# First Negative Integer 2

Given an array **A[]** of size **N** and a positive integer **K**, find the first negative integer for each and every window(**contiguous subarray**) of size **K**.

**Sample Input 0**

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

K = 2

N = 5

# windows = N - K + 1

| -8 | 2 | 3 | -6 | 10 |
|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 |

peek < i - k + 1

i

-8 0 -6 -6

1st k.

add -ve idx

3

after k.

find prev.

remove un

add -ve idx

Wrong

```java
        Queue<Integer> qu = new LinkedList<>();
        int i = 0;
        while(i < k){
            if(A[i] < 0){
                qu.add(i);
            }
            i++;
        }
        while(i < n){
            //find prev ans
            if(qu.size() == 0){
                System.out.print(0 + " ");
            }else{
                System.out.print(A[qu.peek()] + " ");
            }

            //remove unnecessary
            if(qu.size() != 0 && qu.peek() < i+k-1){
                qu.remove();
            }
            if(A[i] < 0){
                qu.add(i);
            }
            i++;
        }
        if(qu.size() == 0){
            System.out.print(0 + " ");
        }else{
            System.out.print(A[qu.peek()] + " ");
        }

    }
```

k=2

-8 2 3 -6 10

0 1 2 3 4

i

4-1

0

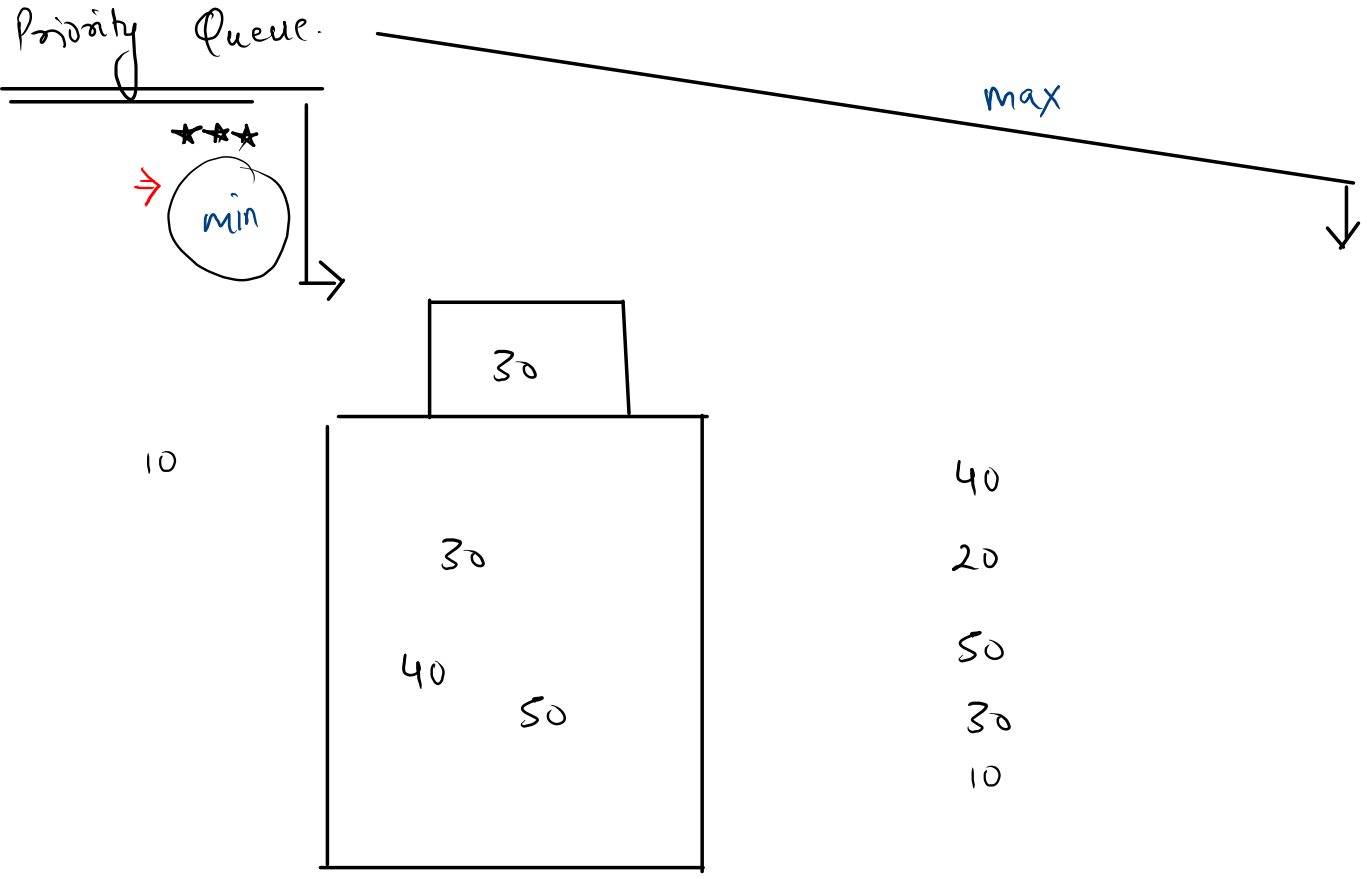
-8 ✓

0 < 2

input

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



```java
import java.util.PriorityQueue;
import java.util.*;

public class Main
{
    public static void main(String[] args) {
        //init
        PriorityQueue<Integer> pq = new PriorityQueue<>();      //min
        //add
        pq.add(40);
        pq.add(30);
        pq.add(30);
        pq.add(50);
        pq.add(20);
        pq.add(10);

        pq.remove();
        pq.remove();
        pq.remove();

        System.out.println(pq.peek());

    }
```

input

Main.java

```java
import java.util.PriorityQueue;
import java.util.*;

public class Main
{
    public static void main(String[] args) {
        //init
        PriorityQueue<Integer> pq = new PriorityQueue<>();      //min
        //add
        pq.add(40);
        pq.add(30);
        pq.add(30);
        pq.add(50);
        pq.add(20);
        pq.add(10);

        while(pq.size() != 0){
            System.out.print(pq.remove() + " ");
        }

    }
}
```

input

10 20 30 30 40 50

Main.java

```java
import java.util.PriorityQueue;
import java.util.*;

public class Main
{
    public static void main(String[] args) {
        //init
        PriorityQueue<Integer> pq = new PriorityQueue<>(Collections.reverseOrder());
        //add
        pq.add(40);
        pq.add(30);
        pq.add(30);
        pq.add(50);
        pq.add(20);
        pq.add(10);

        while(pq.size() != 0){
            System.out.print(pq.remove() + " ");
        }

    }
}
```

max

# priority queue basics

Sample Input 0

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

Sample Output 0

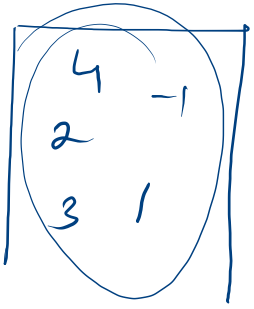
```
4
2
1
1
-1
```

T=5

add → pq → O(log n)

remove

peek → O(1)




1 ... logn } add.

n ... nlogn

nlogn } remove

2 nlogn  O(nlogn)

```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++){
            int x = scn.nextInt();
            pq.add(x);
            System.out.println(pq.peek());
        }
    }
}
```

# Maximum Product of Two Elements in an Array

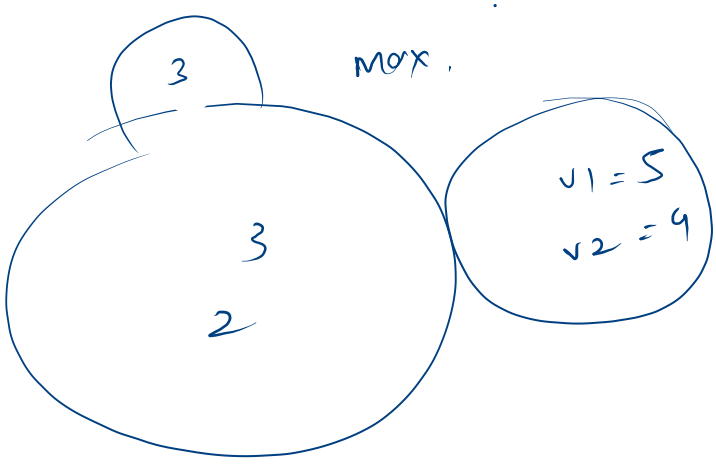
Given the array of integers nums, you will choose two different indices i and j of that array. Return the maximum value of (nums[i]-1)*(nums[j]-1).

### Sample Input 0

```
4
3
4
5
2
```

### Sample Output 0

```
12
```



```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<>((a,b)->b-a);
        int n = scn.nextInt();
        for(int i = 0; i < n; i++){
            pq.add(scn.nextInt());
        }
        int a = pq.remove();
        int b = pq.remove();
        System.out.println((a-1) * (b-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<>(Collections.reverseOrder());
        int n = scn.nextInt();
        for(int i = 0; i < n; i++){
            pq.add(scn.nextInt());
        }
        int a = pq.remove();
        int b = pq.remove();
        System.out.println((a-1) * (b-1));
    }
}
```

# 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
```

$(((G\,B)\,Y)\,R) = 35.$

4 3 2 6

7
13
15

$(((G\,B)\,Y)\,R)$

$7 + 13 + 15$

$= 35$

$(((B\,R)\,G)\,Y)$

5
9
15

29