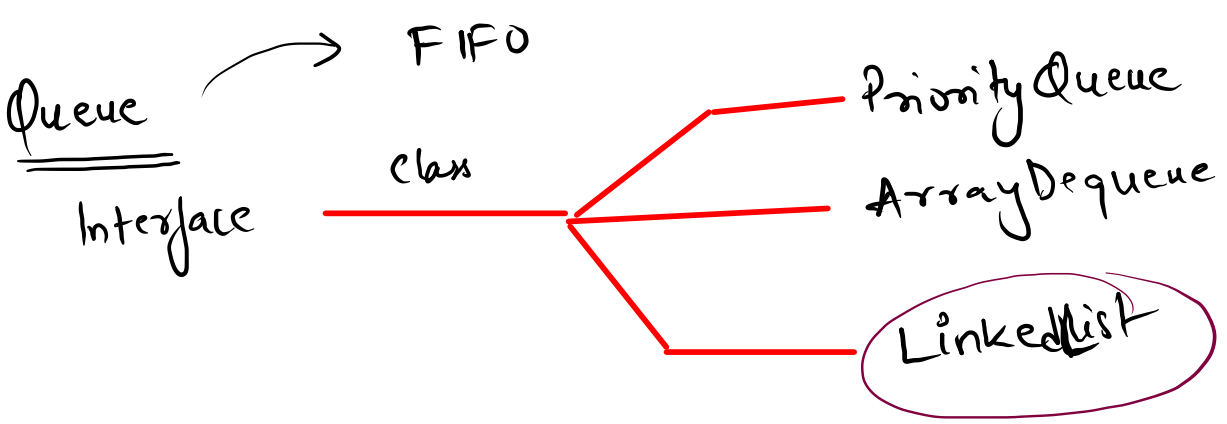
Revision.



add()
remove()
peek() ⇝ front
size()

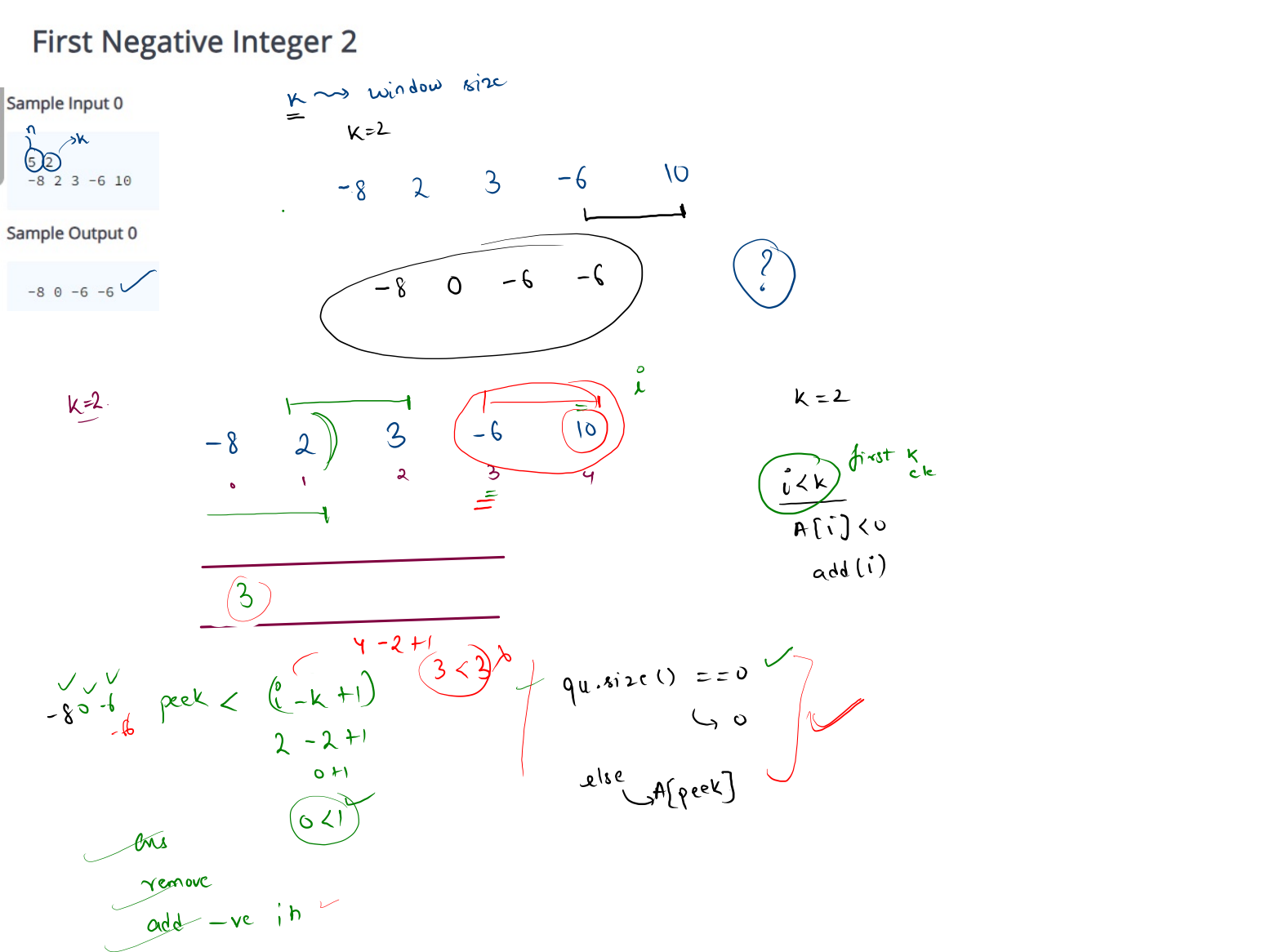
First Negative Integer 2

Sample Input 0

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

Sample Output 0

```
-8 0 -6 -6
```



```
        Queue<Integer> qu = new LinkedList<>();
        int i = 0;

        while(i < k){
            if(A[i] < 0){
                qu.add(i);
            }
            i++;
        }

        while(i < n){
            if(qu.size() == 0){
                System.out.print(0 + " ");
            }
            else{
                System.out.print(A[qu.peek()] + " ");
            }

            while(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=3

k=3

i - k + 1

3 - 3 + 1

0 + 1

-8 ✓

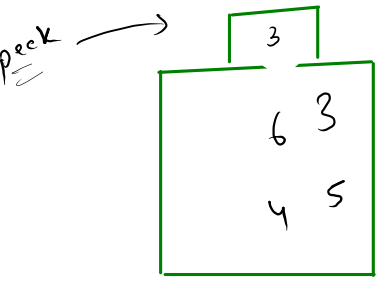
0

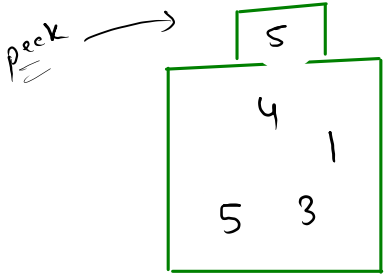
0 < 1

Priority Queue.

↳ DS.

- min
- max
- customize.





String priority.

min → len
max → len
min → lexo

Sorting
Comparator
lambda.

1 →
n → nlogn

log(n) → insert
O(1) → remove.

pq → Heap.

- binary.
- Tree based
  - Tree.

down heapify
up heapify

PQ

```java
import java.util.PriorityQueue;
import java.util.*;
public class Main
{
    public static void main(String[] args) {
        PriorityQueue<Integer> pq = new PriorityQueue<>();

        //add
        pq.add(50);
        pq.add(70);
        pq.add(10);
        pq.add(30);

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


        

    }
}
```

```java
import java.util.PriorityQueue;
import java.util.*;
public class Main
{
    public static void main(String[] args) {
//      PriorityQueue<Integer> pq = new PriorityQueue<>();      //min

//      PriorityQueue<Integer> pq = new PriorityQueue<>(Collections.reverseOrder());
        PriorityQueue<Integer> pq = new PriorityQueue<>((a,b)->{
            return b-a;
        });      //max

        //add
        pq.add(50);
        pq.add(70);
        pq.add(10);
        pq.add(30);

        while(pq.size() != 0){
            System.out.println(pq.remove());
        }
```

1046. Last Stone Weight

Easy 5444 93 Add to List Share

You are given an array of integers `stones` where `stones[i]` is the weight of the i^{th} stone.

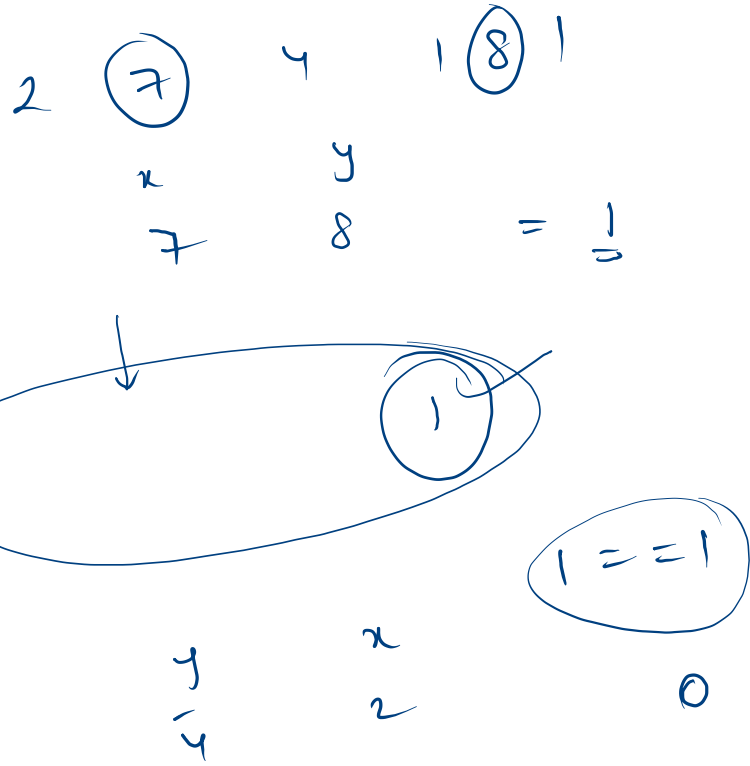
We are playing a game with the stones. On each turn, we choose the **heaviest two stones** and smash them together. Suppose the heaviest two stones have weights `x` and `y` with `x <= y`. The result of this smash is:

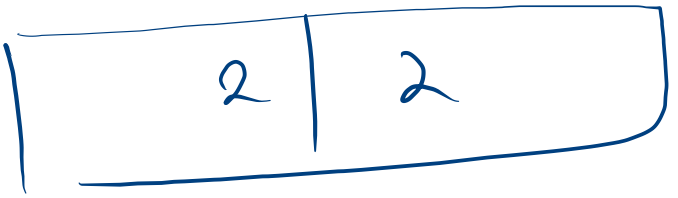
- If `x == y`, both stones are destroyed, and
- If `x != y`, the stone of weight `x` is destroyed, and the stone of weight `y` has new weight `y - x`.

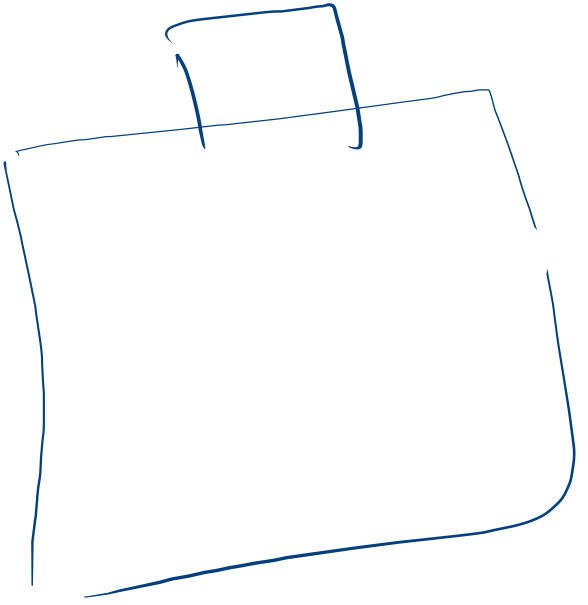
At the end of the game, there is **at most one** stone left.

Return *the weight of the last remaining stone*. If there are no stones left, return `0`.

```
Input: stones = [2,7,4,1,8,1]
Output: 1
Explanation:
```



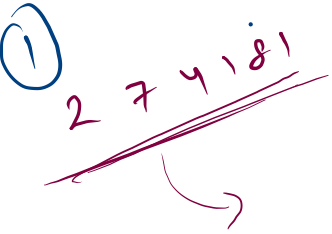




x y

2 2 = 0

2




```java
class Solution {
    public int lastStoneWeight(int[] stones) {
        PriorityQueue<Integer> pq = new PriorityQueue<>((a,b)->{
            return b-a;           // max pq
        });

         for(int ele : stones){
             pq.add(ele);
         }

         while(pq.size() > 1){
             int y = pq.remove();     //8
             int x = pq.remove();      // 7

             if(y-x > 0){
                 pq.add(y-x);
             }
         }

         return pq.size() == 0 ? 0 : pq.peek();

    }
}
```

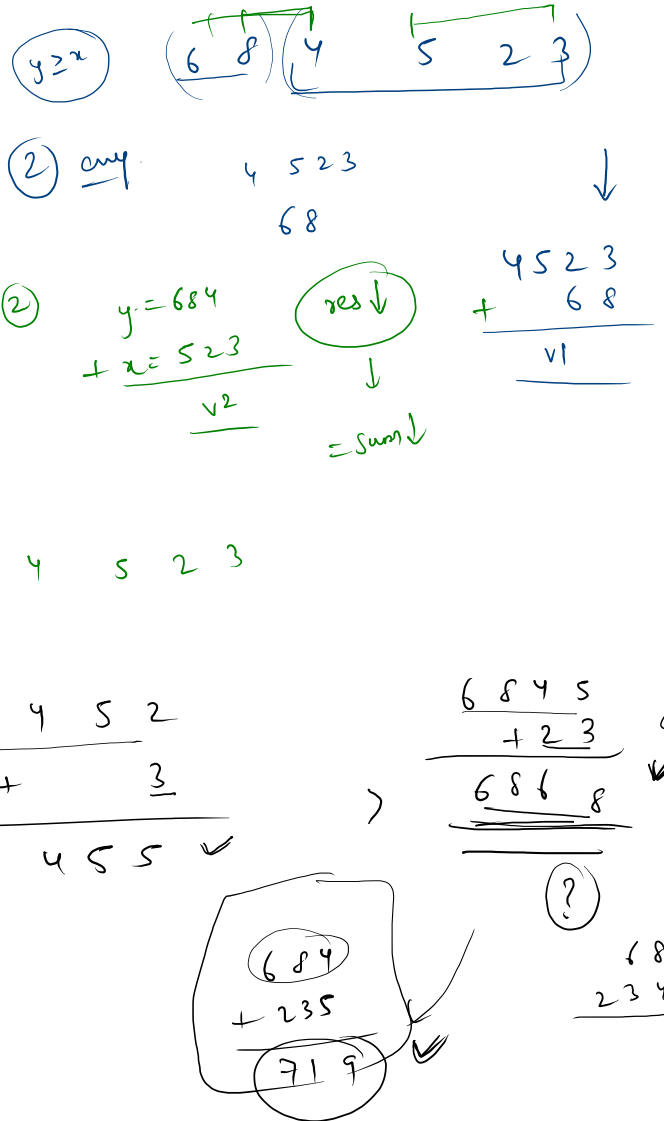
minimum digits

Sample Input 0

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

Sample Output 0

```
604
```



balance

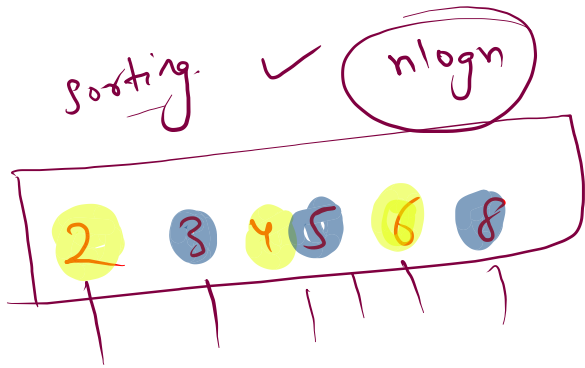
684 523

3 3

684
523
1207

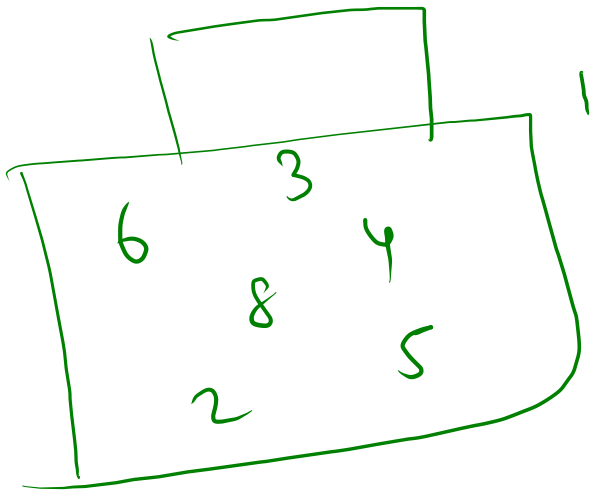
682
453
1135

sorting ✓ nlogn



246
358
604

6 8 4 5 2 3



Integer (min)

form 2 long.

2 4 6

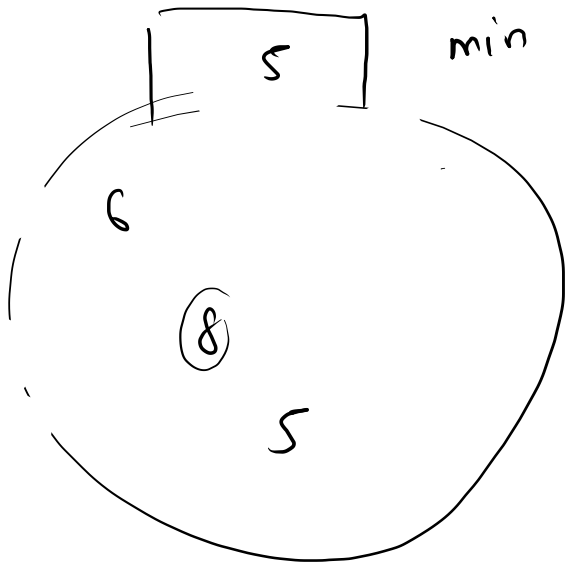
3 5 8

---

ans.

```java
PriorityQueue<Integer> pq = new PriorityQueue<>();          //min

for(int ele : A){
    pq.add(ele);
}

long v1 = 0;
long v2 = 0;

boolean even = true;

while(pq.size() > 0){
    if(even){
        v1 = v1 * 10 + pq.remove();
        even = false;
    }
    else{
        v2 = v2 * 10 + pq.remove();
        even = true;
    }
}
System.out.println(v1 + v2);
```

6 8 4 5 2 3



Subtract numbers.

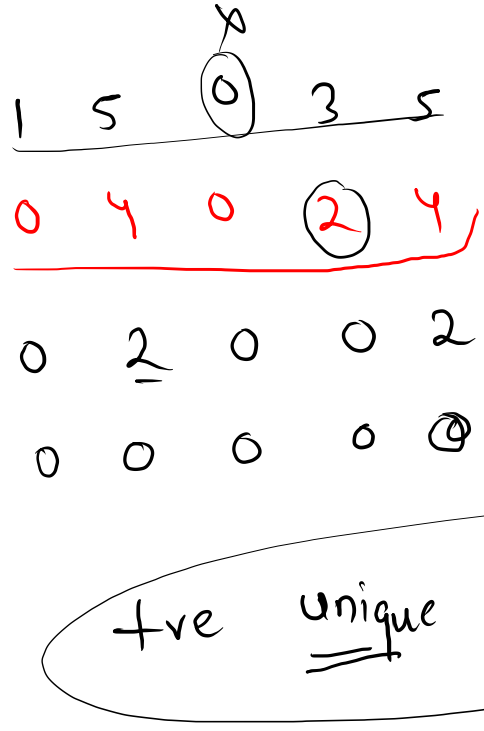
Smallest positive

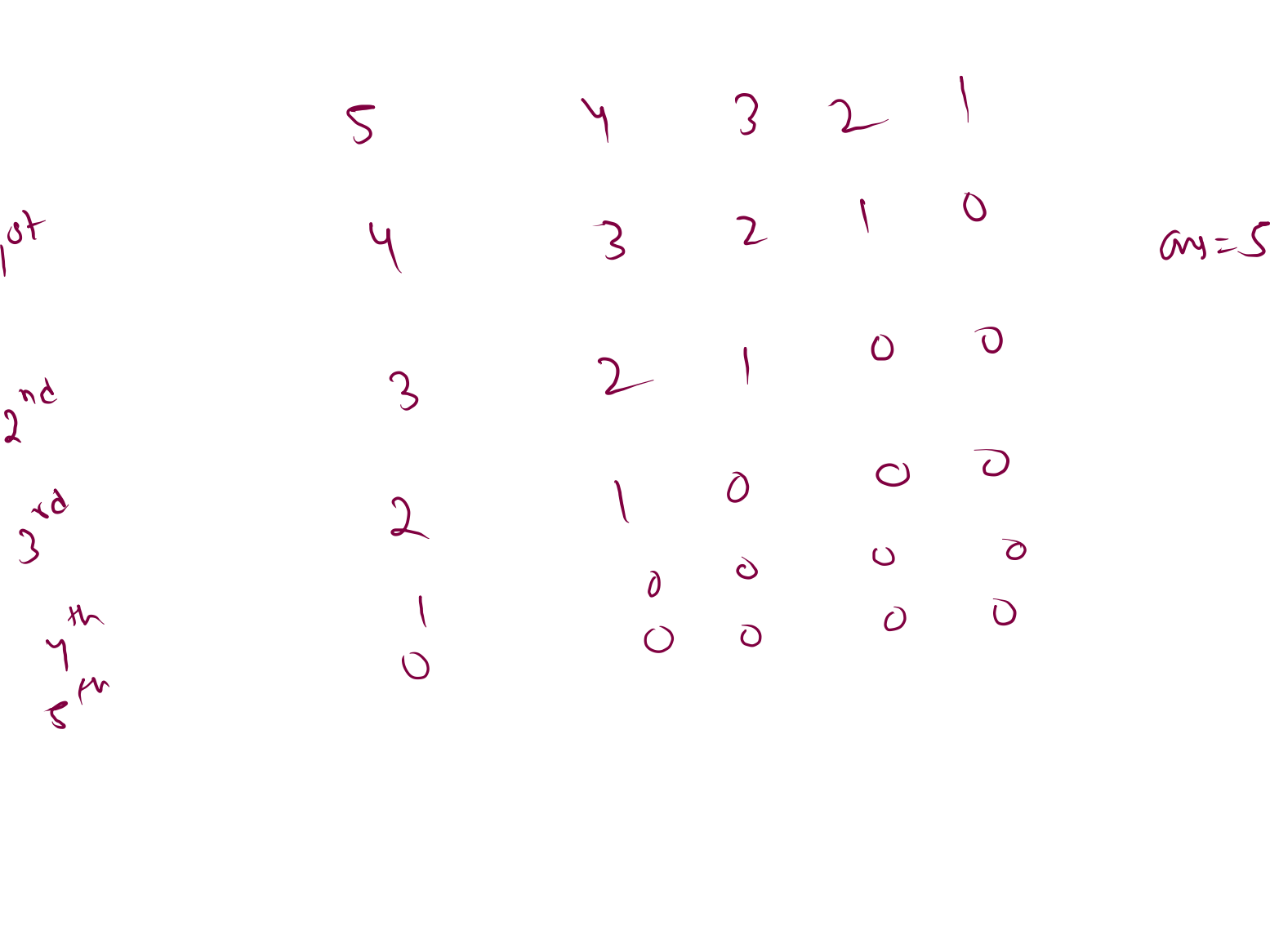
Sample Input 0

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

Sample Output 0

```
3
```

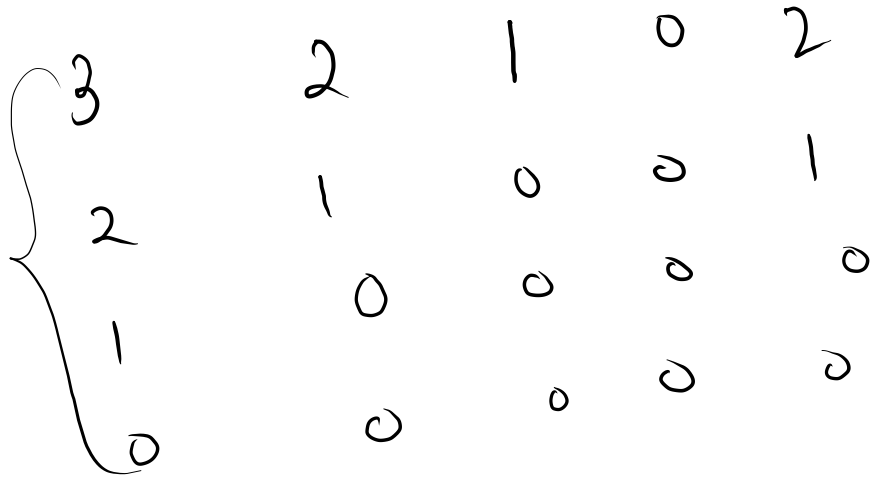



5 4 3 2 1
1st 4 3 2 1 0
ans=5
2nd 3 2 1 0 0
3rd 2 1 0 0 0
0 0 0 0
4th 1 0 0 0 0
5th 0

(4)

5 4 3 2 4

ans = 4



.

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