove → 8

remove → 9

4 7 5 2 3 9 1 8



$\gamma \rightarrow 9$

$\gamma \rightarrow 8$

$\gamma \rightarrow 7$

$\gamma \rightarrow 5$

$\gamma \rightarrow 4$

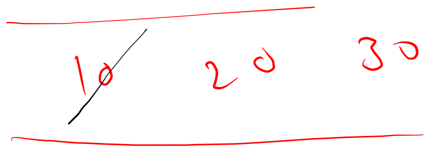
$\gamma \rightarrow 3$

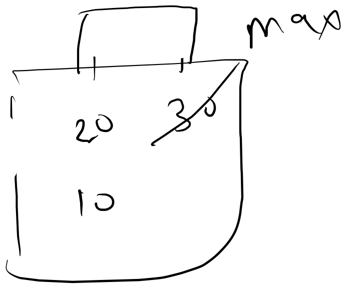
$\gamma \rightarrow 2$

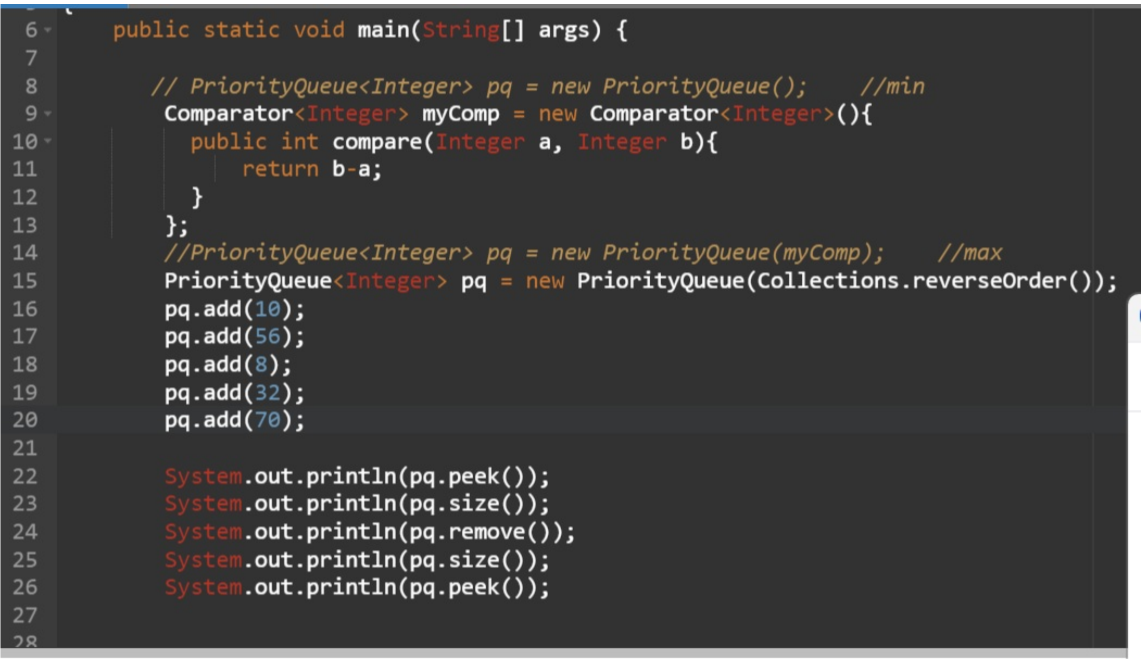
$\gamma \rightarrow 1$

PQ.

- init
- add
- get → peek()
- remove
- size







```
    public static void main(String[] args) {

      // PriorityQueue<Integer> pq = new PriorityQueue();    //min
        Comparator<Integer> myComp = new Comparator<Integer>(){
          public int compare(Integer a, Integer b){
              return b-a;
          }
        };
        //PriorityQueue<Integer> pq = new PriorityQueue(myComp);    //max
        PriorityQueue<Integer> pq = new PriorityQueue(Collections.reverseOrder());
        pq.add(10);
        pq.add(56);
        pq.add(8);
        pq.add(32);
        pq.add(70);

        System.out.println(pq.peek());
        System.out.println(pq.size());
        System.out.println(pq.remove());
        System.out.println(pq.size());
        System.out.println(pq.peek());
```

Priority Queue Basics.

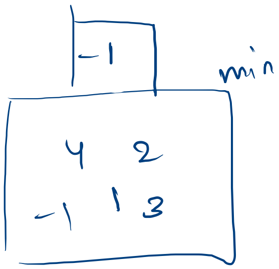
Take T as an integer input. Then take **t** integer elements as input. Each time you take an input. Print the **smallest element** so far, each time a new element is taken as an input.

Sample Input 0

```
5
4
2
1
3
-1
```

Sample Output 0

```
4
2
1
1
-1
```



```java
import java.io.*;
import java.util.*;

public class Solution {

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        PriorityQueue<Integer> pq = new PriorityQueue<>();
        int t = scn.nextInt();
        while(t-- > 0){
            pq.add(scn.nextInt());
            System.out.println(pq.peek());
        }
    }
}
```

Maximum Product of Two Elements in an Array

Given the array of integers nums, you will choose two different indices i and j of that array. Return the maximum value of (nums[i]-1)*(nums[j]-1).

Sample Input 0

```
4
3
4
5
2
```

Sample Output 0

```
12
```

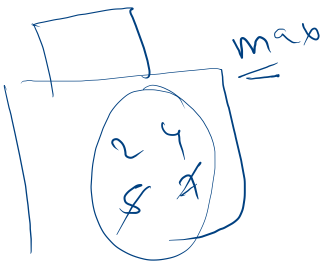
$i \rightarrow \underline{(nums[i] - 1) * (nums[j] - 1)}$

j ✓

$\boxed{35}$

$v1 = 7$

$v2 = 5$



```java
import java.io.*;
import java.util.*;

public class Solution {

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        PriorityQueue<Integer> pq = new PriorityQueue(Collections.reverseOrder());
        int t = scn.nextInt();
        while(t-- > 0){
            pq.add(scn.nextInt());
        }
        int v1 = pq.remove();
        int v2 = pq.remove();
        System.out.println((v1-1) * (v2-1));
    }
}
```

always choose min.

Minimum Cost of ropes 3

There are given N ropes of different lengths, we need to connect these ropes into one rope. The cost to connect two ropes is equal to sum of their lengths. The task is to connect the ropes with minimum cost. Given N size array arr[] contains the lengths of the ropes.

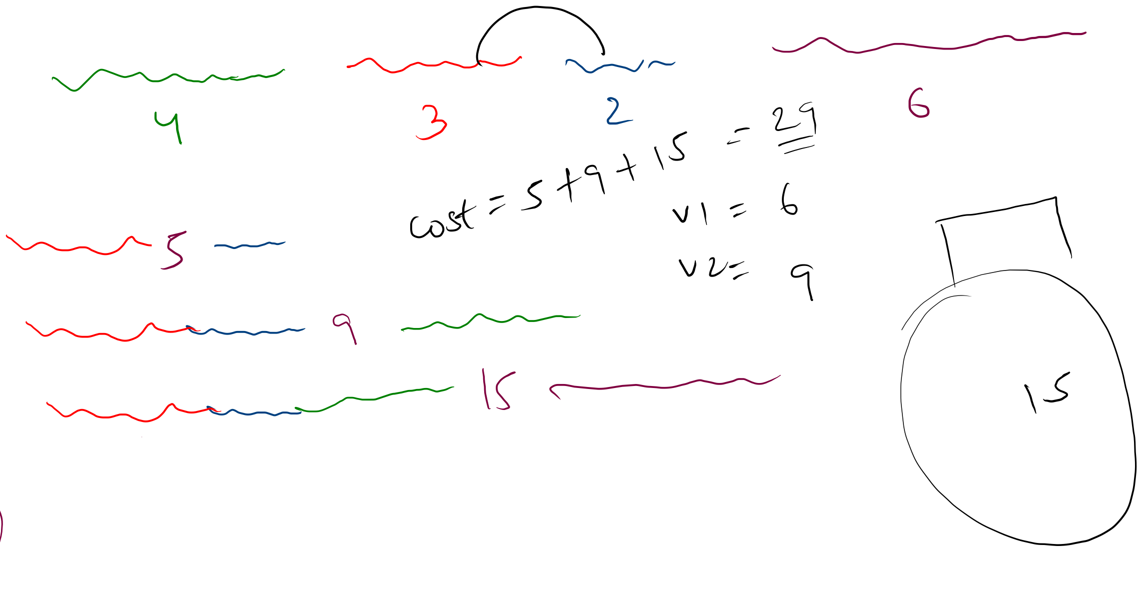
Sample Input 0

```
4
4 3 2 6
```

Sample Output 0

```
29
```

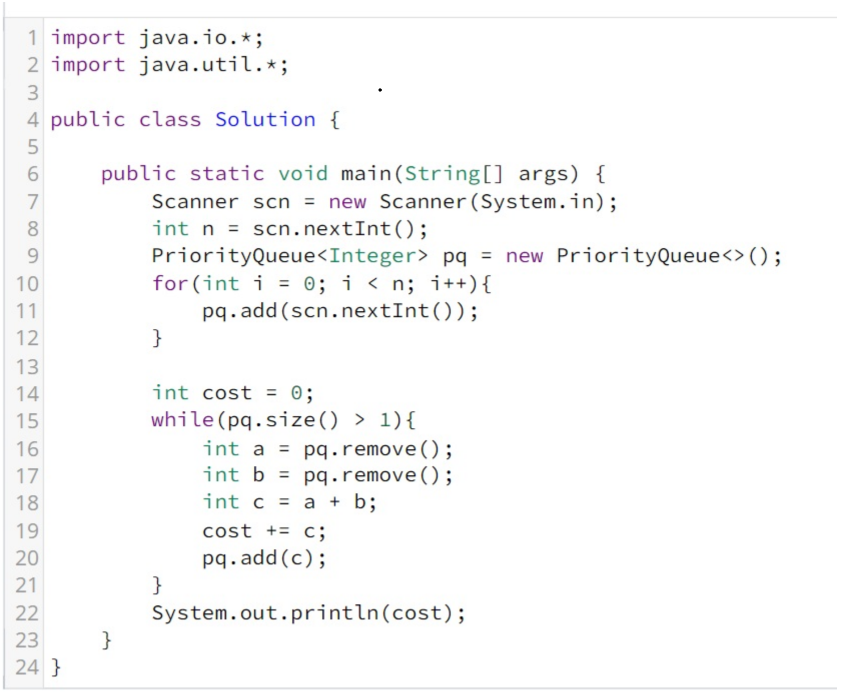
4 ropes



cost = 5 + 9 + 15 = 29

v1 = 6

v2 = 9

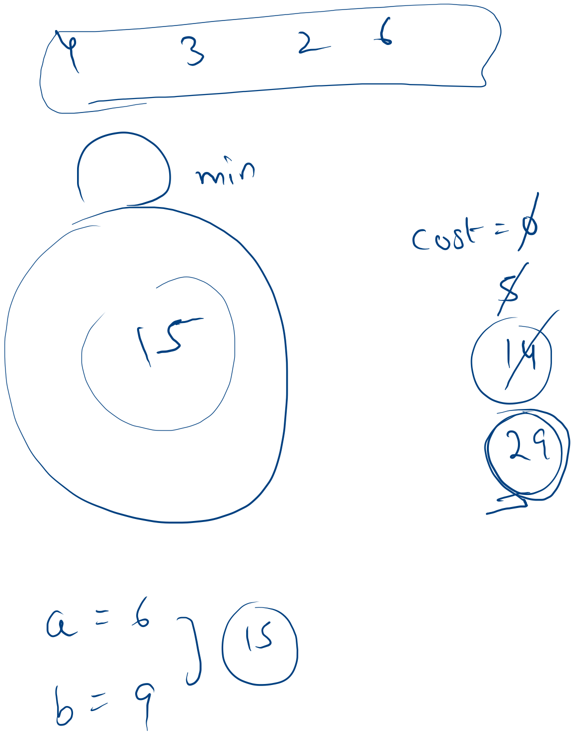


```java
import java.io.*;
import java.util.*;

public class Solution {

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();
        PriorityQueue<Integer> pq = new PriorityQueue<>();
        for(int i = 0; i < n; i++){
            pq.add(scn.nextInt());
        }

        int cost = 0;
        while(pq.size() > 1){
            int a = pq.remove();
            int b = pq.remove();
            int c = a + b;
            cost += c;
            pq.add(c);
        }
        System.out.println(cost);
    }
}
```



4 3 2 6
min
15
cost = ~~0~~
~~5~~
~~14~~
29
a = 6
b = 9
15