minimum digits

n = 6

arr | 6 | 8 | 4 | 5 | 2 | 3 |

2 3 4 5 6 8

valid

043

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

Sample Output 0

```
604
```

Explanation 0

The minimum sum is formed by numbers 358 and 246

num1 = ~~0~~ ~~2~~ ~~24~~ 246

num2 = ~~0~~ ~~3~~ 35

num1 = 0 * 10 + 2 = 2 * 10 + 4 = 24 * 10 + 6 = 246

num2 = 0 * 10 + 3 = 3 * 10 + 5 = 35 * 10 + 8 = 358

Code

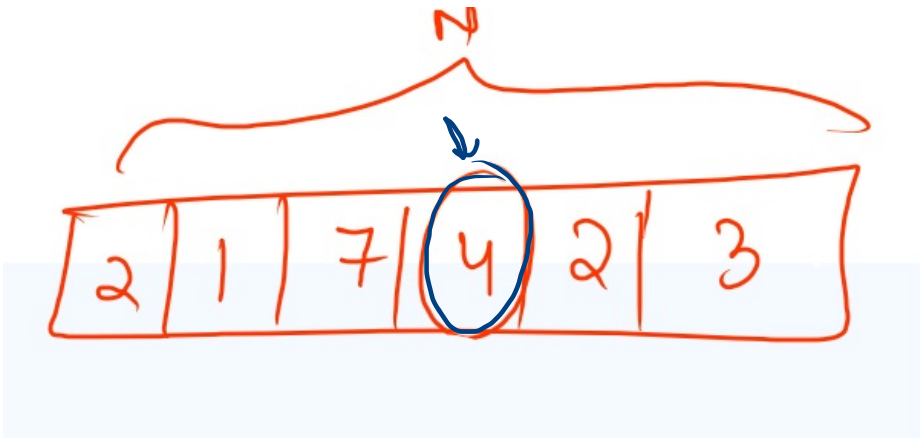
int range = 2^{31}

long range = 2^{63}

```java
public static long miniDigit(int[] arr) {
    PriorityQueue<Integer> pq = new PriorityQueue<>();
    for (int i : arr)
        pq.add(i);

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

maximum diamonds



☆☆☆☆☆

More

time = K

↳ pick only half of the diamonds from any of the bag (it will take 1 min)

ex :- K = 3 min 1 min = 2

Sample Input 0

```
5 3
2 1 7 4 2
```

Sample Output 0

```
14
```

Explanation 0

The state of bags is: [2 1 7 4 2]
You take all diamonds from Third bag (7).
The state of bags becomes: [2 1 3 4 2]
Take all diamonds from Fourth bag (4).
The state of bags becomes: [2 1 3 2 2]
Take all diamonds from Third bag (3).
The state of bags becomes: [2 1 1 2 2]
Hence, number of Diamonds = 7+4+3 = 14.

n = 5, k = 3

k = 1, diamond = 7

k = 2, diamond = 4

k = 3, diamond = 3

ans = 7 + 4 + 3
= 14

Code

```java
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();
    int k = scn.nextInt();
    int[] arr = new int[n];
    for (int i = 0; i < n; i++) arr[i] = scn.nextInt();

    System.out.println(maxDiamond(arr, k));
}

public static int maxDiamond(int[] arr, int k) {
    // maxPQ
    PriorityQueue<Integer> pq = new PriorityQueue<Integer>(Collections.reverseOrder());
    for (int i : arr) pq.add(i);

    int ans = 0;
    while (k-- > 0) {
        int temp = pq.poll();
        ans += temp;
        pq.add(temp / 2);
    }
    return ans;
}
```

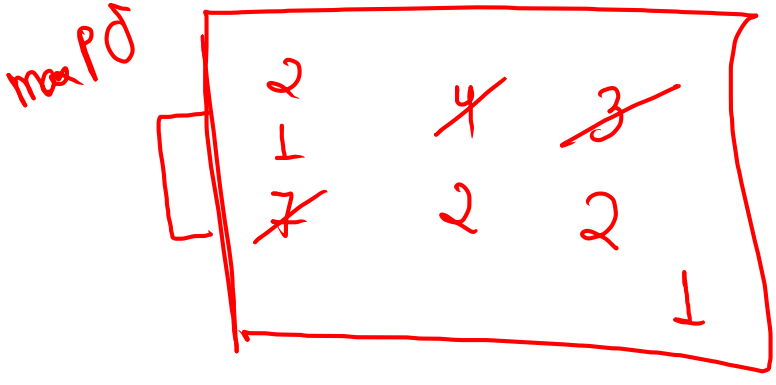
k = 3

3 times

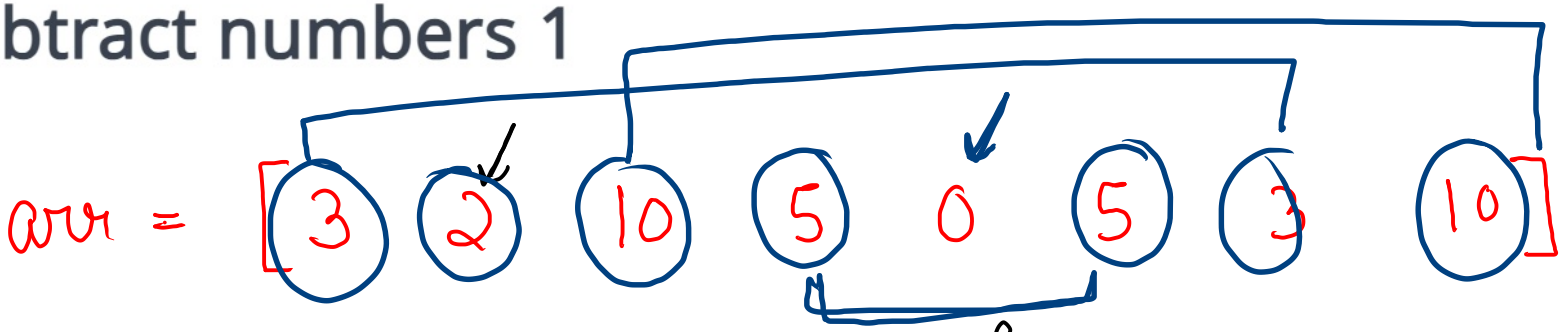
```
5 3
2 1 7 4 2
```

ans = 7+4+3

= 14



subtract numbers 1



choose a no. and subtract it from all

↳ 2 = [1 0 8 3 0 3 1 8]

↳ 1 = [0 0 7 2 0 2 0 7]

↳ 2 = [0 0 5 0 0 0 0 5]

↳ 5 = [0 0 0 0 0 0 0 0]

4 operations.

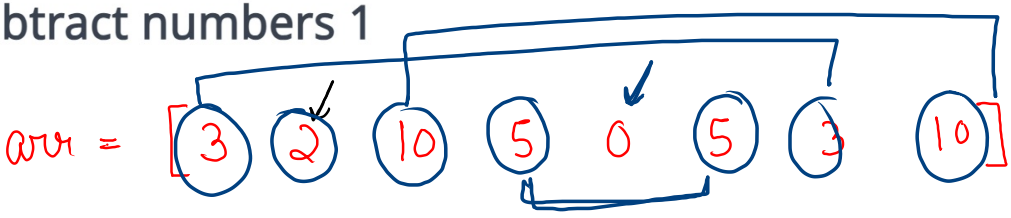
ex

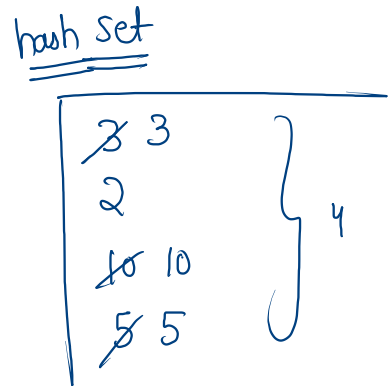
arr = [2 2 2 2 2 4 4]

↳ 2 [0 0 0 0 0 2 2]

↳ 2 [0 0 0 0 0 0 0]

subtract numbers 1






```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(subNumber(arr));
}

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

Find the Running Median

odd [1 2 3 4 5], med = 3.0

even [1 2 3 4 5 6], med = 3.5

Note:- imp

if arr.size is odd

median = middle element

if arr.size is even

median = (sum of 2 middle elements)/2

example

```
a[] size n = 6
a = [12, 4, 5, 3, 8, 7]
```

median

arr = [12] → 12.0

arr = [4, 12] → 8.0 ⇒ (12+4)/2

arr = [4, 5, 12] → 5.0

arr = [3, 4, 5, 12] → 4.5 ⇒ (4+5)/2

arr = [3, 4, 5, 8, 12] → 5.0

arr = [3, 4, 5, 7, 8, 12] → 6.0 ⇒ (5+7)/2

→ Brute force approach

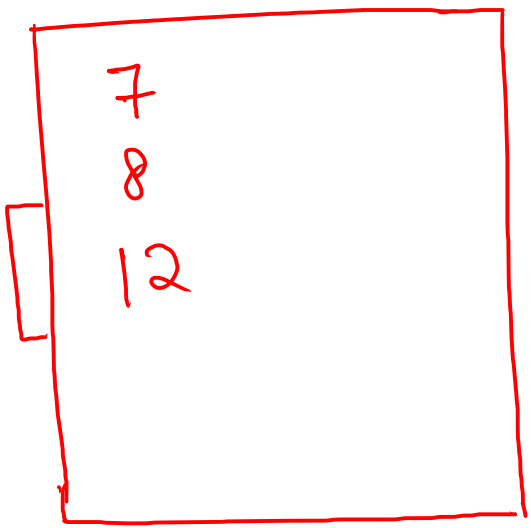
↳ Create AL
↳ add elements one by one
  ↳ sort the AL
  ↳ if even size = (sum of 2 mid. ele)/2
  ↳ else odd size = middle element

Optimised approch

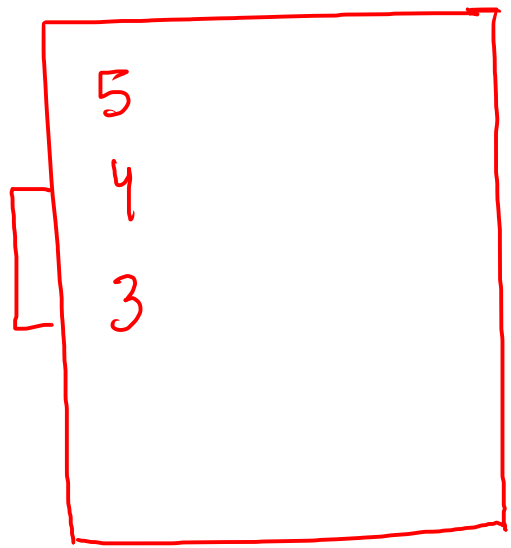
```
a[] size n = 6
a = [12, 4, 5, 3, 8, 7]
```

faith :-
put smaller element in maxPQ
and larger element in minPQ

minPQ



maxPQ



ans

⇒ 12.0
⇒ 8.0
⇒ 5.0
⇒ 4.5
⇒ 5.0
⇒ 6.0

```
static PriorityQueue<Integer> minPQ = new PriorityQueue<>();
static PriorityQueue<Integer> maxPQ = new PriorityQueue<>(Collections.reverseOrder());

public static void add(int x) {
    if ( maxPQ.size() > 0 && maxPQ.peek() <= x ) {
        minPQ.add(x);
    } else {
        maxPQ.add(x);
    }

    // balancing
    if ( Math.abs( maxPQ.size() - minPQ.size() ) > 1 ) {
        if ( maxPQ.size() > minPQ.size() ) {
            int num = maxPQ.poll();
            minPQ.add( num );
        } else {
            int num = minPQ.poll();
            maxPQ.add( num );
        }
    }
}
```

add element

find med

```
public static double median() {
    int size = minPQ.size() + maxPQ.size();
    double res;
    if (size % 2 == 0) {

        res = 0;
        if (minPQ.size() > 0) res += minPQ.peek();
        if (maxPQ.size() > 0) res += maxPQ.peek();
        res /= 2;

    } else {
        if (minPQ.size() > maxPQ.size()) {
            return minPQ.peek();
        } else {
            return maxPQ.peek();
        }
    }

    return res;
}
```

main

```
public static List<Double> runningMedian(List<Integer> arr) {
    List<Double> ans = new ArrayList<Double>();

    for (int i = 0; i < arr.size(); i++) {
        int num = arr.get(i);
        add( num );
        ans.add(median());
    }

    return ans;
}
```