Minimum Cost of ropes 3

$$arr = [\underset{0}{2}, \underset{1}{3}, \underset{2}{5}, \underset{3}{4}, \underset{4}{2}]$$

PQ

minHeap

rope1 = ~~2~~ ~~3~~ ~~4~~ 7

rope2 = ~~2~~ ~~4~~ ~~5~~ 9

cost = 4 + 7 + 9 + 16

= 36

gmp

psudo code

1) create min heap

2) add all elements of arr in pq

3) loop until only 1 element left in PQ

3.1) get 2 smallest ropes from PQ

3.2) ans += (num1 + num2)

Imp → 3.3) add (num1+num2) in P

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();
    }
    int ans = minCost(arr, n);
    System.out.println(ans);
}
public static int minCost(int[] arr, int n) {
    PriorityQueue<Integer> pq = new PriorityQueue<>();
    for (int i = 0; i < n; i++) {
        pq.add( arr[i] );
    }

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

        cost = cost + rope;

        pq.add( rope );
    }
    return cost;
}
```

T.C = O(nlogn)

S.C = O(n)

subtract numbers 1

	0	1	2	3	4	5	6	7	8
arr =	1	5	0	3	5	5	1	3	5

step 1 :- choose a minimum no, which is non-zero (x)

step 2 :- substract x from every no. except zero

ans = 3

process

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

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

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

approach

arr = [1, 5, 0, 3, 5, 5, 1, 3, 5]
0 1 2 3 4 5 6 7 8

x, y, z — non-zero

[x, z, 0, y, z, z, x, y, z]

~~y~~ [a, b, 0, 0, b, b, a, 0, b]

a, b

d [0, 0, 0, 0, d, d, 0, 0, d]

set

1
5
3

Key observation :-

In 1 step, we are removing all 1 type of elements (non-zero)

Note:- our ans is no. of non-zero elements (without duplicacy)

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();
    }
    int ans = substractOne(arr, n);
    System.out.println(ans);
}
public static int substractOne(int[] arr, int n) {
    HashSet<Integer> set = new HashSet<>();
    for (int i : arr) {
        if ( i > 0 ) {
            set.add(i);
        }
    }
    return set.size();
}
```

T.C = O(n)

S.C = O(n)

duplicacy is removed

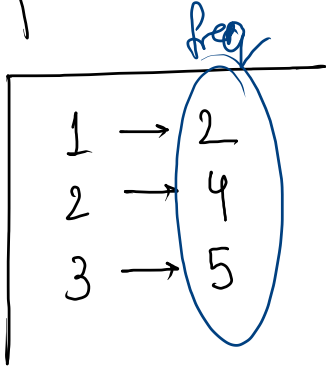
Reduce Array Size to the half

arr = [3, 2, 3, 3, 2, 1, 2, 1, 2, 3, 3]

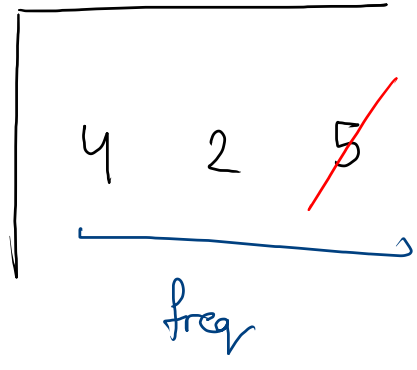
task :- perform mini. steps

↳ in 1 step :- pick any no. and remove all the occ. of that no.

hash map



PQ (max heap)



psudo code

1) create hashmap
2) calculate freq
3) create max heap
4) store all the freq of map
5) loop until array size become half or less
    5.1) size = size - pq.poll()
        Count++
6) return count.

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();
    }
    int ans = reduceSize(arr, n);
    System.out.println(ans);
}
public static int reduceSize(int[] arr, int n) {

    HashMap<Integer, Integer> map = new HashMap<>();
    for (int i = 0; i < n; i++)
        map.put( arr[i], map.getOrDefault( arr[i], 0 ) + 1 );

    PriorityQueue<Integer> pq = new PriorityQueue<>(Collections.reverseOrder());
    for (int val : map.values()) {
        pq.add( val );
    }

    int size = n;
    int count = 0;
    while ( size > n / 2 ) {
        size = size - pq.poll();
        count++;
    }
    return count;
}
```

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

$\cong O(n \log n)$

$S.C = O(n + n)$

$\cong O(n)$