You are given a positive integer array skill of even length **N** where `skill[i]` denotes the skill of the **ith** player. Divide the players into **n/2** teams of size **2** such that the total skill of each team is equal.

The chemistry of a team is equal to the product of the skills

Return the sum of the chemistry of all the teams, or **return** teams such that the total skill of each team is equal.

Input Format

The first line contains **N**, i.e. the size of the array.

The second line contains **N** space-separated positive integ

Constraints

```
2 <= N <= 10^5

N is even.

1 <= skill[i] <= 1000
```

Output Format

Return the sum of the chemistry of all the teams, or **return** teams such that the total skill of each team is equal.

Sample Input 0

```
6
3 2 5 1 3 4
```

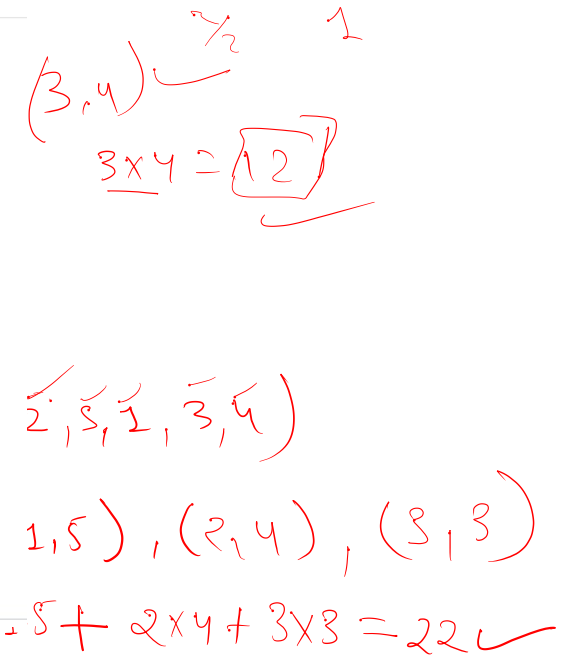
Sample Output 0

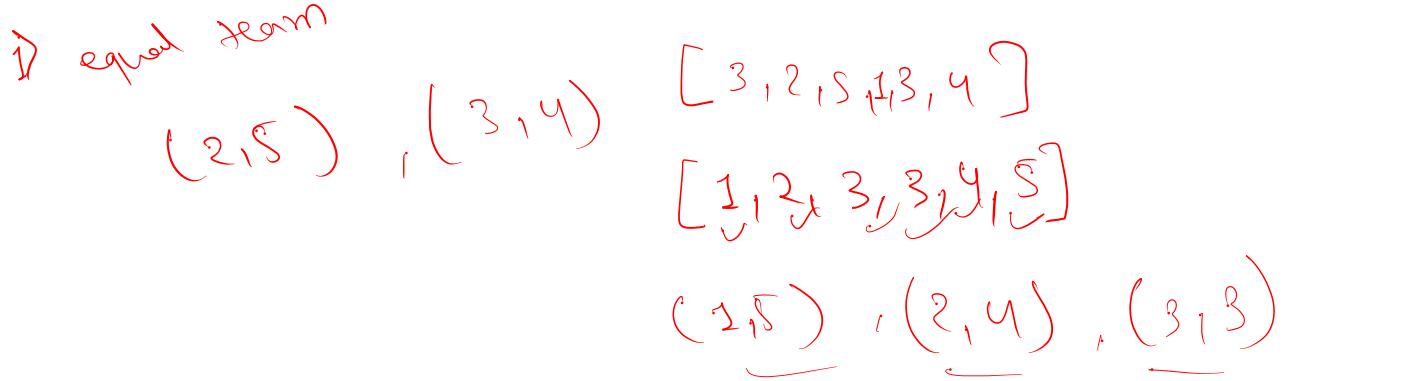
```
22
```

Submitted Code

Language: Java 15

```java
import java.io.*;
import java.util.*;

public class Solution {

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

        Arrays.sort(arr);

        int i=0,j=n-1,sum=0,ans=0;
        sum= arr[i]+arr[j];
        while(i<j){
            if(arr[i]+arr[j] !=sum){
                System.out.print(-1);
                return;
            }
            ans+=arr[i]*arr[j];
            i++;
            j--;

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





Explanation 0

Divide the players into the following teams: (1, 5), (2, 4), (3, 3), where each team has a total skill of 6. The sum of the chemistry of all the teams is: 1 * 5 + 2 * 4 + 3 * 3 = 5 + 8 + 9 = 22.

Sample Input 1

```
2
3 4
```

Sample Output 1

```
12
```

Explanation 1

The two players form a team with a total skill of 7. The chemistry of the team is 3 * 4 = 12.

Given a **0-indexed** integer array `nums`, find the **leftmost middleIndex** (i.e., the smallest amongst all the possible ones).

A middleIndex is an index where `nums[0] + nums[1] + ... + nums[middleIndex-1] == nums[middleIndex+1] + nums[middleIndex+2] + ... + nums[nums.length-1]`.

If `middleIndex == 0`, the left side sum is considered to be **0**. Similarly, if `middleIndex == nums.length - 1`, the right side sum is considered to be **0**.

Return the leftmost middleIndex that satisfies the condition, or **-1** if there is no such index.

Input Format

The first line contains **N**, i.e. the size of the array.

The second line contains **N** space-separated positive integers `nums[i]` denoting elements of the array.

Constraints

```
1 <= N <= 100

-1000 <= nums[i] <= 1000
```

Output Format

Return the leftmost index which satisfies the condition.

Sample Input 0

```
5
2 3 -1 8 4
```

Sample Output 0

```
3
```

Explanation 0

The sum of the numbers before index 3 is: 2 + 3 + -1 = 4 The sum of the numbers after index 3 is: 4 = 4

Submitted Code

Language: Java 15

```
import java.io.*;
import java.util.*;

public class Solution {

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

        int right=0,left=0;
        for(int i=0;i<n;i++)right+=arr[i];

        for(int i=0;i<n;i++){
            right-=arr[i];
            if(right== left){
                System.out.print(i);
                return;
            }
            left+=arr[i];
        }
        System.out.print(-1);
    }
}
```

Make a prefix array of size **N** such that at the **kth** index of the prefix array store the smallest element from the **left** till the **kth** index of the given array.

Input Format

First line contains integer **N** representing soze of array.

Second line contains **N** integers as array elements.

Constraints

```
1 <= N <= 10^4

-10^4 <= A[i] <= 10^4
```

Output Format

Return the prefix array.

Sample Input 0

```
7
1
88
3
2
16
10
9
```

Sample Output 0

```
1
1
1
1
1
1
1
```

Submitted Code

Language: Java 15

```
1  import java.io.*;
2  import java.util.*;
3
4  public class Solution {
5
6      public static void main(String[] args) {
7          Scanner sc = new Scanner(System.in);
8          int n = sc.nextInt();
9          int arr[] = new int[n];
10         for(int i=0;i<n;i++)arr[i]=sc.nextInt();
11
12         for(int i=1;i<n;i++)arr[i]=Math.min(arr[i-1],arr[i]);
13
14         for(int i=0;i<n;i++)System.out.println(arr[i]);
15     }
16 }
```

There is a biker going on a road trip. The road trip consists of **n + 1** points at different altitudes. The biker starts his trip on point **0** with altitude equal **0**.

You are given an integer array `gain` of length **N** where `gain[i]` is the net gain in altitude between points **i** and **i + 1** for all **(0 <= i < n)**. Return the highest altitude of a point.

Input Format

First line of input contains integer **N** representing the size of array.

Second line of input contains **N** integers representing elements of array.

Constraints

```
1 <= N <= 100

-100 <= gain[i] <= 100
```

Output Format

Return the highest altitude.

Sample Input 0

```
5
-5 1 5 0 -7
```

Sample Output 0

```
1
```

Explanation 0

The altitudes are [0,-5,-4,1,1,-6]. The highest is 1.

Submitted Code

Language: Java 15

```
1  import java.io.*;
2  import java.util.*;
3
4  public class Solution {
5
6      public static void main(String[] args) {
7         Scanner sc = new Scanner(System.in);
8          int n = sc.nextInt();
9          int arr[] = new int[n];
10         for(int i=0;i<n;i++)arr[i]=sc.nextInt();
11
12         int diff[]=new int[n+1];
13         diff[0]=0;
14         int idx=1;
15
16         for(int i=0;i<n;i++){
17             diff[idx]=diff[i]+arr[i];
18             idx++;
19         }
20         int max= Integer.MIN_VALUE;
21         for(int i=0;i<diff.length;i++)max=Math.max(diff[i],max);
22
23         System.out.print(max);
24     }
25 }
```