5TH JAN 2023

FIRST :-

Minimum number of jumps :: Medium

Given an array of **N** integers **arr[]** where each element represents the max length of the jump that can be made forward from that element. Find the minimum number of jumps to reach the end of the array (starting from the first element). If an element is **0**, then you cannot move through that element.

Note: Return -1 if you can't reach the end of the array.

Example 1:

```
Input:
N = 11
arr[] = {1, 3, 5, 8, 9, 2, 6, 7, 6, 8, 9}
Output: 3
Explanation:
First jump from 1st element to 2nd
element with value 3. Now, from here
we jump to 5th element with value 9,
and from here we will jump to the last.
```

Example 2:

```
Input :
N = 6
arr = {1, 4, 3, 2, 6, 7}
Output: 2
Explanation:
First we jump from the 1st to 2nd element
and then jump to the last element.
```

Your task:
You don't need to read input or print anything. Your task is to complete function **minJumps()** which takes the array **arr** and it's size **N** as input parameters and returns the minimum number of jumps. If not possible return -1.

Expected Time Complexity: O(N)
Expected Space Complexity: O(1)

Constraints:
$1 \leq N \leq 10^7$
$0 \leq arr_i \leq 10^7$

CODE SECTION:-

```
class Solution
{
public:
    int maxjump(int arr[], int i, int n)
    {
        int x = arr[i];
        int max = INT32_MIN;
        int maxi = -1;
        while (x != 0)
        {
            if (i + 1 == n - 1)
            {
                return n;
            }
            else if (arr[i + 1] > max)
            {
                max = arr[i + 1];
                maxi = i + 1;
            }
            i++;
            x--;
        }
        return maxi;
    }
```

```
    int minJumps(int arr[], int n)
    {
        // Your code here
        // brute force method
        //     int i=0;
        //     int count=0;
        //     int *maxm=0;
        //     if(arr[0]==0){
        //         return -1;
        //     }
        //   while(i<n){
        //       i=maxjump(arr,i,n);
        //       count++;
        //   }
        //     return count;

      //optimize method
        if (arr[0] == 0 && n > 1)
            return -1;
        if (n == 1)
            return 0;
        int far = 0;
        int curr = 0;
        int jump = 0;
        for (int i = 0; i < n - 1; i++)
        {
            far = max(far, i + arr[i]);
            if (i == curr)
            {
                jump++;
                curr = far;
            }
        }
        if (curr < n - 1)
            return -1;
        return jump;
    }
};
```

SECOND :-

Count Reverse Pairs : : Hard

You are given an array of **N** integers **arr**, find the count of reverse pairs.

A pair of indices (i, j) is said to be a **reverse pair** if both the following conditions are met:

- `0 <= i < j < N`
- `arr[i] > 2 * arr[j]`

Example 1:

```
Input:
N = 6
arr = [3, 2, 4, 5, 1, 20]
Output:
3
Explanation:
The Reverse pairs are
(i)   (0, 4), arr[0] = 3, arr[4] = 1, 3 > 2(1)
(ii)  (2, 4), arr[2] = 4, arr[4] = 1, 4 > 2(1)
(iii)(3, 4), arr[3] = 5, arr[4] = 1, 5 > 2(1)
```

Example 2:

```
Input:
N = 5
arr= [2, 4, 3, 5, 1]
Output:
3
Explanation:
(i)    (1, 4), arr[1] = 4, arr[4] = 1, 4 > 2 * 1
(ii)   (2, 4), arr[2] = 3, arr[4] = 1, 3 > 2 * 1
(iii)  (3, 4), arr[3] = 5, arr[4] = 1, 5 > 2 * 1
```

Your Task:
Complete the function **countRevPairs**(), which takes integer a list of N integers as input and returns the count of Reverse Pairs.

Expected Time Complexity: O(N logN)
Expected Auxiliary Space: O(N)

Constraints:

1 <= N <= 50000

1 <= arr[i] <= 10^9

CODE SECTION:-

```
class Solution
{

private:
    int count;

    void checkCount(vector<int> &nums, int start, int mid, int end)
    {

        // two pointers;

        int l = start, r = mid + 1;

        while (l <= mid && r <= end)
        {

            if ((long)nums[l] > (long)2 * nums[r])
            {

                count += (mid - l + 1);

                r++;
            }
            else
            {

                l++;
            }
        }

        // worst case might be nlog(n)

        sort(nums.begin() + start, nums.begin() + end + 1);
        return;
        // every step sort
    }

    void mergeSort(vector<int> &nums, int start, int end)
    {

        if (start == end)
            return;

        int mid = (start + end) / 2;

        mergeSort(nums, start, mid);
```

```
        mergeSort(nums, mid + 1, end);

        checkCount(nums, start, mid, end);

        return;
    }

public:
    int countRevPairs(int n, vector<int> nums)
    {

        // Code here

        if (!nums.size())
            return 0;

        count = 0;

        mergeSort(nums, 0, nums.size() - 1);

        return count;
    }
};
```

-: Done for the today :-