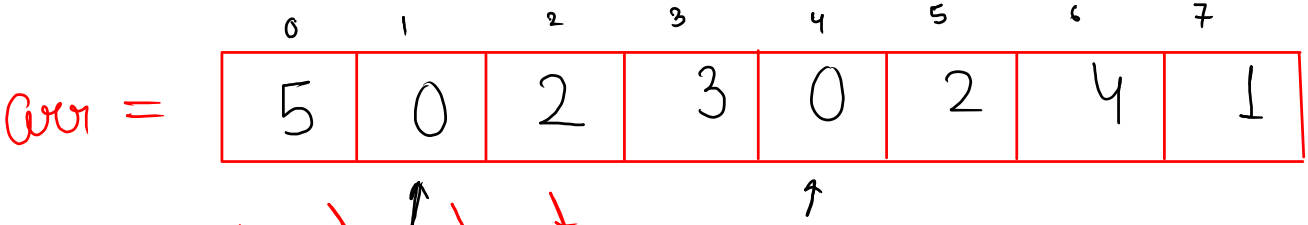
minimum digits



num1 = 0 1 2 4

num2 = 0 2 3 5

ans = 124 + 235
= smallest

PQ

minHeap

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

Code

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

    long num1 = 0;
    long num2 = 0;
    while ( pq.size() > 0 ) {
        if ( pq.size() % 2 == 0 ) {
            num1 = num1 * 10 + pq.peek();
            pq.poll();
        } else {
            num2 = num2 * 10 + pq.peek();
            pq.poll();
        }
    }
    return num1 + num2;
}
```

gm?

$T.C = n \log(n)$

Dry run

arr :- 1 2 3 4 5 6 7

num1 = 1357

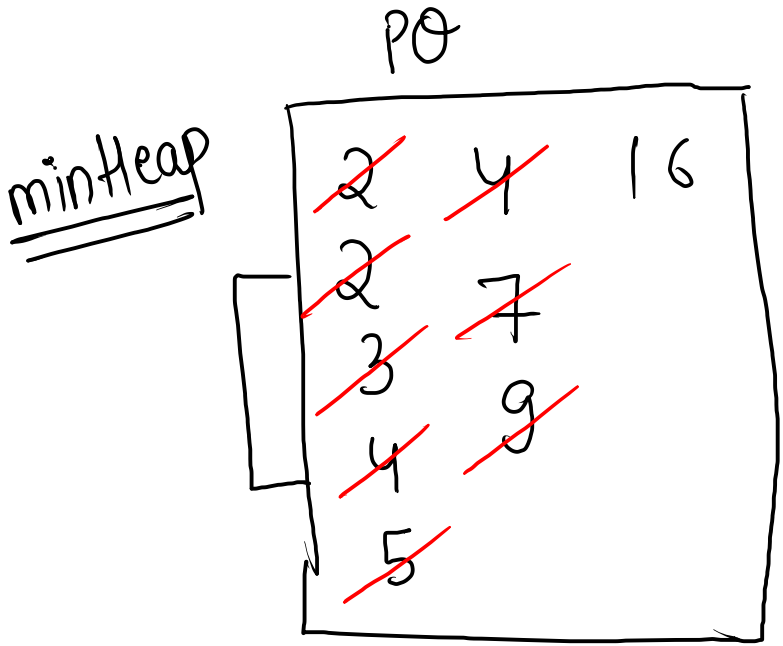
num2 = 246

Minimum Cost of ropes 3

arr =

2	3	5	4	2

Cost = 4 + 7 + 9 + 16 = 36



psudo code

1) Create minHeap
2) fill up the PQ
3) loop until PQ size > 1
    3.1) remove 2 elements
    3.2) add the elements and store them in answer
    3.3) added elements should be added in PQ

Code

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

    int cost = 0;
    while ( pq.size() > 1 ) {
        int num1 = pq.poll();
        int num2 = pq.poll();
        int sum = (num1 + num2);

        cost = cost + sum;
        pq.add( sum );
    }
    return cost;
}
```

$ToC = O(N \log(N))$

subtract numbers 1

arr:- [1 , 5 , 0 , 3 , 5 , 5 , 1 , 3]

step1 → choose a minimum no. which is not zero (x)

step2 → substract x from every +ve element

arr = [0 , 4 , 0 , 2 , 4 , 4 , 0 , 2] , x = 1

arr = [0 , 2 , 0 , 0 , 2 , 2 , 0 , 0] , x = 2

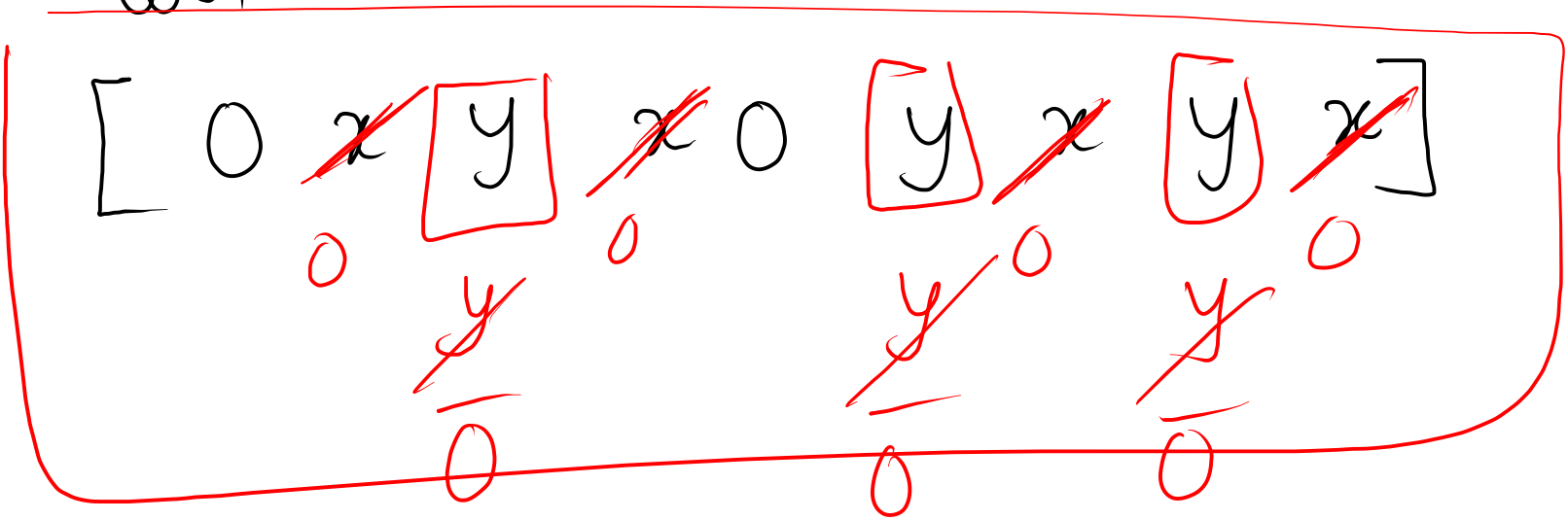
arr = [0 , 0 , 0 , 0 , 0 , 0 , 0 , 0] , x = 2

ans = 3

$x = 2$

$y = 5 - 2 = 3$

Arr

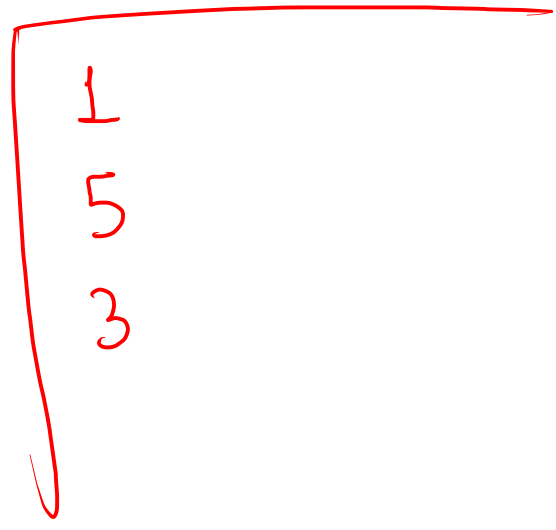


Key observation:-

in 1 step, we are removing all 1 type of element (non-zero)

arr:- [1 , 5 , 0 , 3 , 5 , 5 , 1 , 3]

Hashset



size = 3

code

T.C = O(N)

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

duplicacy is removed