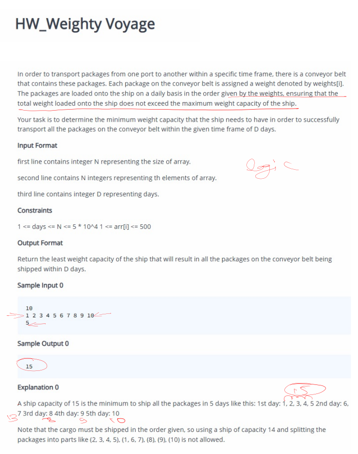


HW_Weighty Voyage

In order to transport packages from one port to another within a specific time frame, there is a conveyor belt that contains these packages. Each package on the conveyor belt is assigned a weight denoted by weights[i]. The packages are loaded onto the ship on a daily basis in the order given by the weights, ensuring that the total weight loaded onto the ship does not exceed the maximum weight capacity of the ship.

Your task is to determine the minimum weight capacity that the ship needs to have in order to successfully transport all the packages on the conveyor belt within the given time frame of D days.

Input Format

first line contains integer N representing the size of array.

second line contains N integers representing th elements of array.

third line contains integer D representing days.

logic

Constraints

1 <= days <= N <= 5 * 10^4 1 <= arr[i] <= 500

Output Format

Return the least weight capacity of the ship that will result in all the packages on the conveyor belt being shipped within D days.

Sample Input 0

```
10
1 2 3 4 5 6 7 8 9 10
5
```

Sample Output 0

```
15
```

Explanation 0

15

A ship capacity of 15 is the minimum to ship all the packages in 5 days like this: 1st day: 1, 2, 3, 4, 5 2nd day: 6, 7 3rd day: 8 4th day: 9 5th day: 10

13 8 9 10

Note that the cargo must be shipped in the order given, so using a ship of capacity 14 and splitting the packages into parts like (2, 3, 4, 5), (1, 6, 7), (8), (9), (10) is not allowed.

1 2 3 4 5 6 7 8,9,10

e totalW = 55 55/10 = 5.5

s MaxW = 10
mid = 5
while (~~st~~ <= m) {

~~mid~~ = MaxW + (t - MaxW)/2

day = 1, SumW = 0
for (int i = 0; i < n; i++) {

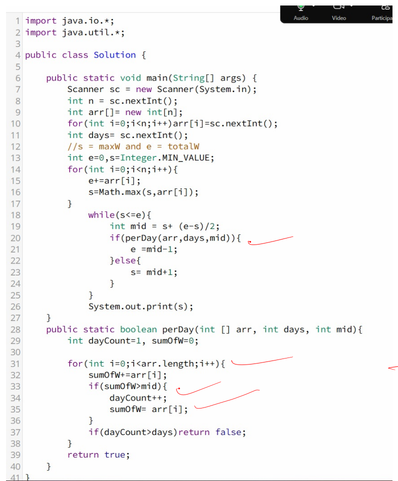
SumW += ~~arr~~[i];
if (sumW > mid) {
day ++;
SumW = arr[i];
}
if (day > @days)
return false;

}

}
return true;

if (true) {
MaxW = mi~~d~~ - 1

else {
totalW = mid + 1;
}



```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();
        int days= sc.nextInt();
        //s = maxW and e = totalW
        int e=0,s=Integer.MIN_VALUE;
        for(int i=0;i<n;i++){
            e+=arr[i];
            s=Math.max(s,arr[i]);
        }
            while(s<=e){
                int mid = s+ (e-s)/2;
                if(perDay(arr,days,mid)){
                    e =mid-1;
                }else{
                    s= mid+1;
                }
            }
            System.out.print(s);
    }
    public static boolean perDay(int [] arr, int days, int mid){
        int dayCount=1, sumOfW=0;

        for(int i=0;i<arr.length;i++){
            sumOfW+=arr[i];
            if(sumOfW>mid){
                dayCount++;
                sumOfW= arr[i];
            }
            if(dayCount>days)return false;
        }
        return true;
    }
}
```

total
M

Su
~~day~~ > mid
day ++

HW_Minimal Maximum Sum

You are provided with an integer array nums and another integer k. Your task is to divide the array nums into k non-empty subarrays such that the maximum sum of any subarray is as small as possible.

Return the minimized largest sum of the split.

NOTE:-A subarray is a contiguous part of the array.

the painter

weight voyage

Input Format

first line of input contains integer N representing the size of array.

second line of input contains N integers representing the elements of array.

third line of input contains integer k.

Constraints

1 <= N <= 1000

0 <= nums[i] <= 10^6

1 <= k <= min(50, N)

Output Format

Return the minimized largest sum of the split.

Sample Input 0

```
5
7 2 5 10 8
2
```

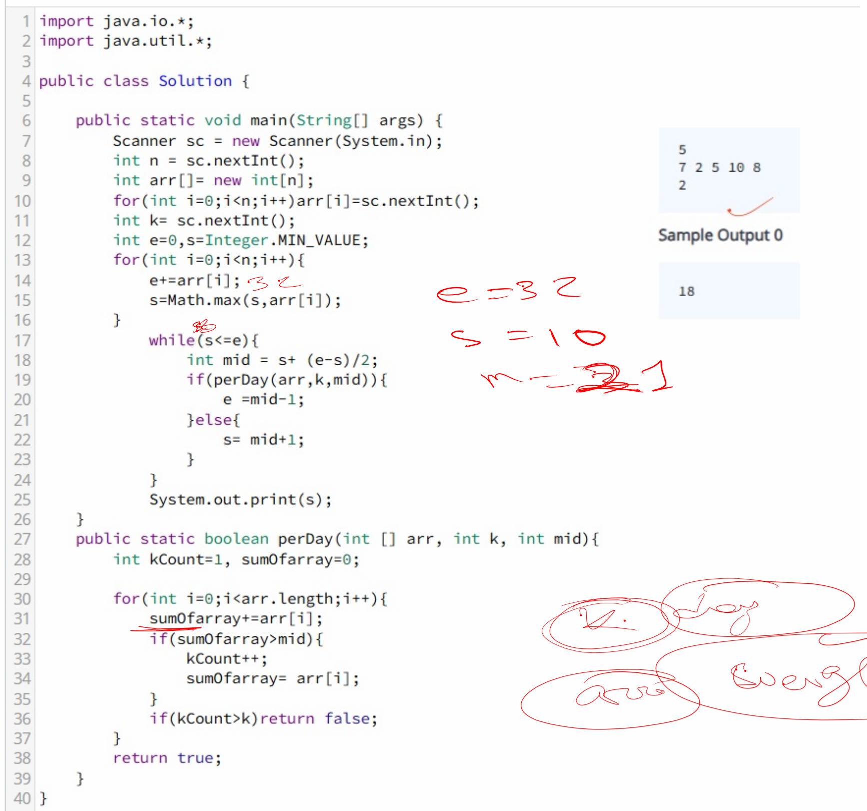
2k

Sample Output 0

```
18
```

Explanation 0

There are four ways to split nums into two subarrays. The best way is to split it into [7,2,5] and [10,8], where the largest sum among the two subarrays is only 18.



```
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 k= sc.nextInt();
        int e=0,s=Integer.MIN_VALUE;
        for(int i=0;i<n;i++){
            e+=arr[i];
            s=Math.max(s,arr[i]);
        }
            while(s<=e){
                int mid = s+ (e-s)/2;
                if(perDay(arr,k,mid)){
                    e =mid-1;
                }else{
                    s= mid+1;
                }
            }
            System.out.print(s);
    }
    public static boolean perDay(int [] arr, int k, int mid){
        int kCount=1, sumOfarray=0;

        for(int i=0;i<arr.length;i++){
            sumOfarray+=arr[i];
            if(sumOfarray>mid){
                kCount++;
                sumOfarray= arr[i];
            }
            if(kCount>k)return false;
        }
        return true;
    }
}
```

32

e=32

s=10

m=~~2~~21

```
5
7 2 5 10 8
2
```

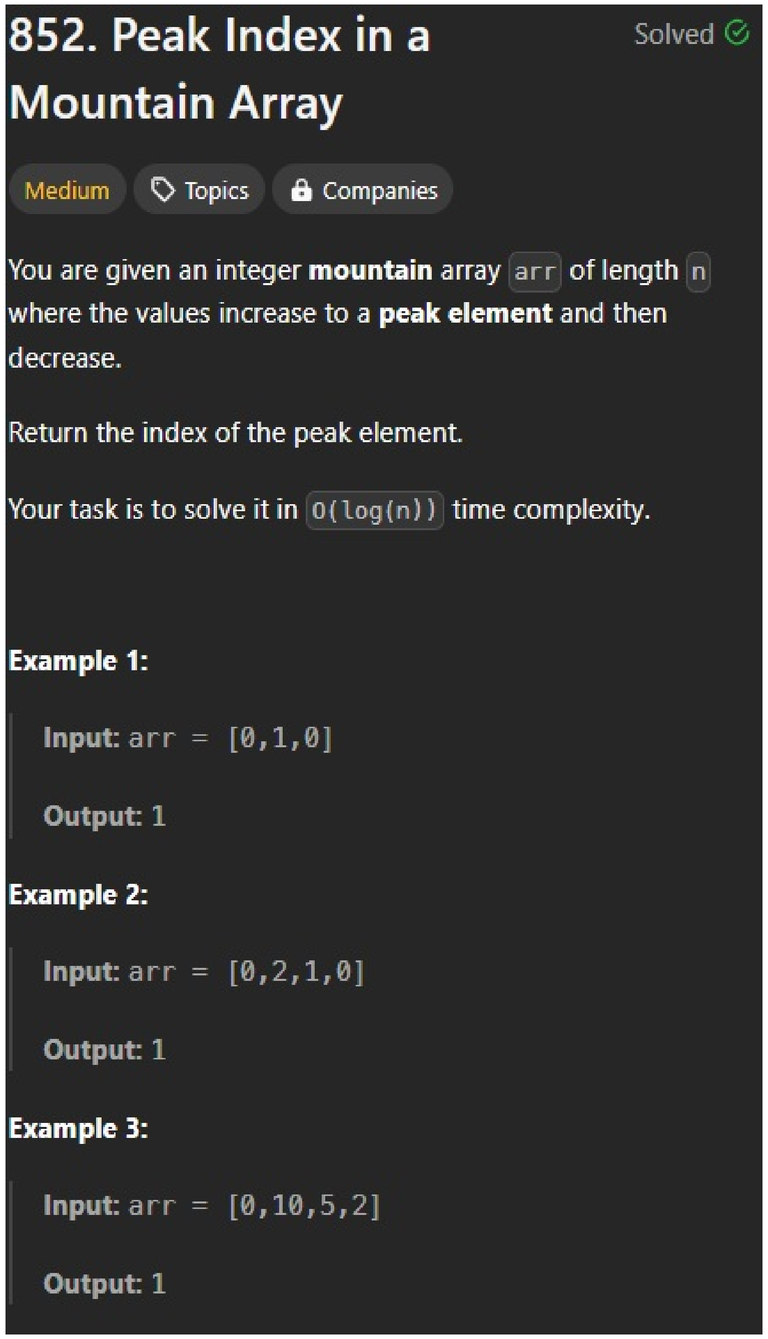
Sample Output 0

```
18
```

k. day

arr weight

Arr ayls [illegible]



852. Peak Index in a Mountain Array

Solved

Medium | Topics | Companies

You are given an integer **mountain** array `arr` of length `n` where the values increase to a **peak element** and then decrease.

Return the index of the peak element.

Your task is to solve it in `O(log(n))` time complexity.

Example 1:

Input: `arr = [0,1,0]`

Output: 1

Example 2:

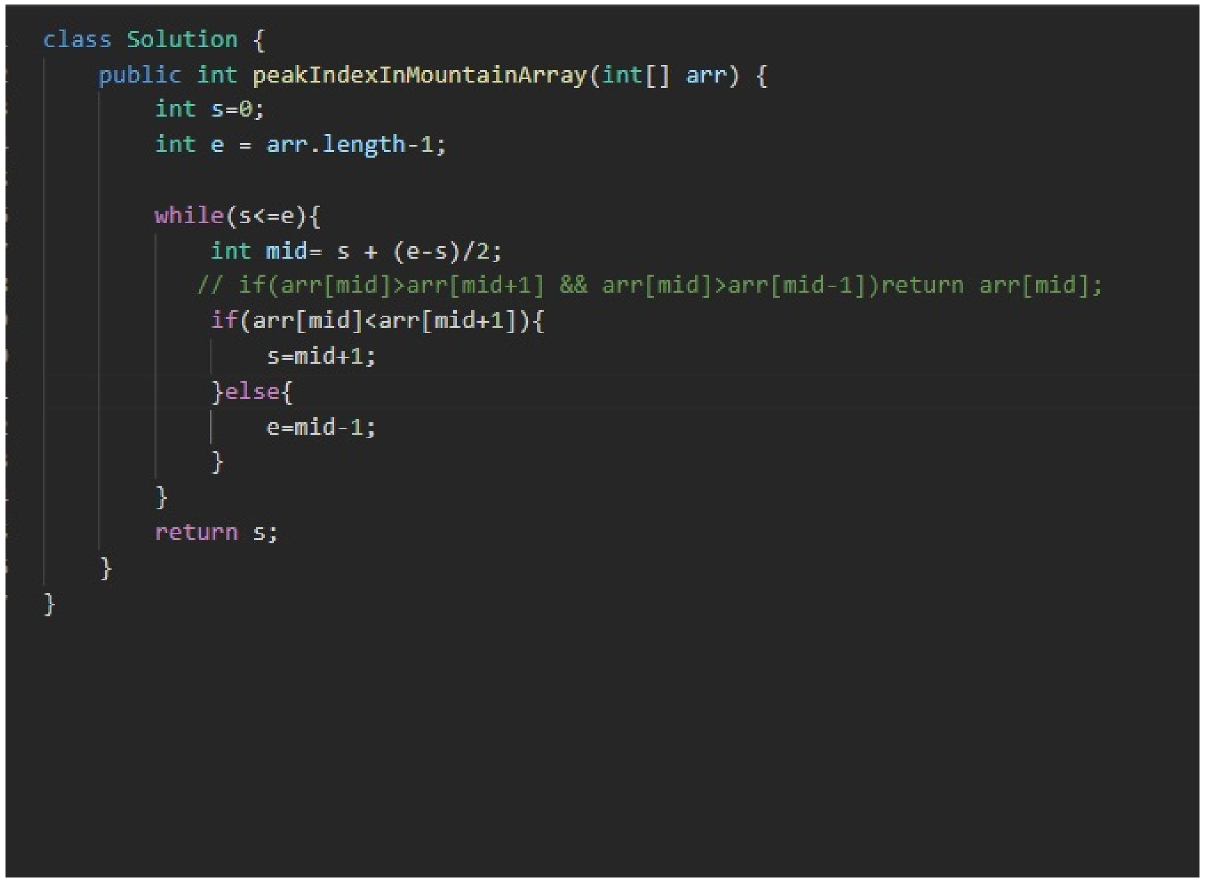
Input: `arr = [0,2,1,0]`

Output: 1

Example 3:

Input: `arr = [0,10,5,2]`

Output: 1



```java
class Solution {
    public int peakIndexInMountainArray(int[] arr) {
        int s=0;
        int e = arr.length-1;

        while(s<=e){
            int mid= s + (e-s)/2;
           // if(arr[mid]>arr[mid+1] && arr[mid]>arr[mid-1])return arr[mid];
            if(arr[mid]<arr[mid+1]){
                s=mid+1;
            }else{
                e=mid-1;
            }
        }
        return s;
    }
}
```