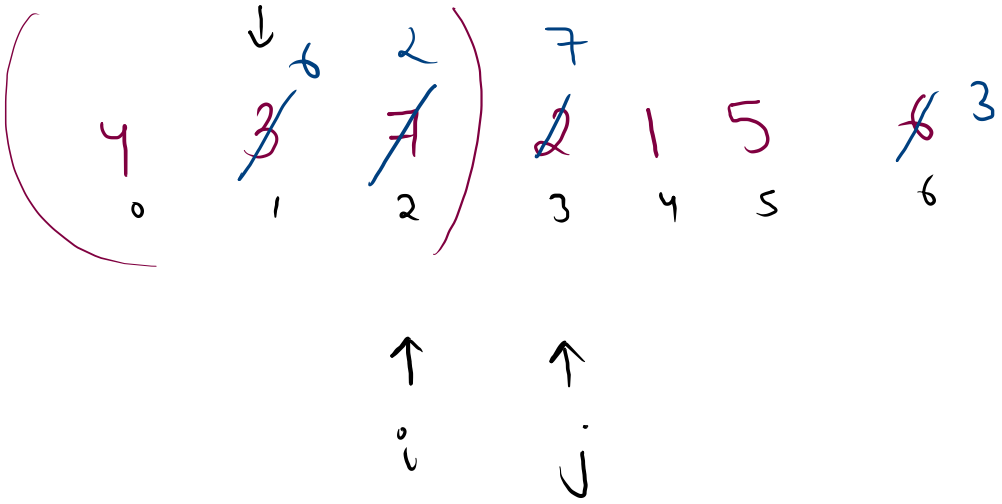
Sort by parity.

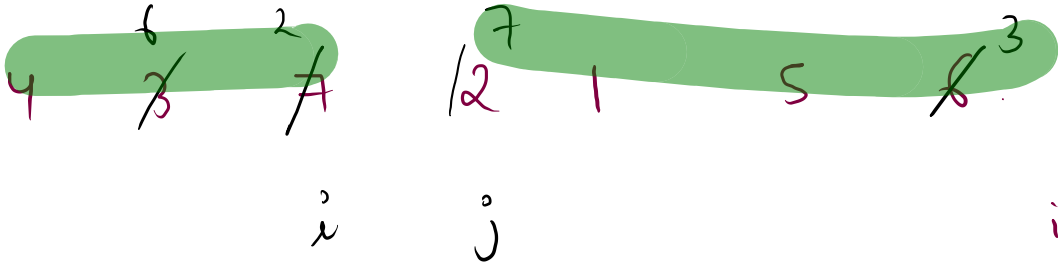
even .... odd ...
sorted sorted

↳ 3 1 4 2

(4 2) (3 1)

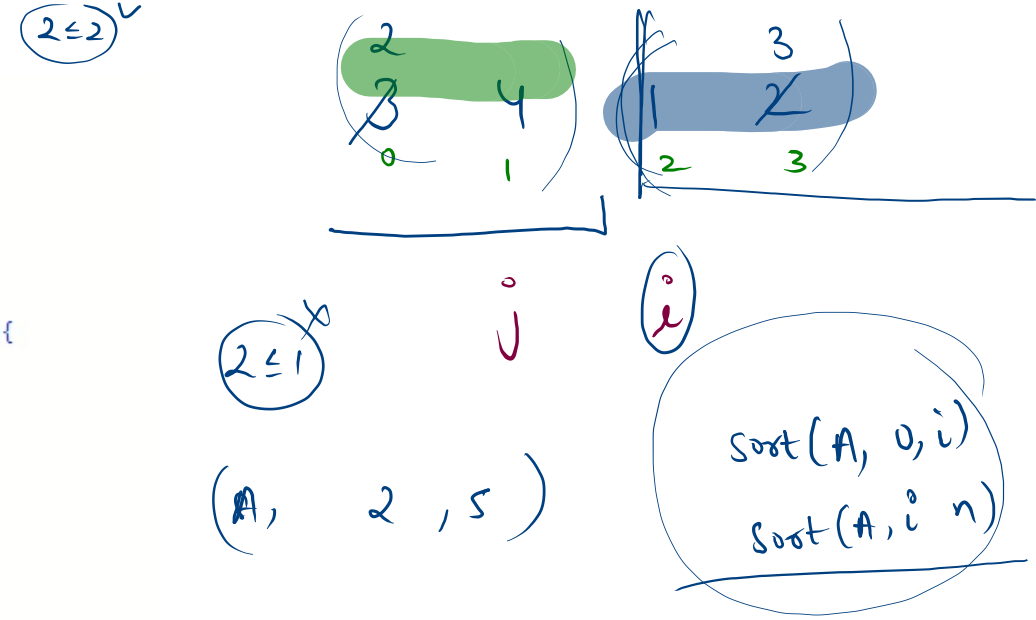
2 4 1 3 → ans.


4 3 7 2 1 5 6 3
0 1 2 3 4 5 6
i
j
i i++
j j--
swap

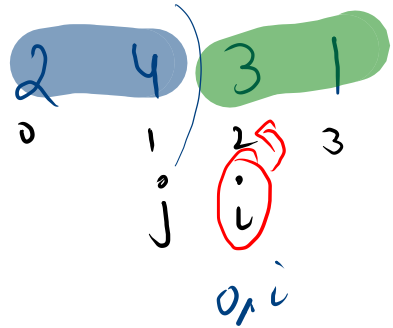



```
if (A[i] % 2 == 0)
{   i++
}
else if (A[j] % 2 != 0)
{   j--
}
else {  Swap
}
```

```
11      int i = 0;
12      int j = n-1;
13
14      while(i <= j){
15          if(A[i] % 2 == 0){
16              i++;
17          }
18          else if(A[j] % 2 != 0){
19              j--;
20          }
21          else{
22              int tmp = A[i];
23              A[i] = A[j];
24              A[j] = tmp;
25              i++;
26              j--;
27          }
28      }
29
```



```java
import java.util.*;
public class Main {
    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();
        Integer [] A = new Integer[n];
        for(int i = 0; i < n; i++){
            A[i] = scn.nextInt();
        }
        int i = 0;
        int j = n-1;
        while(i <= j){
            if(A[i] % 2 == 0){
                i++;
            }
            else if(A[j] % 2 != 0){
                j--;
            }
            else{
                int tmp = A[i];
                A[i] = A[j];
                A[j] = tmp;
                i++;
                j--;
            }
        }
        Arrays.sort(A, 0, i);
        Arrays.sort(A, i, n);

        for(i = 0; i < n; i++){
            System.out.print(A[i] + " ");
        }
    }
}
```



(0, i)

[0, i-1]

[i, n-1]

Sort (0, 2)

Sort (2, 4)

2 4 (3 1)

2 1 (3 3 4 6)

sort (A, 2, 6)

↳ Nawab?

Sort an array in wave form

Sort an array in wave form

Problem Statement

Let me introduce you to John, who loves to play with numbers. One day, he was given an unsorted array of integers and was challenged to transform it into a wave-like array. John found it interesting and decided to take up the challenge.

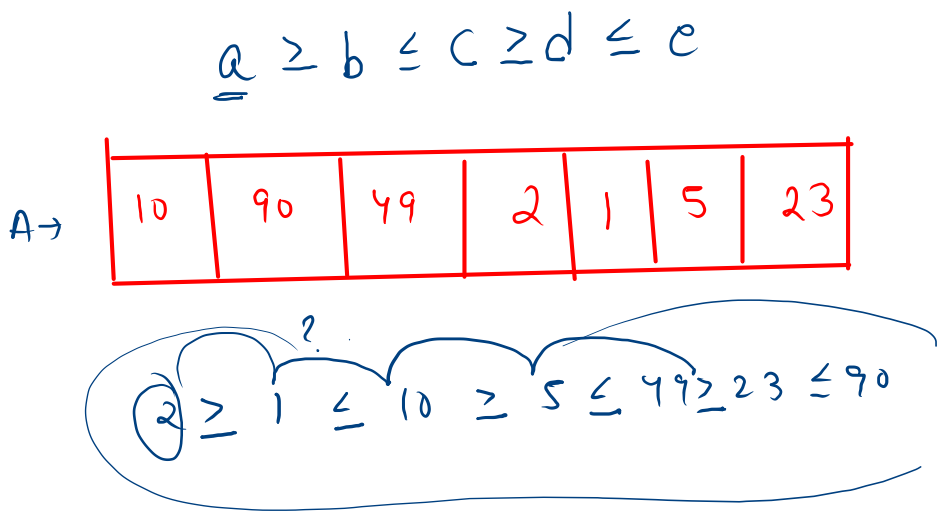
Help John by sorting the array into a wave like array. An array **arr[0..n-1]** is sorted in wave form if

arr[0] >= arr[1] <= arr[2] >= arr[3] <= arr[4] >= .....

```
7
10 90 49 2 1 5 23
```

Sample Output 0

```
2 1 10 5 49 23 90
```



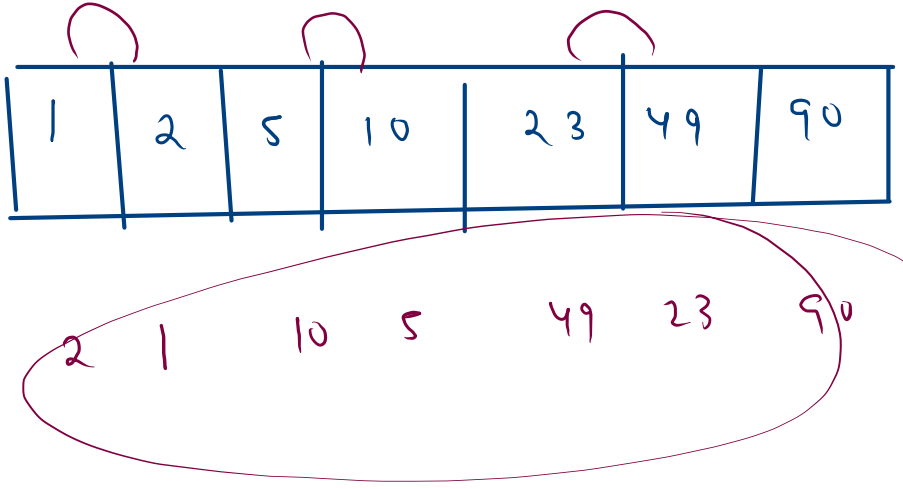
```
7
10 90 49 2 1 5 23
```

Sample Output 0

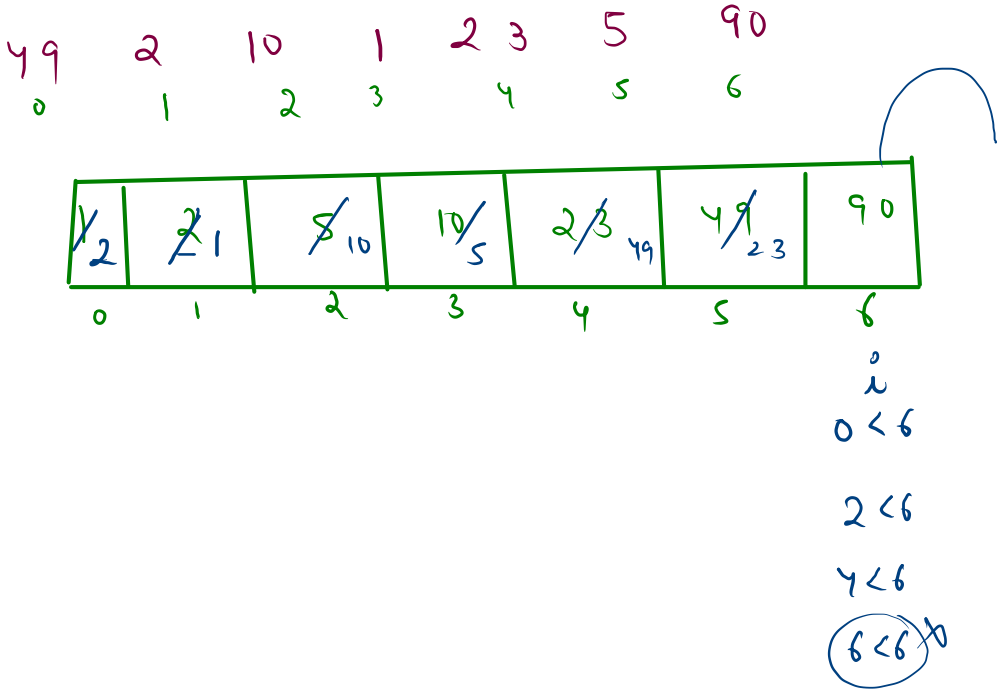
```
2 1 10 5 49 23 90
```

2 step.

1. sort (asc.)
2. swap alternate.



```java
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();
        Integer [] A = new Integer[n];
        for(int i = 0; i < n; i++){
            A[i] = scn.nextInt();
        }

        //step 1. sort
        Arrays.sort(A);

        //step 2. swap alternate
        for(int i = 0; i < n-1; i += 2){
            int tmp = A[i];
            A[i] = A[i+1];
            A[i+1] = tmp;
        }

        for(int i = 0; i < n; i++){
            System.out.print(A[i] + " ");
        }
    }
}

```



Minimum difference

Minimum difference

Problem Statement

You are given a 0-indexed integer array nums, where **nums[i]** represents the score of the **ith** student. You are also given an integer **k**.

Pick the scores of any **k** students from the array so that the difference between the **highest** and the **lowest** of the **k** scores is minimized.

Return the minimum possible difference.

n = 4

9 4 1 7

0 1 2 3

k = 2

=

9 4

9 1

9 7

4 1

4 7

1 7

h l

9 4 = 5

9 1 = 8

9 7 = 2

4 1 = 3

7 4 = 3

7 1 = 6

d

2

ans = 2

9 4 1 7

k = 2

h l

9 4 = 5

9 1 = 8

9 7 = 2

4 1 = 3

7 4 = 3

⇒ 1 = 6

9 4 4 1

9 1 4 7

9 7 1 7

↓ d = ~~0~~ ~~5~~ 2

k = 3

9 4 1 7

ABC

h	l
9	1 = 8
9	4 = 5
9	1 = 8
7	1 = 6

9 4 1

9 4 7

9 1 7

4 17

↓d = ~~9~~ ~~8~~ 5

k = 4

9 1 4 7

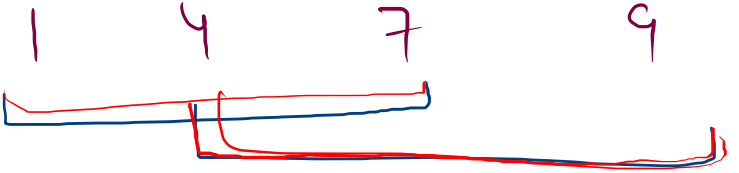
h l

9 1 = 8

9 1 4 7

↓ d = 8

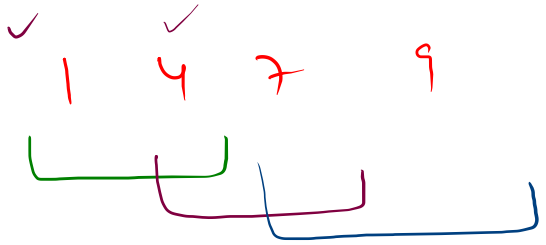
sort



h l

7 1 = 6

9 4 = 5



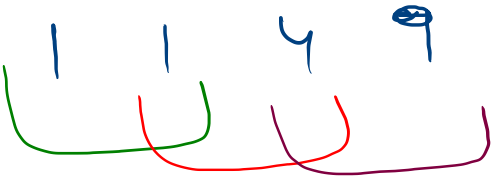
h l

g → 4 1

4 ≠ 4

r → 9 7

d = ~~9~~ ~~3~~ 2

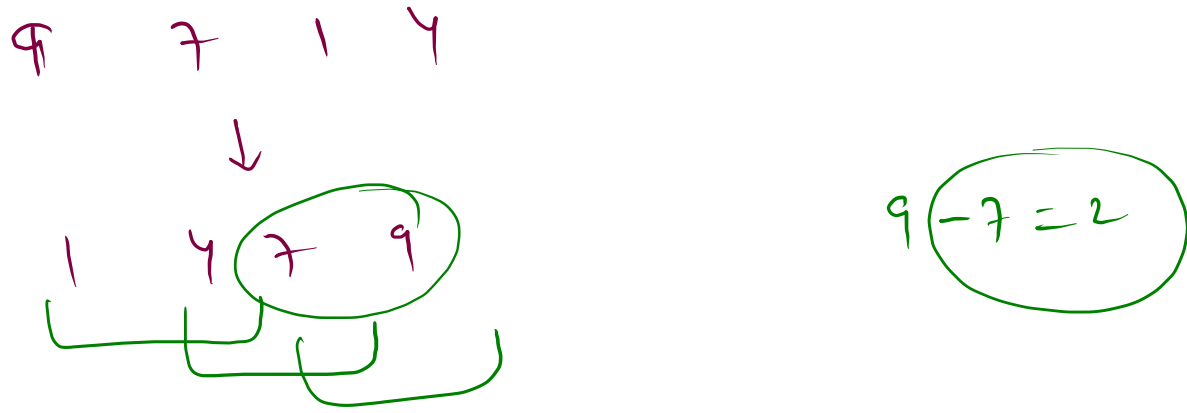


h l

g 1 1 0

4 1 = 3

9 4 = 5


9 7 1 4
1 4 7 9
9-7=2

9 7 1 4 ⟶ 1 4 7 9 $k=3$.

0 1 2 3

i

✓ ✓

$h = A[i+k-1]$ $l = A[i]$

n i k

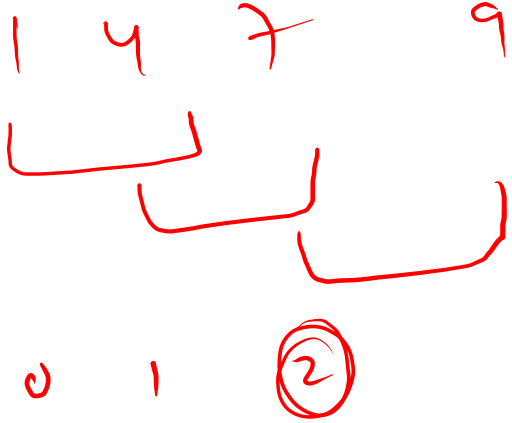
$1 + 3 - 1 = 3$

$i \le n-k$

$i \le 4-3$

$i \le 1$

$i < k-1$



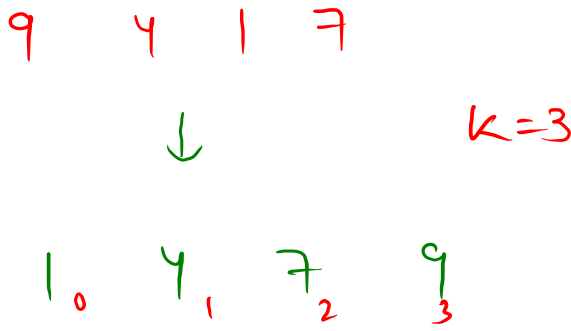
$k = 2$

$i \leq n - k$

```java
import java.util.*;
public class Main {
    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();
        Integer [] A = new Integer[n];
        for(int i = 0; i < n; i++){
            A[i] = scn.nextInt();
        }

        int k = scn.nextInt();
        int ans = Integer.MAX_VALUE;
        Arrays.sort(A);
        for(int i = 0; i <= n-k; i++){
            int h = A[i+k-1];
            int l = A[i];

            int d = h-l;
            ans = Math.min(ans, d);

        }
        System.out.println(ans);
    }
}
```

Console

Run Code



Problem Statement

Meet Laura, a data analyst who was given a task to identify **peak elements** from an array of numerical data. Laura was fascinated by the concept of peak elements and found them to be useful in many real-world scenarios, such as identifying the maximum temperature in a dataset.

Find the peak elements by comparing each element with its adjacent elements and find elements that satisfy the given condition.

arr[i] is a peak element only if **arr[i-1] < arr[i] > arr[i+1].**

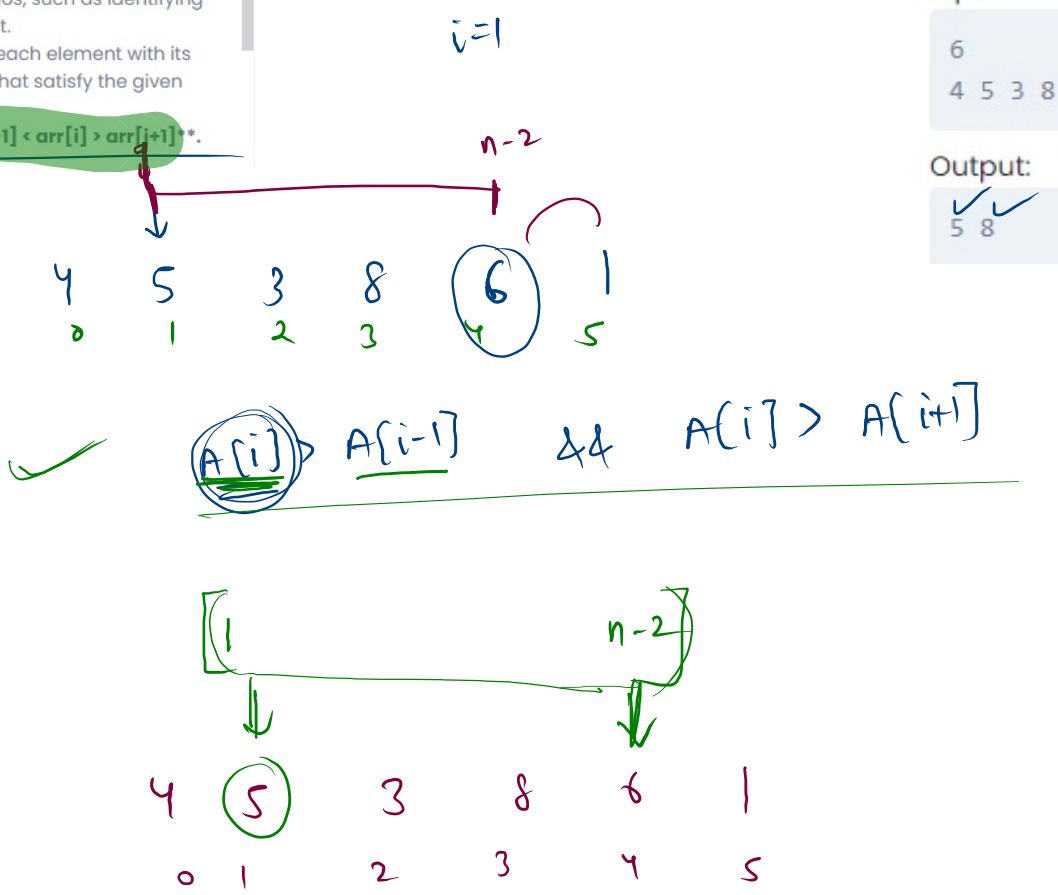
Test Case 1

Input:

```
6
4 5 3 8 6 1
```

Output:

```
5 8
```



```java
import java.util.*;
public class Main {
    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();
        Integer [] A = new Integer[n];
        for(int i = 0; i < n; i++){
            A[i] = scn.nextInt();
        }

        for(int i =1; i  <= n-2; i++){
            if(A[i] > A[i+1] && A[i] > A[i-1]){
                System.out.print(A[i] + " ");
            }
        }
    }
}

```

5 8