Eolymp | 11287 | Peak Element

Nama : **Timothy Hosia Budianto**

NRP : **50525211098**

Permasalahan dan pendekatan :

Peak Element

A **peak element** is an element that is strictly greater than its neighbors.

Given a **0**-indexed integer array **a**, find a peak element, and return its index. If the array contains multiple peaks, return the index to any of the peaks. You may imagine that $a_{-1} = a_n = -\infty$. In other words, an element is always considered to be strictly greater than a neighbor that is outside the array.

Input

The first line contains the size of array **n** ($n \le 10^6$). The second line contains **n** integers $a_0, a_1, \ldots a_{n-1}$. It is known that $|a_i| \le 10^9$ and also that $a_i \ne a_{i+1}$ for $i = 0, 1, \ldots, n - 2$..

Output

Print the index of a peak element. If the array contains multiple peaks, return the index to any of the peaks.

Time limit **1** second

Memory limit **128** MiB

Pada soal ini meminta untuk mencari pada index berapakah di sebuah array, ada bilangan yang nilai sebeum dan sesudahnya memiki nilai lebih rendah dari nilai yang ditinjau. Nilai yang akan kita cari ini diberi istilah **peak element** untuk memudahkan. Pada soal ini memiliki constraint waktu 1 detik dan memory 12 mib.

Input example #1	Output example #1
4 1 2 3 1	2

Input example #2	Output example #2
7 1 2 1 3 5 6 4	1

Pada sample test case pertama dilihat bahwa peak element adalah nilai 3 di index 2 karena nilai sebelum dan sesudahnya lebih rendah daripada nilai 3. Pada sample test case 2 terlihat bahwa terdapat 2 peak yaitu nilai 2 dan 6. Tetapi ditinjau dari persyaratan soal, nilai peak yang diambil adalah nilai yang muncul pertama yaitu nilai 2 diindex ke 1.

Abstraksi :

Pada soal ini sebenarnya bisa menggunakan pendekatan untuk mengcompare setiap elemen dalam array untuk mencari peak element, tapi pendekatan tersebut kurang efektif. Oleh karena itu kita akan menerapkan salah satu konsep dalam dynamic programming yaitu konsep **flying table**. Dimana kita hanya akan menyimpan nilai **sebelum, sekarang, dan sesudah** untuk di compare demi menemukan nilai peak element. Selain itu dalam code kita tidak perlu menyimpan nilai variable lain yang tidak dibtuhkan melainkan hanya yang akan dibutuhkan.

Kita akan membuat pseudocode algoritma dalam approach soal ini sebagai berikut

1. N ←
2. **for** N ← with i from 0 to N
3. current ←
4. if(i=0) before = next = current
5. else if(current<next)
6. print i – 1 ; break ;
7. else
8. before=next ; next = current ;

Implementasi Code

```cpp
#include <cstdio>
#include <algorithm>
using namespace std;
#define gc getchar
#define ULL unsigned long long
typedef long long ll;

ll read(){
    ULL value = 0;bool ne=0;
    char c = gc();
    while(c==' ' or c=='\n') c =gc();
    if(c=='-'){ne = 1;c = gc();}
    while(c>='0' and c<='9'){
        value = (value<<3)+(value<<1)+c-'0';c = gc();}
    if(ne) value*=-1;
    return value;
}

int main()
{
    ll N;
    int beffore,next, current;
    N=read();
    for(int i =0; i<=N; i++){
        current=read();
        if(i==0) beffore=next=current;
        else if(current<next){
            printf("%d", i-1);
            return 0;
        }
        else{
            beffore=next;
            next= current;
        }
    }

    return 0;
}
```

Line 1-6 adalah library dan define yang akan digunakan dalam kode

Line 8-17 adalah fungsi read untuk bypass handler

Line 21-22 adalah variable yang akan digunakan, dimana nilai yang kita simpan hanya aka nada 3 yaitu before after dan current

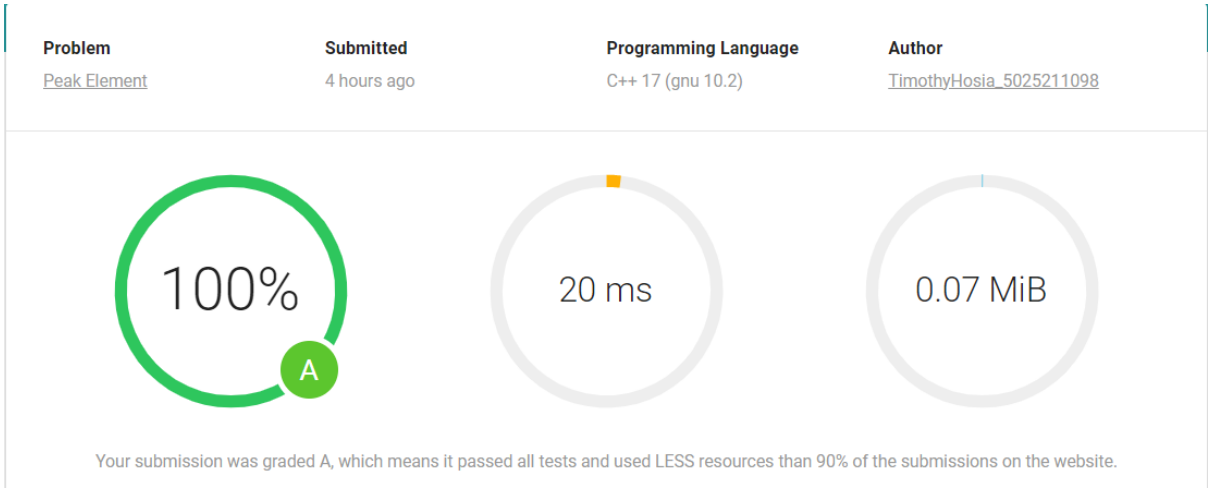
Line 23-24 adalah melakukan perulangan sebanyak N u

Line 25 pertama memasukan nilai current

Line 26 jika ini iterasi pertama nilai before dan next akan disama dengankan current

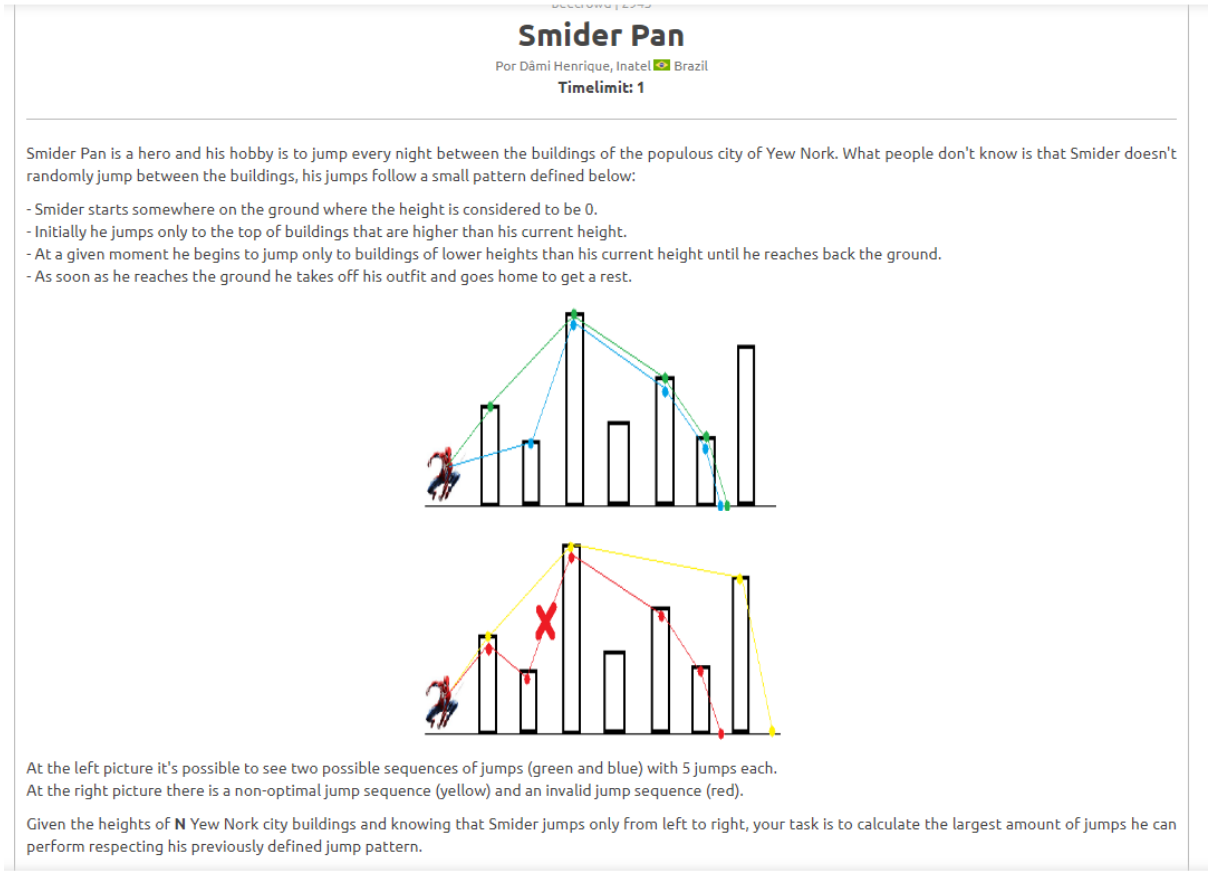
Line 27-30 akan mengecheck jika nilai next kurang dari nilai next akan meprintkan index ke berapa lalu break

Line 31-34 adalah mengupdate nilai before next dan current untuk iterasi berikutnya



Beecrowd | 2943 | Smider Pan

Permasalahan dan pendekatan :



Smider Pan

Por Dâmi Henrique, Inatel Brazil

Timelimit: 1

Smider Pan is a hero and his hobby is to jump every night between the buildings of the populous city of Yew Nork. What people don't know is that Smider doesn't randomly jump between the buildings, his jumps follow a small pattern defined below:

- Smider starts somewhere on the ground where the height is considered to be 0.
- Initially he jumps only to the top of buildings that are higher than his current height.
- At a given moment he begins to jump only to buildings of lower heights than his current height until he reaches back the ground.
- As soon as he reaches the ground he takes off his outfit and goes home to get a rest.

At the left picture it's possible to see two possible sequences of jumps (green and blue) with 5 jumps each.
At the right picture there is a non-optimal jump sequence (yellow) and an invalid jump sequence (red).

Given the heights of **N** Yew Nork city buildings and knowing that Smider jumps only from left to right, your task is to calculate the largest amount of jumps he can perform respecting his previously defined jump pattern.

Soal ini dengan perumpamaan spiderman yang terus melompat dari suatu Gedung ke Gedung lain, meminta untuk mencari berapa jumlah paling Gedung paling banyak yang dapat dilewati spiderman dengan syarat harus **ascending** kemudian **descending**.

Input

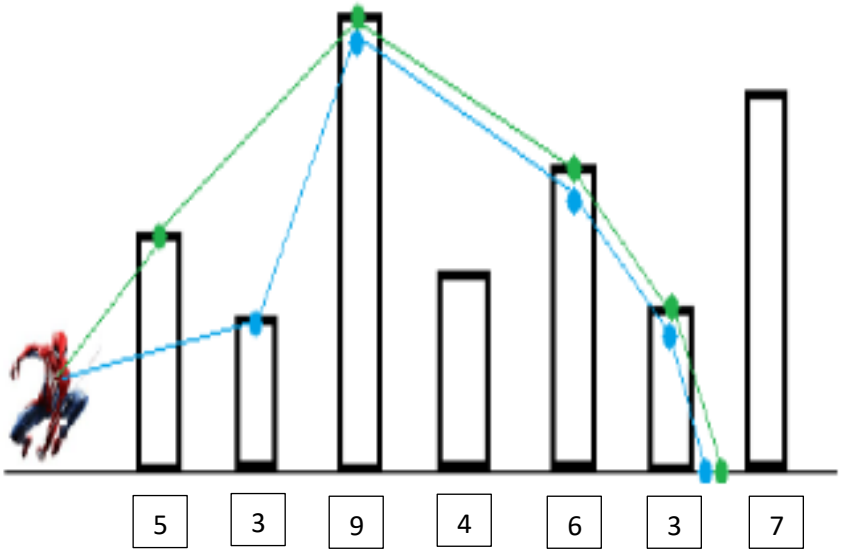
The first line contains an integer **N** ($1 \le N \le 10^3$) representing the number of Yew Nork buildings.
The second line will have **N** integers H_i ($1 \le H_i \le 10^6$), those being the heights of the buildings.

Output

Print a single integer representing the greatest possible amount of jumps that Smider Pan will be able to perform.

Input Sample	Output Sample
7 5 3 9 4 6 3 7	5
1 5	2

Soal ini akan memberikan jumlah Gedung yang ada, kemudian ketinggian Gedung yang disimbolkan sebagai sebuah integer dalam sebuat list angka. Kemudian kita diminta meng outputkan jumlah Langkah yang harus diambil oleh spiderman. Dilihar dari contoh test case akan diilustrasikan sebgai berikut agar mudah dipahami. Pada test case 1 sebagai berikut

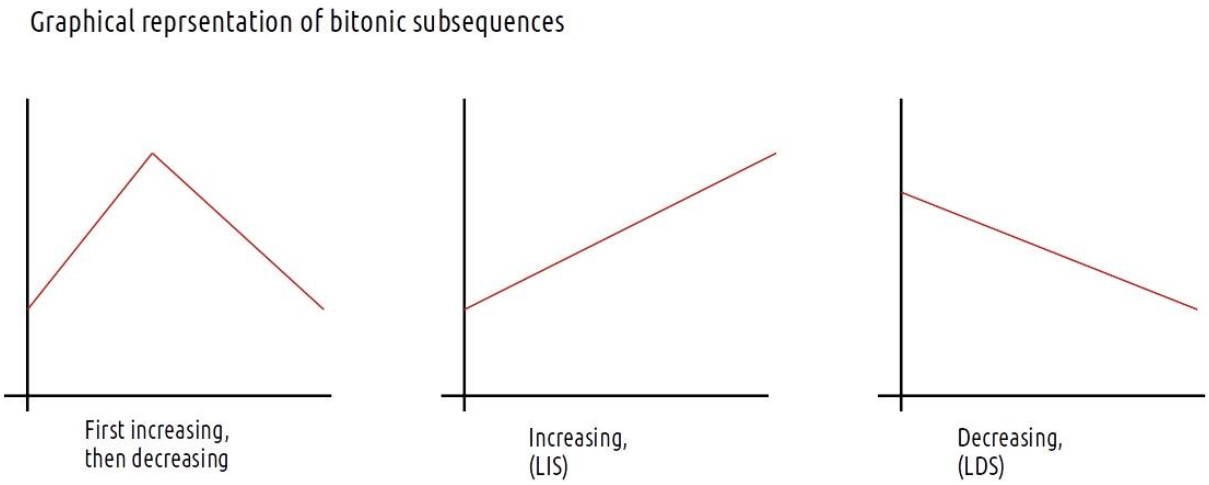


Berikut adalah jalan yang bisa diambil jika kita mengikuti syarat ascending lalu descending. Sedangkan pada test case 2, akan mengoutputka nilai 2 karna hanya ada 1 gedung jadi hanya membutuhkan Langkah naik dan turun.

Abstraksi :

Soal untuk mecari sequence increasing dan decreasing adalah implementasi dari **longest sequence bitonic.** Kita diminta untuk mencari pajang dari bitonic sequence terpanjang yang bisa didapatkan. Sebenarnya sebuah sequence yang increasing saja bisa

disebut bitonic dengan bagian decreasingnya adalah kosong. Sebaliknya decreasing order bisa dibilang bitonic dengan bagian increasingnya adalah kosong.



Sumber dari https://takeuforward.org/data-structure/longest-bitonic-subsequence-dp-46/

Selain itu pada implementasi kita akan menggunakan konsep **dynamic programming** dimana kita akan membuat 2 buah list array dimana kita akan menyimpan sequences yang akan terjadi. Kita akan membuat **lis[]** untuk menyimpan longest increasing sequences dan **lds[]** untuk menyimpang longest decreasing sequences. Pada akhirnya kita akan me returnkan nilai maximal dari lis[i] + lds[i] – 1 dimana i berindex 0 sampai n-1.

Mari kita lebih mengerti soal diatas dengan penjelasan sebuah testcase

Given array=	10	22	9	33	21	50	41	60	80	3

Akan menghasilkan

Longest increasing subsequence:

Given array	10	22	9	33	21	50	41	60	80	3
Longest length	1	2	1	3	2	4	4	5	6	1
Increasing subsequences	10	10 22	9	10 22 33	10 21	10 22 33 50	10 22 33 41	10 22 33 50 60	10 22 33 50 60 80	3

Longest decreasing subsequence:

Given array	10	22	9	33	21	50	41	60	80	3
Longest length	3	3	2	3	2	3	2	2	2	1
Decreasing subsequences	10 9 3	22 9 3	9 3	33 21 3	21 3	50 41 3	41 3	60 3	80 3	3

Pada table diatas bisa dilhat bahwa sequence terpanjang yang bisa didapat adalah penjumlahan antara {10, 22, 33, 50, 60, 80} dengan panjang 6 dijumlahkan dengan Decreasing subsequence: {80, 3} dengan panjang 2. Max length = (6 + 2) - 1 = 7 // di minus 1 karna nilai 80 muncul 2 kali. Sehingga menghasilkan Longest Bitonic Subsequence obtained: { 10, 22, 33, 50, 60, 80, 3 }

Implementasi Code

Berikut implementasi kode beserta penjelasan kode

```
1  /* Dynamic Programming implementation of longest bitonic subsequence problem */
2  #include<stdio.h>
3  #include<stdlib.h>
4  using namespace std;
5  #define gc getchar
6  #define ULL unsigned long long
7  typedef long long ll;
8
```

Library dan define yang akan digunakan

```
int read(){
    ULL value = 0;bool ne=0;
    char c = gc();
    while(c==' ' or c=='\n') c =gc();
    if(c=='-'){ne = 1;c = gc();}
    while(c>='0' and c<='9'){
        value = (value<<3)+(value<<1)+c-'0';c = gc();}
    if(ne) value*=-1;
    return value;
}
```

Fungsi read untuk bypass handler

```
/* lbs() returns the length of the Longest Bitonic Subsequence in
    arr[] of size n. The function mainly creates two temporary arrays
    lis[] and lds[] and returns the maximum lis[i] + lds[i] - 1.

    lis[i] ==> Longest Increasing subsequence ending with arr[i]
    lds[i] ==> Longest decreasing subsequence starting with arr[i]
*/
int lbs( int arr[], int n )
{
    int i, j;

    /* Allocate memory for LIS[] and initialize LIS values as 1 for
       all indexes */
    int *lis = new int[n];
    for (i = 0; i < n; i++)
       lis[i] = 1;

    /* Compute LIS values from left to right */
    for (i = 1; i < n; i++)
       for (j = 0; j < i; j++)
          if (arr[i] > arr[j] && lis[i] < lis[j] + 1)
             lis[i] = lis[j] + 1;
```

```
    /* Allocate memory for lds and initialize LDS values for
       all indexes */
    int *lds = new int [n];
    for (i = 0; i < n; i++)
       lds[i] = 1;

    /* Compute LDS values from right to left */
    for (i = n-2; i >= 0; i--)
       for (j = n-1; j > i; j--)
          if (arr[i] > arr[j] && lds[i] < lds[j] + 1)
             lds[i] = lds[j] + 1;


    /* Return the maximum value of lis[i] + lds[i] - 1*/
    int max = lis[0] + lds[0] - 1;
    for (i = 1; i < n; i++)
      if (lis[i] + lds[i] - 1 > max)
          max = lis[i] + lds[i] - 1;
    return max;
}
```

```
63  /* Driver program to test above function */
64  int main()
65  {
66    int N = read();
67    int arr[N];
68
69    for(int i =0; i<N ; i++){
70      arr[i]=read();
71    }
72    int n = sizeof(arr)/sizeof(arr[0]);
73
74    if(N==1){
75      printf("2\n");//jika 1 gedung menghasilkan output 2
76    }else{
77      printf("%d\n", lbs( arr, n )+ 1 );//ditambahkan 1 untuk langkah turun
78    }
79
80    return 0;
81  }
```

Bukti accept

SUBMISSION # 30731399

PROBLEM:	2943 - Smider Pan
ANSWER:	**Accepted**
LANGUAGE:	C++17 (g++ 7.3.0, -std=c++17 -O2 -lm) [+0s]
RUNTIME:	0.000s
FILE SIZE:	2.07 KB
MEMORY:	-
SUBMISSION:	11/8/22, 4:35:13 PM