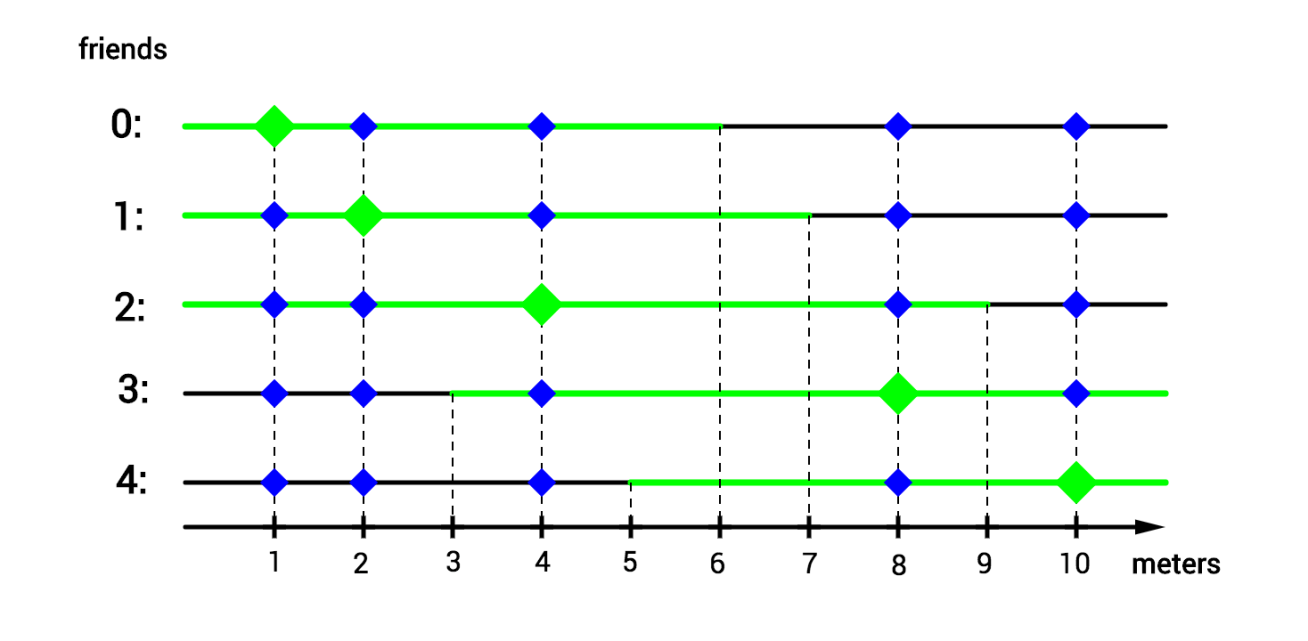
Several friends live along a straight street. They are friends, thus they enjoy visiting each other. However, they are lazy so none of them wants to visit a friend living more than maxDist meters away from them.

Given array houses representing coordinates of points where the friends live (in meters) and an integer maxDist, return an array representing the number of friends each person would be willing to visit.

**Example**

For houses = [1, 2, 4, 8, 10] and maxDist = 5, the output should be  
lazyFriends(houses, maxDist) = [2, 2, 3, 2, 1].

Check out the image below for better understanding:



**Input/Output**

* **[time limit] 3000ms (cs)**
* **[input] array.integer houses**

Sorted non-empty array of distinct integers.

*Guaranteed constraints:*  
1 ≤ houses.length ≤ 104,  
-104 ≤ houses[i] ≤ 104.

* **[input] integer maxDist**

*Guaranteed constraints:*  
2 ≤ maxDist ≤ 105.

* **[output] array.integer**

<https://codefights.com/challenge/dXZBrfLtuSQhSB8Xf/solutions>

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace ConsoleApplication1

{

class Program

{

static int[] lazyFriends(int[] houses, int maxDist)

{

List<int> res = new List<int>();

for (int i = 0; i < houses.Length; i++)

{

int cont = 0;

int izq = i-1;

while (izq >= 0 && houses[i] - houses[izq] <= maxDist)

{

cont++;

izq--;

}

int der = i+1;

while (der < houses.Length && houses[der] - houses[i] <= maxDist)

{

cont++;

der++;

}

res.Add(cont);

}

return res.ToArray();

}

static void Main(string[] args)

{

//int[] houses = {1, 2, 4, 8, 10};

//int maxDist = 5;

//[2, 2, 3, 2, 1]

int[] houses= {-5, 0, 5, 10, 15};

int maxDist = 10;

//[2, 3, 4, 3, 2]

foreach (int elem in lazyFriends(houses, maxDist))

{

Console.Write(elem + " ");

}

Console.ReadLine();

}

}

}

//------------OTRAS SOLUCIONES--------------

1. #include <iostream>

#include <stdio.h>

#include <vector>

#include <map>

using namespace std;

std::vector<int> lazyFriends(std::vector<int> houses, int maxDist) {

    int n = houses.size();

    vector<int> lf;

    map<int,int> map;

    int count =0;

    int index =0;

    for(int i =-10000; i<=10000; i++) {

        if((index<n) && (i==houses[index])){

            count++;

            index++;

        }

        map[i] = count;

    }

    for(int i =0; i<n; i++) {

        int curr = houses[i];

        int left = max(curr-maxDist-1, -10000);

        int right = min(curr +maxDist, 10000);

        lf.push\_back(map[right]-map[left]-1);

    }

    return lf;

}

int main() {

  int houses[] = {1, 2, 4, 8, 10};

  std::vector<int> v;

  for(int i =0; i < 5; i++) {

      v.push\_back(houses[i]);

  }

  int maxDist = 5;

  std::vector<int> res = lazyFriends(v,maxDist);

  for(int i =0; i<res.size(); i++) {

      printf("%d ", res[i]);

    }

   system("pause");

  return 0;

}

static int[] lazyFriends(int[] houses, int maxDist)

{

int n = houses.Length;

List<int> lf = new List<int> ();

Dictionary<int, int> map = new Dictionary<int,int> ();

int count = 0;

int index = 0;

for (int i = -10000; i <= 10000; i++)

{

if ((index < n) && (i == houses[index]))

{

count++;

index++;

}

map[i] = count;

}

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

{

int curr = houses[i];

int left = Math.Max(curr - maxDist - 1, -10000);

int right = Math.Min(curr + maxDist, 10000);

lf.Add(map[right] - map[left] - 1);

}

return lf.ToArray();

}