minimum digits

(smallest no. will always have digits in ascending order.)

arr = [2 , 3 , 0 , 9 , 0 , 8]

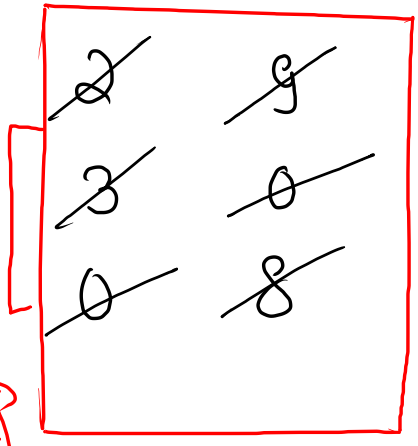
indices: 0 1 2 3 4 5

num1 = 0 2 8

num2 = 0 3 9

6 7

dry run



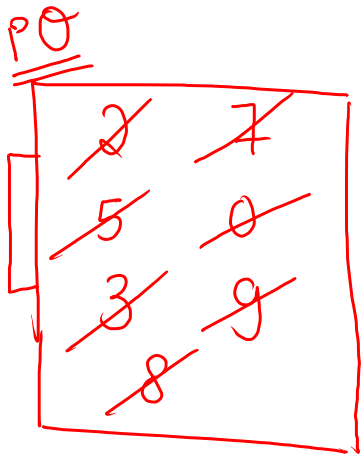
rem = ~~0~~ ~~0~~ ~~2~~ ~~3~~ ~~8~~ 9

num1 = 28

num2 = 39

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(minimumDigits(arr, n));
}
public static long minimumDigits(int[] arr, int n) {
    PriorityQueue<Integer> pq = new PriorityQueue<>();
    for (int i : arr) {
        pq.add(i);
    }

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



Size = ~~7~~ ~~6~~ ~~5~~ ~~4~~ ~~3~~ ~~2~~ ~~1~~ 0

rem = ~~0~~ ~~2~~ ~~3~~ ~~5~~ ~~7~~ ~~8~~ 9

num1 = 379

num2 = 258

Minimum Cost of ropes 3

arr = [2 , 5 , 4 , 3 , 7] 5 , 9 , 12 21

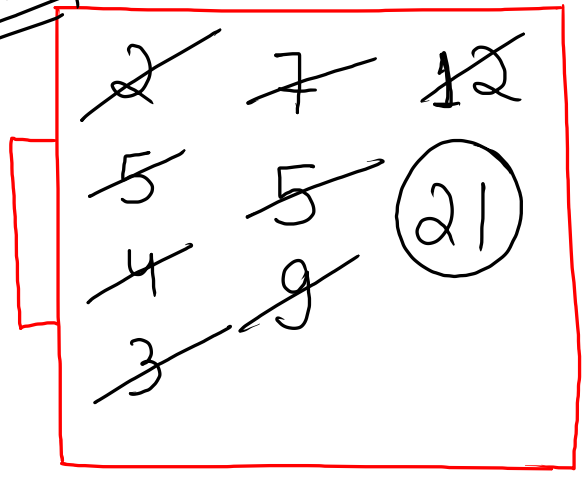
Cost = 7 + 11 + 14 + 21 = 53

Cost = 5 + 9 + 12 + 21 = 47

arr = [2 , 5 , 4 , 3 , 7]

0 1 2 3 4

minHeap



cost = 0 + 5 + 9 + 12 + 21

Note:- after combining 2 ropes, make sure to add that rope in PQ again

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 rope1 = pq.poll();
        int rope2 = pq.poll();
        cost += rope1 + rope2;
        pq.add( rope1 + rope2 );
    }
    return cost;
}
```

T.C = O(n log(n))

subtract numbers 1

arr = [1 , 5 , 0 , 3 , 5 , 5 , 1 , 3]
(indices: 0 1 2 3 4 5 6 7)

---

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

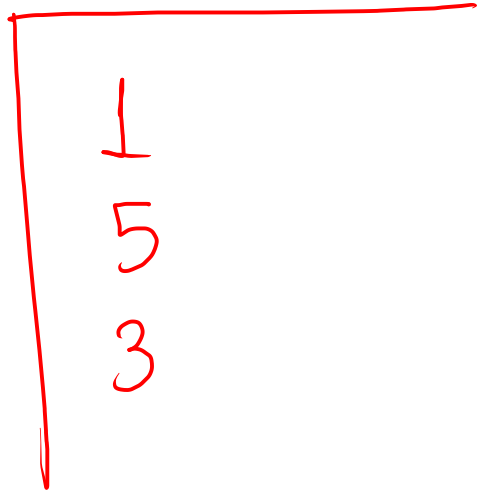
x = 2 , arr = [0 , 2 , 0 , 0 , 2 , 2 , 0 , 0] ✓

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

ans = no. of operations = 3

ans :- no. of non-zero unique elements

hashset



set.size() = 3

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

Revision :-

- ✓ sorting, lambda function
- ✓ arrays, subarrays (Kadan's algo)
- 2 pointers
- Prefix array
- Arrays as Hashmap
- 2d arrays
- Strings, substring
- Binary Search (BSLB, BSUB)
- Arraylist
- Stack
- Hashmap, hashset
- Queue
- PQ

→ Sorting

- Bubble sort
- Incertion sort
- Selection sort

- Arrays.sort(arr)
- Arrays.sort(arr, Collections.reverseOrder())

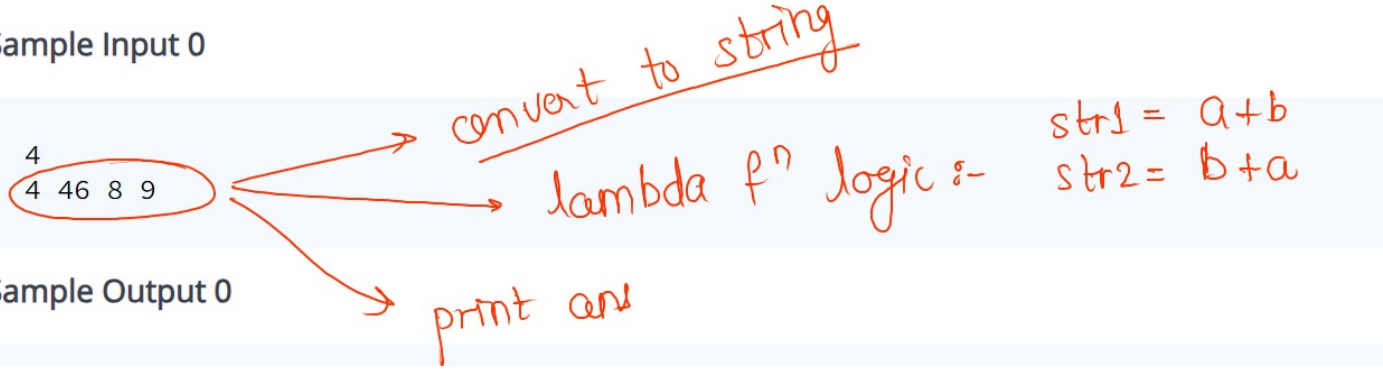
```
Arrays.sort(arr, (a, b) -> {
    return
});
```

- return a-b // ↑ing order based on value
- return b-a // ↓ing " " "
- return -1 // a will print first
- return +1 // b " " "

form the largest no.

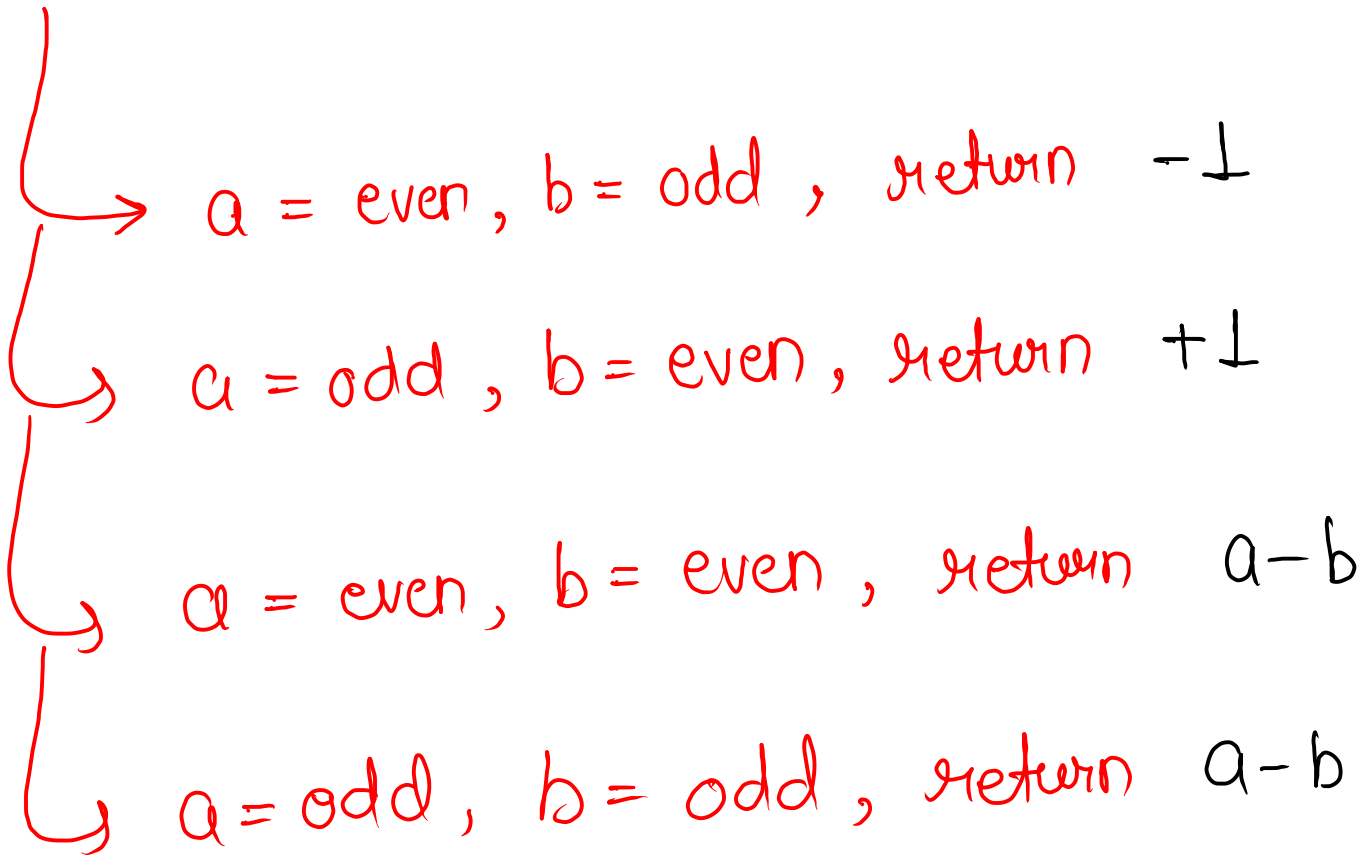
Sample Input 0

```
4
4 46 8 9
```



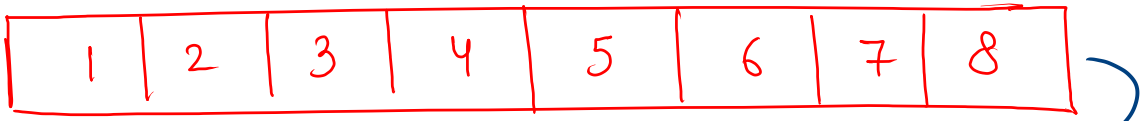
Sample Output 0

Sort Array By Parity

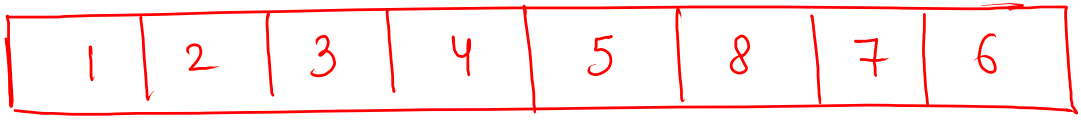


⇒ Rotate Right (k = 3)

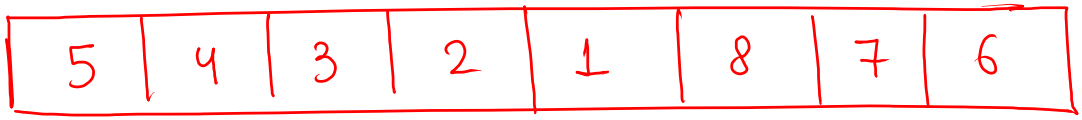
arr:-



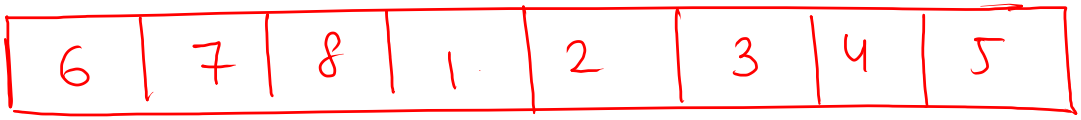
step 1



step 2



step 3



observation

1 2 3 4 5 6 ,	K = 0 , 6 , 12 . . —
6 1 2 3 4 5 ,	K = 1 , 7 , 13 . — —
5 6 1 2 3 4 ,	K = 2 , 8 , 14 . — —
4 5 6 1 2 3 ,	K = 3 , 9 , 15 . — .
3 4 5 6 1 2 ,	K = 4 , 10 , 16 . — —
2 3 4 5 6 1	K = 5 , 11 , 17 . — —

K = 14

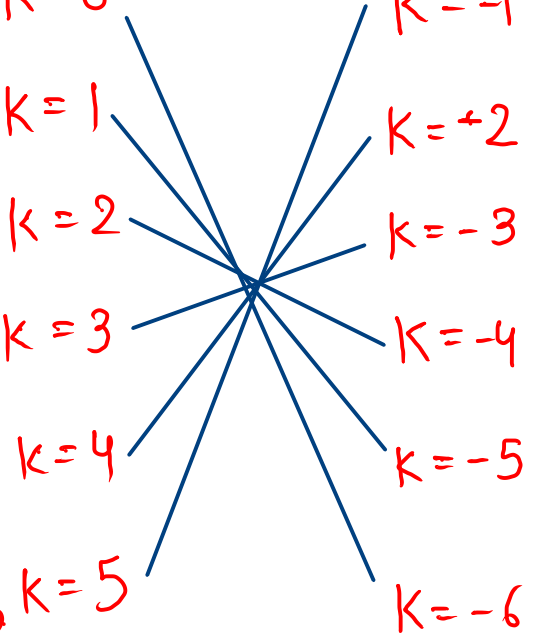
$$K = K \% n$$

$$= 14 \% 6$$

$$= 2$$

Observation

1 2 3 4 5 6 ,	K=0		K=-1	, 2 3 4 5 6 1
6 1 2 3 4 5 ,	K=1		K=-2	, 3 4 5 6 1 2
5 6 1 2 3 4 ,	K=2		K=-3	4 5 6 1 2 3
4 5 6 1 2 3 ,	K=3		K=-4	5 6 1 2 3 4
3 4 5 6 1 2 ,	K=4		K=-5	6 1 2 3 4 5
2 3 4 5 6 1 ,	K=5		K=-6	1 2 3 4 5 6



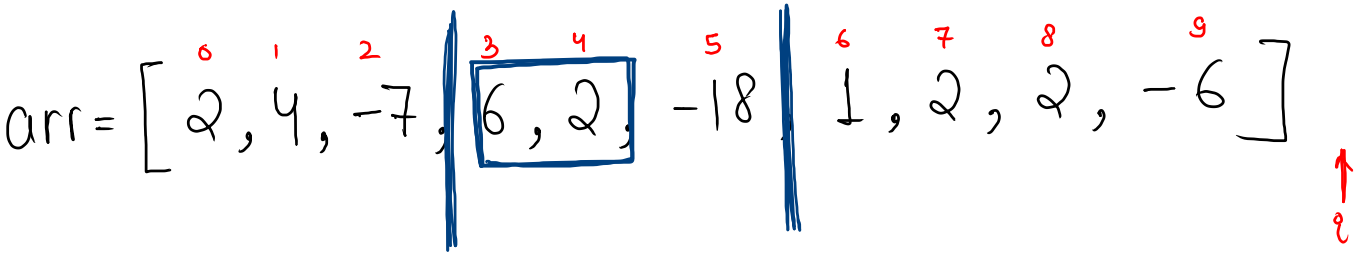
$$K = K + n$$

$K = -4$

$K = -4 + 6$

$= 2$

Max Subarray 2

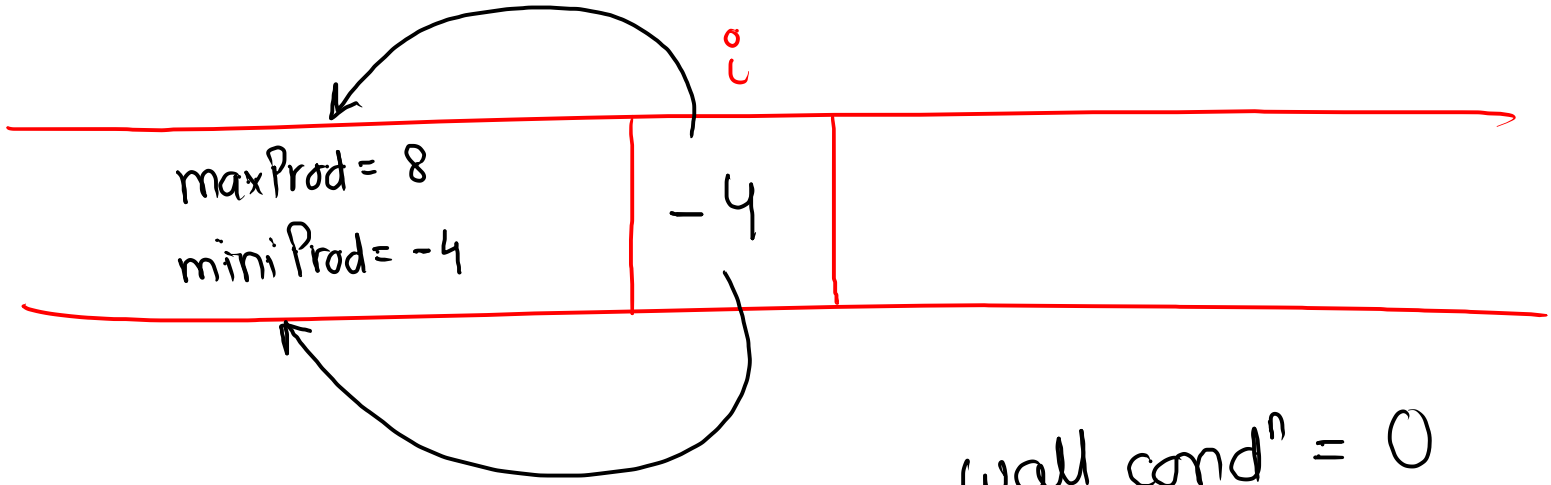



```java
public static int maxSumSubarray(int[] arr, int n) {
    int sumSoFar = 0;
    int maxSum = Integer.MIN_VALUE;
    for (int i = 0; i < n; i++) {

        if ( sumSoFar < 0 ) {
            sumSoFar = arr[i];
        } else {
            sumSoFar += arr[i];
        }

        if ( sumSoFar > maxSum ) {
            maxSum = sumSoFar;
        }
    }
    return maxSum;
}
```

sumSoFar = 0 2 6 -1 6 8 -10 1 3 5 -1

maxSum = -∞ 2 6 8

Maximum Product Subarray 2

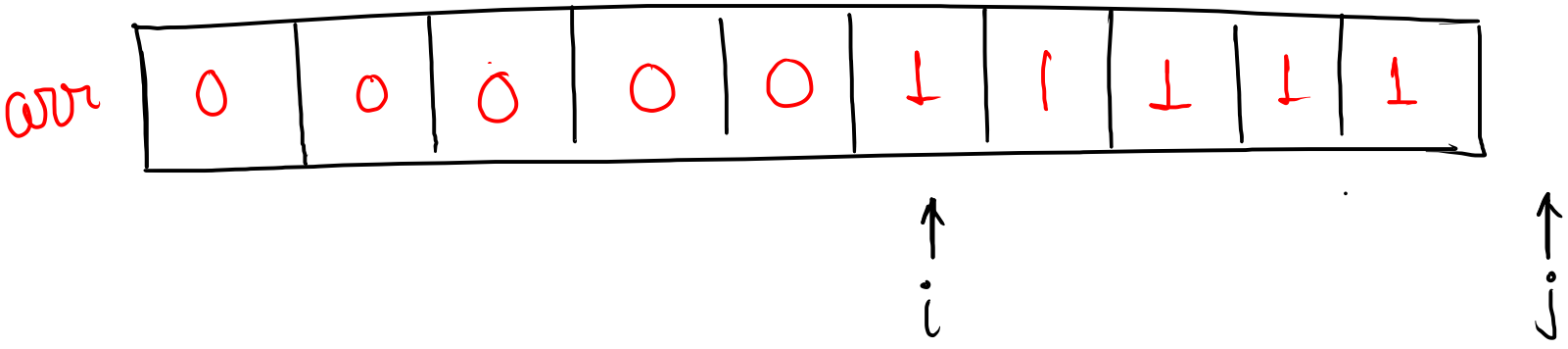


wall cond^n = 0

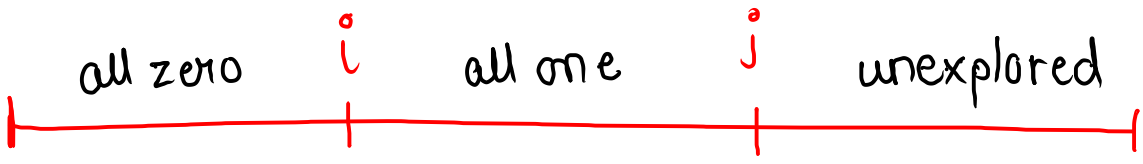
maxProd = max (curr * maxProd, curr * miniProd, curr) ;

minProd = min (curr * maxProd, curr * miniProd, curr) ;

⇒ 2 pointer



faith :-



in O(n) time

⇒ 3 sum

$$arr[i] + arr[j] + arr[k] = 0$$

$$arr[i] + arr[j] = -1 * arr[k]$$

$arr[i] + arr[j]$ ↑ target sum for each k value

$-1 * arr[k]$ target