Priority Queue.

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).

Constraints

2 <= nums.length <= 500

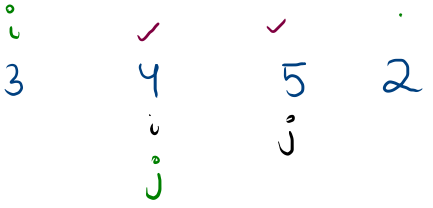
1 <= nums[i] <= 10^3

Sample Input 0

```
4
3
4
5
2
```

Sample Output 0

```
12
```



(4-1) * (5-1)

3 x 4 = 12

(3-1) x (4-1)

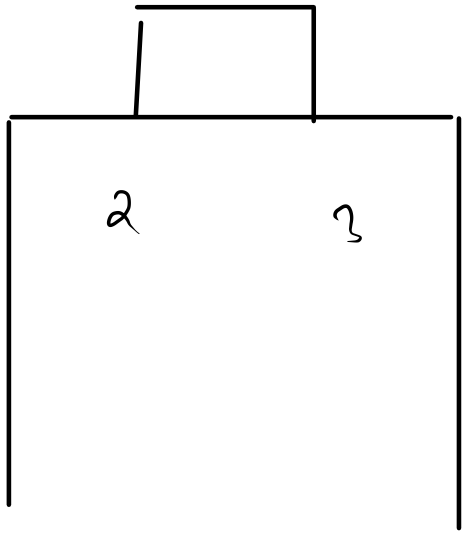
2 x 3 = 6

1. sort ✗

2 3 4 5 nlogn

2. PQ

↳ max



v1 = 5

v2 = 4

ans = $(v_1 - 1) * (v_2 - 1)$

max 1 / max 2

```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<>(Collections.reverseOrder());
        while(n-- > 0){
            pq.add(scn.nextInt());
        }
        int v1 = pq.remove();
        int v2 = pq.remove();
        System.out.println((v1-1) * (v2-1) );
    }
}
```

n

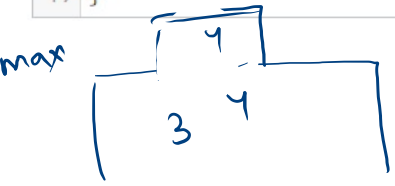
4

3 ✓

5 ✓

7 ✓

4 ✓



minimum digits

Problem | Submissions | Leaderboard | Discussions

Given an array of **digits** (values are from 0 to 9), find the **minimum possible sum of two numbers** formed from digits of the array. All digits of the given array must be used to form the two numbers.

Any combination of digits may be used to form the two numbers to be summed. Leading zeroes are permitted.

If forming two numbers is **impossible** (i.e. when n==0) then the "sum" is the value of the only number that can be formed.

Sample Input 0

```
6
6 8 4 5 2 3
```

Sample Output 0

```
604
```

6 8 4 5 2 3

2 numbers

a + b = sum

4 8 6
3 2 5
8 1 1

a = 8 4 5 2 3
b = 6
8 4 5 2 9

a 6 8 4 5
b 2 3
6 8 6 8

6 8 4 5 2 3

6 8 4 5 2 + 3 = 6 8 4 5 5.

6 8 4 + 5 2 3 = 1207

(486) 235

6 8 4 5 2 3

min.

a → "2 4 6"

b → 3 5 8

→ int / Integer

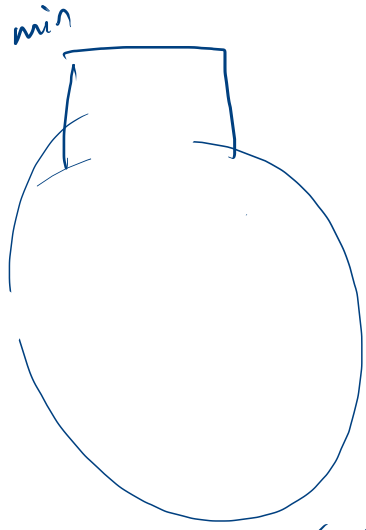
a

b

```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<>();
        while(n-- > 0){
            pq.add(scn.nextInt());
        }
        String a = "";
        String b = "";
        int itrOddEven = 0;
        while(pq.size() != 0){
            if(itrOddEven % 2 == 0 ){
                a += pq.remove();
            }else{
                b += pq.remove();
            }
            itrOddEven++;
        }
        long v1 = Long.parseLong(a);
        long v2 = Long.parseLong(b);
        System.out.println(v1 + v2);
    }
}
```

```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<>();
        while(n-- > 0){
            pq.add(scn.nextInt());
        }
        String a = "";
        String b = "";
        int itrOddEven = 0;
        while(pq.size() != 0){
            if(itrOddEven % 2 == 0 ){
                a += pq.remove();
            }else{
                b += pq.remove();
            }
            itrOddEven++;
        }
        long v1 = Long.parseLong(a);
        long v2 = Long.parseLong(b);
        System.out.println(v1 + v2);
    }
}
```

6 8 4 5 2 3



a → "246"

b → "358"

v1 → 246

v2 → 358

ic = 0 1 2 3 4 5

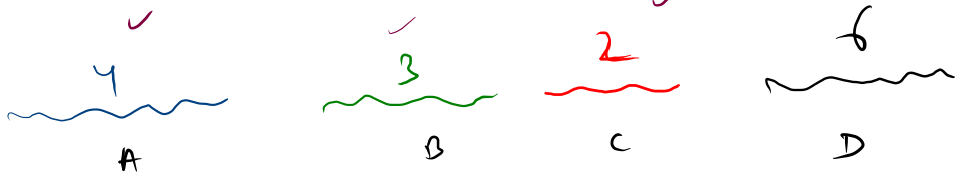
Minimum Cost of ropes 3

Problem | Submissions | Leaderboard | Discussions

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.

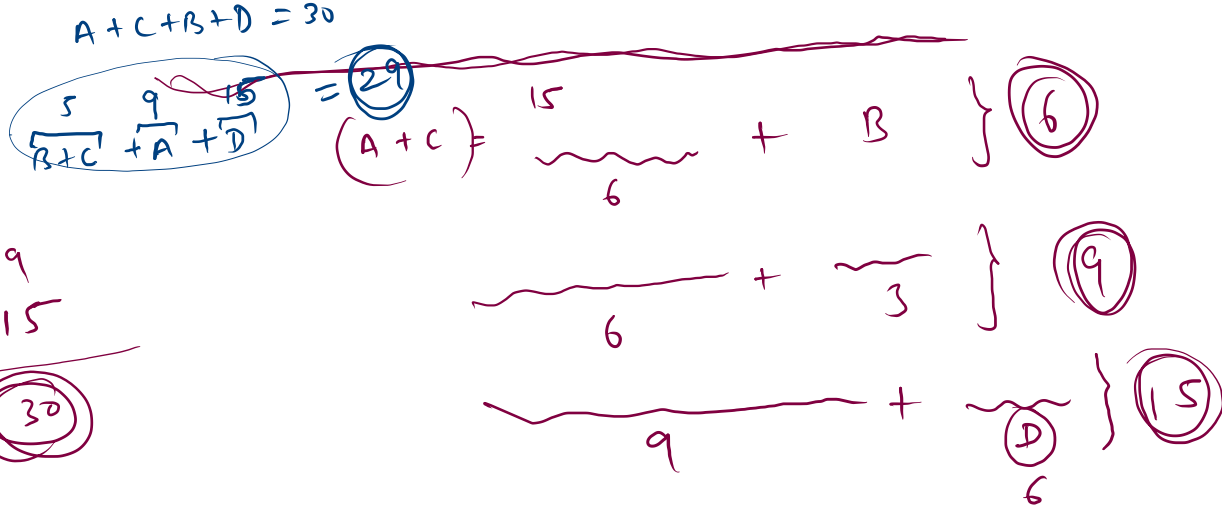
Sample Input 0

```
4
4 3 2 6
```



Sample Output 0

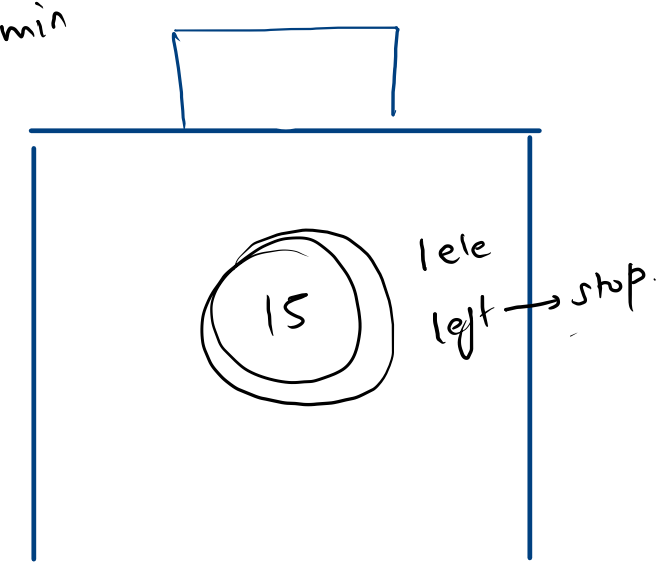
```
29
```



4 3 2 6

ans = 5 + 9 + 15

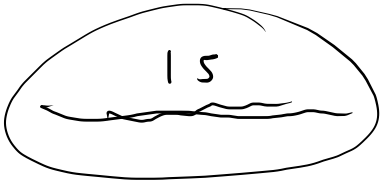
29



v1 = 6

v2 = 9

15



ans = 5 + 9 + 15

29

2 + 3 = 5

4 + 5 = 9

6 + 9 = 15

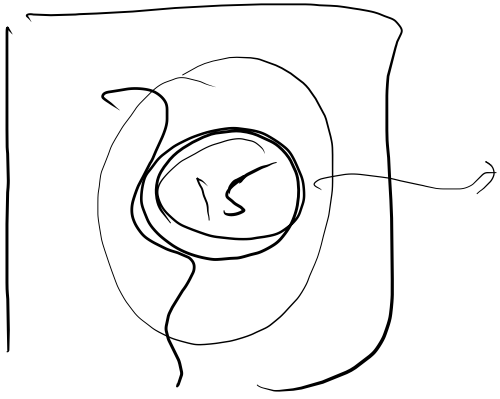
4 2 3 6

cost = 2+3
+9
+15

29

v1 = 9

v2 = 6



```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();
        int [] A = new int[n];
        for(int i = 0; i < n; i++){
            A[i] = scn.nextInt();
        }
        PriorityQueue<Integer> pq = new PriorityQueue<>();
        for(int ele : A){
            pq.add(ele);
        }
        int cost = 0;
        while(pq.size() > 1){
            int a = pq.remove();
            int b = pq.remove();
            cost += (a+b);
            pq.add(a+b);
        }
        System.out.println(cost);

    }
}
```

4 2 3 6

cost = 5 + 9 + 15 = 29

a = 6

b = 9

subtract numbers 1

Problem | Submissions | Leaderboard | Discussions

You are given a non-negative integer array **nums**. In one operation, you must:

- Choose a positive integer x such that x is less than or equal to the **smallest non-zero element in nums**.
- Subtract x from every **positive** element in **nums**.

Return the **minimum** number of operations to make every element in **nums** equal to 0.

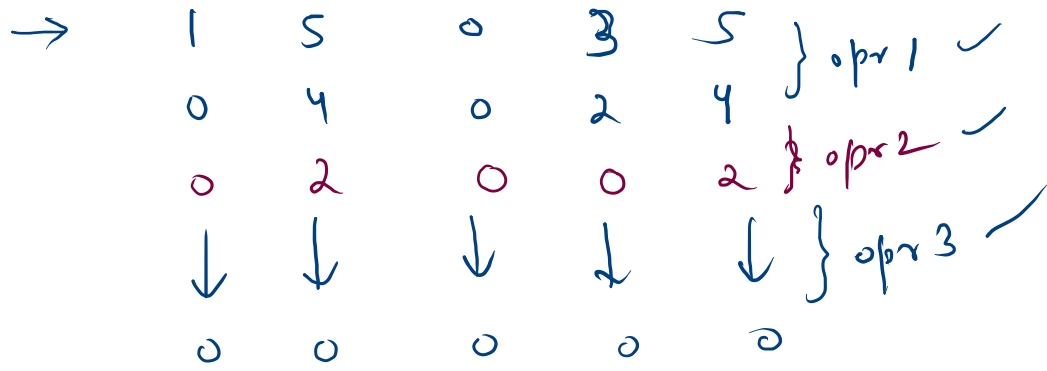
$x = 2$

Sample Input 0

```
5
1 5 0 3 5
```

Sample Output 0

```
3
```




1 5 0 3 5
0 4 0 2 4

$x = 1$

1	1	5	0	1	4	2
0	0	4	0	0	3	1

$x = 3$ (1 move)

3	3	3	3
0	0	0	0

2	4	3	5
0	2	1	3
0	1	0	2
0	0	0	1
0	0	0	0

$x = 2$

$x = 1$

$x = 1$

$x = 1$

opr = 4

opr = ∅ ✗ ✗ ✗ (4)

					x			
n = 2	3	2	4	0	4	0	3	5
n = 1	1	0	2	0	2	0	1	3
n = 1	0	0	1	0	1	0	0	2
n = 1	0	0	0	0	0	0	0	1
	0	0	0	0	0	0	0	0

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

public class Solution {

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        HashSet<Integer> hs = new HashSet();
        int n = scn.nextInt();
        while(n-- > 0){
            int val = scn.nextInt();
            if(val > 0){
                hs.add(val);
            }
        }
        System.out.println(hs.size());
    }
}
```