package com.company;

//Problem 3 Assignment 2

//Time complexity – O(n^2) Space complexity – O(n)

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.Scanner;

public class Main {

public static void main(String[] args) throws IOException {

// write your code here

Scanner scan = new Scanner(System.in);

int n = scan.nextInt();

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

String[] s = new String[n];

int count = 0;

int[] map = new int[n]; //map for array map to find coverage of sorted data

Data[] data = new Data[n];

Data[] sorted = new Data[n];

int[] arr = new int[n];

s = br.readLine().split(" "); //input in arr

for (int i = 0; i < n; i++) {

arr[i] = Integer.parseInt(s[i]);

}

for (int i = 0; i < n; i++) { //all ranges of helper stored in Data class object array

int srange = i - arr[i];

int erange = i + arr[i];

int totalrange = 2\*arr[i] +1;

if(arr[i]==-1){ //if value is -1,set ranges -1 and index as i

srange=-1;

erange=-1;

totalrange=-1;

data[i] = new Data(srange,erange,totalrange,i);

continue;

}else { //making sure starting range index do not get negative

while (srange < 0) {

totalrange--;

srange++;

}

while (erange >= n) {

totalrange--;

erange--;

}

//Storing helpers start index and end index where it can provide help in Data class //object anlong with total range it provides help

data[i] = new Data(srange, erange, totalrange, i);

//System.out.println(data[i].index);

}

}

//sort according to ranges max to min- insertion sort used here, hence O(n^2) time

for (int i = 0; i < n; i++) {

Data k =new Data(data[i].startrange,data[i].endrange, data[i].range,data[i].index);

//k = data[i].range;

int j =i-1;

while (j>=0&&data[j].range>k.range){

data[j+1] = data[j];

j--;

}

data[j+1]= k;

}

for (int i = 0; i < n; i++) {

//System.out.println(data[i].index);

}

//Sorting in desc

// for (int i = n-1; i >= 0 ; i--) {

// //System.out.println(data[i].range);

// if(data[i].range ==-1){

// sorted[n-(i+1)] = new Data(-1,-1,-1,i);

// }else {

// sorted[n-(i+1)] = new Data(data[i].startrange,data[i].endrange,data[i].range,data[i].index);

// }

// }

for (int i = 0; i < n; i++) { //initializing map array

//System.out.println(sorted[i].index);

map[i] = -1;

}

for (int i = n-1; i >= 0; i--) {

if(data[i].range==-1)break;

for (int j = data[i].startrange; j <= data[i].endrange ; j++) {

if(map[j]==1)continue;

else {

map[j]=1; //indices set to 1 from -1 where helper is covering indices

count++;

data[i].flag=1; //Putting flag as 1 to helpers that contribute to final answer

}

// if(count==n){

// System.out.print(i);

// return;

// }

}

}

count = 0;

for (int i = 0; i < n; i++) {

if (data[i].flag==1) count++;

if(map[i]==-1){

count = -1;

break;

}

}

System.out.println(count);

}

}

class Data{ //To store info on all rangers on a helper

public int startrange; //starting index of helpers help range

public int endrange; //end index of helpers help range

public int range; //total range of helper where it can serve

public int index; //to store original index-used in keeping track

public int flag;

//Constructors

Data(int srange, int erange ,int trange,int indx){

flag=0;

startrange = srange;

endrange = erange;

range = trange;

index=indx;

}

Data(){

startrange=0;

endrange=0;

range=0;

index=0;

flag=0;

}

}

Approach – The input data is stored in class Data which stores data on each helper such as its starting index where it can provide help, ending index where it can provide help and the total range calculated from the index it is present and the value of index.

After storing all data, the array is sorted using insertion sort which I implemented in O(n^2).

After sorting, the data (helpers) is chosen in descending order of their range to serve and flagged if and only if they cover an index not covered by any other helper before. An extra array of size n is used to keep track to indices.