Find a peak element

Given an array of integers. Find a peak element in it. An array element is peak if it is NOT smaller than its neighbors. For corner elements, we need to consider only one neighbor. For example, for input array {5, 10, 20, 15}, 20 is the only peak element. For input array {10, 20, 15, 2, 23, 90, 67}, there are two peak elements: 20 and 90. Note that we need to return any one peak element.

Following corner cases give better idea about the problem.
1) If input array is sorted in strictly increasing order, the last element is always a peak element. For example, 50 is peak element in {10, 20, 30, 40, 50}.
2) If input array is sorted in strictly decreasing order, the first element is always a peak element. 100 is the peak element in {100, 80, 60, 50, 20}.
3) If all elements of input array are same, every element is a peak element.

It is clear from above examples that there is always a peak element in input array in any input array.

A **simple solution** is to do a linear scan of array and as soon as we find a peak element, we return it. The worst case time complexity of this method would be O(n).

Can we find a peak element in worst time complexity better than O(n)?
We can use Divide and Conquer to find a peak in O(Logn) time. The idea is Binary Search based, we compare middle element with its neighbors. If middle element is greater than both of its neighbors, then we return it. If the middle element is smaller than the its left neighbor, then there is always a peak in left half (Why? take few examples). If the middle element is smaller than the its right neighbor, then there is always a peak in right half (due to same reason as left half). Following is C implementation of this approach.

```c
// A divide and conquer solution to find a peak element element
#include <stdio.h>

// A binary search based function that returns index of a peak element
int findPeakUtil(int arr[], int low, int high, int n)
{
    // Find index of middle element
    int mid = low + (high - low)/2;  /* (low + high)/2 */

    // Compare middle element with its neighbours (if neighbours exist)
    if ((mid == 0 || arr[mid-1] <= arr[mid]) &&
            (mid == n-1 || arr[mid+1] <= arr[mid]))
        return mid;

    // If middle element is not peak and its left neighbor is greater than it
    // then left half must have a peak element
    else if (mid > 0 && arr[mid-1] > arr[mid])
        return findPeakUtil(arr, low, (mid -1), n);

    // If middle element is not peak and its right neighbor is greater than i
    // then right half must have a peak element
    else return findPeakUtil(arr, (mid + 1), high, n);
}

// A wrapper over recursive function findPeakUtil()
int findPeak(int arr[], int n)
{
    return findPeakUtil(arr, 0, n-1, n);
}

/* Driver program to check above functions */
int main()
{
    int arr[] = {1, 3, 20, 4, 1, 0};
    int n = sizeof(arr)/sizeof(arr[0]);
    printf("Index of a peak point is %d", findPeak(arr, n));
    return 0;
}
```

Output:

```
Index of a peak point is 2
```

Time Complexity: O(Logn) where n is number of elements in input array.

Exercise:
Consider the following modified definition of peak element. An array element is peak if it is greater than its neighbors. Note that an array may not contain a peak element with this modified definition.

References:
http://courses.csail.mit.edu/6.006/spring11/lectures/lec02.pdf
http://www.youtube.com/watch?v=HtSuA80QTyo

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Related Topics:

- Find Union and Intersection of two unsorted arrays
- Pythagorean Triplet in an array
- Maximum profit by buying and selling a share at most twice
- Design a data structure that supports insert, delete, search and getRandom in constant time
- Print missing elements that lie in range 0 – 99
- Iterative Merge Sort
- Group multiple occurrence of array elements ordered by first occurrence
- Given a sorted and rotated array, find if there is a pair with a given sum

Tags: Divide and Conquer

Tweet 8+1 1

Writing code in comment? Please use **ideone.com** and share the link here.

94 Comments GeeksforGeeks 1 Login

Recommend Share Sort by Newest

Join the discussion…

Jerry Goyal • 9 days ago
those who are arguing that method won't work for duplicates read the definition of peak element carefully "An array element is peak if it is NOT smaller than its neighbors". so for {5,6,6,6,6,6,4} peak will be any "6".
• Reply • Share ›

anuj • a month ago

what if the input is arr[]={1,5,5,4,3,2,4,5,1}
5 and 5 are same so will it consider 5 a peak or wont return any peak element ?

• Reply • Share ›

Rajesh Jaiswal • a month ago

if the input is int arr[] = {1, 3, 20, 4, 1, 0,56}; this will ignore right half , So its wrong

• Reply • Share ›

Newbie Rajesh Jaiswal • a month ago

We just need to find any peak that is present so its not wrong

• Reply • Share ›

Guest • 2 months ago

By this logic , if the middle element is smaller than the immediate left element then I completely ignore the right subarray. Which is not the right thing to do

• Reply • Share ›

Guest • 3 months ago

Gives wrong o/p for 1, 19, 15, 19, 28, 32, 39. Flaw in the logic. It assume that if mid is not smaller than mid-1 then peak lies on right side. This example exploit this loophole.

• Reply • Share ›

guest Guest • 2 months ago

39 is a peak too, according to definition.

• Reply • Share ›

Mohammad Azeem • 3 months ago

I found the right rotate logic a bit non-intuitive, so I modified the logic a bit. Let's assume that negative elements are to be fixed at even positions and positive elements are to be fixed at odd positions. While scanning the array from left to right, if we encounter a negative element at an odd index or a positive element at an even index, we scan the rest of the array for a replacement & swap that element with the replacement. When no replacements are found, our work is done. Code in Java at: http://ideone.com/NEdOXW

• Reply • Share ›

Guest • 3 months ago

<script src="http://ideone.com/e.js/NEdOXW" type="text/javascript"></script>

• Reply • Share ›

Aman Joshi • 3 months ago

My code for Peak Value.. Suggestions are appreciated. thank You

int x=0,pos; //Global Initialization

```
int A[N];

void peak(int low,int high)

{

int mid;

mid=low+(high-low)/2;

if(A[mid]>x)

{

x=A[mid];

pos=mid;
```

see more

• Reply • Share ›

Nagabhushan Sundarakrishna • 4 months ago

There are particular cases where this algorithm does not work..
like, when you have repeated numbers next to each other, algorithm does not know which side to choose and thinks it is at a peak which is incorrect.

Consider below set..

{5, 0, 0, 0, 0}

This is the output (which is incorrect):
Index of a peak point is 2

• Reply • Share ›

Aman Joshi Nagabhushan Sundarakrishna • 3 months ago

The above algorithm is for finding any one of the peak elements according to the definition given above. Peak does not mean the largest element in the set. In your example 5 & 0 are the two peaks and we need to print any one of them.

• Reply • Share ›

Nagabhushan Sundarakrishna Aman Joshi • 3 months ago

I know it will not find the highest peak.. But according to the algorithm if it lands on an index which does not have any higher or lower values on either sides, then it will consider it as a peak (which is the definition of a peak). I found the same kind of logic in other places as well, which confused me, so posted this question here. I just found the defn of peak to be flawed. Thats all.

• Reply • Share ›

Gaurav Verma • 4 months ago

this program does not find each and every peak point in an array... For input array {10, 20, 15, 2, 23, 90, 67}, there are two peak elements: 20 and 90

here is a program that finds every possible peak point:-

```
#include <stdio.h>

#include <stdlib.h>

void check(int **,int,int);

void peak(int *n,int,int);

int main()

{

int a[]={10,20,30,5,20,3,90,97,10};

int l=0,t=(sizeof(a)/sizeof(a[1]))-1;
```

see more

• Reply • Share ›

raghul Gaurav Verma • 4 months ago

the algorithm is to find A peak, not all peaks.

• Reply • Share ›

EigenHarsha • 5 months ago

According To Defination "An array element is peak if it is NOT smaller than its neighbors", and we have input {1,2, 80, 5, 6, 8, 17, 25, 60}; than it doesn't work ! if my input set is wrong than suggest me !

• Reply • Share ›

GeeksforGeeks Mod EigenHarsha • 4 months ago

Please take a closer look. The above program returns index of peak element, not the element itself. For your example, the returned value is 8 which is index of 60.

• Reply • Share ›

prahaladd • 7 months ago

what happens for input array {1, 3, 4, 2, 7, 5, 8, 6}? here we would get only one of the peaks

• Reply • Share ›

Random prahaladd • 6 months ago

Above algorithm find only one occurrence so it will always miss all occurrences except one.

• Reply • Share ›

guest • 8 months ago

what happens in this case 8 , 9, 5, 6 , 2, 1, 3? middle element=6 which is greater than 5. However, the peak is in the left. Am I missing something?

• Reply • Share ›

RK ➦ guest • 8 months ago

6 will be peak element

2 • Reply • Share ›

Guest • 9 months ago

the algo is wrong. consider this case.
1 2 2 4 4 10
the answer should be either 1 or 3 but its giving 5
the culprit is at

else if (mid > 0 && arr[mid-1] > arr[mid])
return findPeakUtil(arr, low, (mid -1), n);

it should be changed to

else if (mid > 0 && arr[mid-1] >= arr[mid])
return findPeakUtil(arr, low, (mid -1), n);

2 • Reply • Share ›

RK ➦ Guest • 9 months ago

10 is also a peak element because 10 has only one neighbor 4 and 10>4 so 10 is also a peak element and this algo have to print only one peak element so it can print anyone of 10/4/2 (index 1/3/5)

1 • Reply • Share ›

_rahulg • a year ago

Instead of :

if ((mid == 0 || arr[mid-1] <= arr[mid]) &&
(mid == n-1 || arr[mid+1] <= arr[mid]))
shouldn't it be :

if ((mid == low || arr[mid-1] <= arr[mid]) &&
(mid == high || arr[mid+1] <= arr[mid]))

• Reply • Share ›

srinivas devaki _rahulg • 9 months ago

NO,
that is for extreme ones. reread the 1, 2 corner cases

Reply • Share ›

AlienOnEarth • a year ago

can we modify binary search in such a way that it will return all the peak elements?

3 • Reply • Share ›

The_Geek AlienOnEarth • 7 months ago

Not possible, then it will need O(n) time.

Reply • Share ›

arjomanD • a year ago

Why don't seek for the maximum element in Array ? It's Guaranteed that maximum element exists and we know it's peak element and also in some cases (like sorted arrays) it's the only Answer !

Reply • Share ›

svcongnghe arjomanD • a year ago

maximum element in Array is only 1 peak element whereas "peak elements" can be more than 1 in a array.

Reply • Share ›

nile arjomanD • a year ago

for dat the complexity is O(n) ...(for finding max) , bt the sol gives us a betr complexity of O(log(n))...

2 • Reply • Share ›

zzer • a year ago

the boundary checking is not clear. We can check bounday when the size if 1 or 2(we just return the bigger element) as below:

```
int find_peak(int arr[],int low,int high)

{

if(low == high)

return arr[low];

if(low +1 == high)

return arr[low]>arr[high]?arr[low]:arr[high];
int mid = low + (high-low)/2;

if(arr[mid] >= arr[mid-1] && arr[mid] >= arr[mid+1])

return arr[mid];

else if(arr[mid] < arr[mid-1])

return find_peak(arr,low,mid-1);

else

return find_peak(arr,mid+1,high);

}
```

• Reply • Share ›

Saurabh • a year ago

"If the middle element is smaller than the its left neighbor, then there is always a peak in left half." If the numbers from start are in decreasing order till middle then middle element is smaller than the left neighbor, however there is no peak in left subarray. Do we consider Arr[-1] == Arr[n+1] == -Infinity ?.

• Reply • Share ›

Jonathan Chen ➦ Saurabh • a year ago

If I'm understanding correctly, you are saying "If the numbers from element[0] to element[middle] are all decreasing", that right?

Then, it's the first element of the array (element[0]). Corner case.
Since it's decreasing, the right neighbor of element[0] is smaller than it. And, element[0] has no left neighbor. So, it's a peak.

2 • Reply • Share ›

santhosh • a year ago

{5,0,1,0,2,10,9} for this what will be peak element.

• Reply • Share ›

Arthur ➦ santhosh • a year ago

For this the peek value is 5, 1, 10. Since we are interested only in one value, by using D&C as mentioned above the result will be 5.

• Reply • Share ›

sambhavsharma • a year ago

If the middle element is smaller than the its left neighbor, then there is always a peak in left half.

Pretty nice deduction.

5 • Reply • Share ›

Priyanka Gupta • a year ago

I think this algorithm will have complexity of O(logn) only for distinct elements, for duplicates worst case can become O(n). As you said if all elements are equal every element is peak, we need to make sure of it, that every element is equal.

```
if (a[mid] < a[mid+1])
{ // search right }
else if (a[mid] < a[mid-1])
{ // search left }
else // we are at a plateau, not up-slope, not down-slope
{
// search right peak
// search left peak
// return the largest or equal
}
```

2 • Reply • Share ›

Parvin Taheri • a year ago

for the exercise part, I think we can not do binary search. take these two test cases: {20, 1, 10,10,1} and {2, 1, 10, 10,1}. in the first one there is only one peak in the left side and in the second one there is no peak. so I think we need to search both sides of the mid point.

• Reply • Share ›

Ankit Chaudhary Parvin Taheri • a year ago

Agree with u, binary search will not work for exercise part.

• Reply • Share ›

Raj Baraiya Ankit Chaudhary • 5 months ago

In question it is mentioned that peak element if it is greater than or equal to.

• Reply • Share ›

codeKaichu • a year ago

the reason binary search works (i.e if mid < left element, there's atleast one peak on left and similar case for right part) can be better visualized if you can draw a plot of the points.

• Reply • Share ›

pinkfloyda • a year ago

Recently in my blog, I have discussed this problem why it works, check http://comtechstash.wordpress....

• Reply • Share ›

annonymoe • a year ago

with small logical comparison of the derivative at low, high and mid you can do much better at finding a peak rather than giving the first and the last element typically for most of the cases

• Reply • Share ›

RahulMJ • 2 years ago

//C++ implementation which returns a list of peaks found in the array
// A divide and conquer solution to find a peak element element

#include <iostream>

#include <vector>

using namespace std;

// A binary search based function that returns index of a peak element

void findPeakUtil(int arr[], int low, int high, int n, vector<int> *indexList)

{

// Fin index of middle element

int mid = low + (high - low)/2; /* (low + high)/2 */

// Compare middle element with its neighbours (if neighbours exist)

see more

• Reply • Share ›

manjunathak • 2 years ago

The questions is not clear. Is it to find the list of peak elements or at least one peak element?

• Reply • Share ›

TA manjunathak • a year ago

find 'a' peak

1 • Reply • Share ›

Sriharsha g.r.v • 2 years ago

only one of the peak elements are being returned.....if i am not wrong..this could look bad in this case {10,8,2,4,3,7} ..it might return a[0]which is 10 leaving a[3] which is 4 and its peek element as well.....

4 • Reply • Share ›

Guest • 2 years ago

only one of the peak elements are being returned.....if i am not wrong..thios could look bad in this case {10,8,2,4,3,7} ..it might return a[0]which is 10 leaving a[3] which is 4 and peek element as well.....

Reply • Share ›

Akshay • 2 years ago

This would be a perfect code

#include <stdio.h>

#include<conio.h>

int main()

{

int a[]={10, 20, 15, 2, 23, 90, 67};

int len=sizeof(a)/sizeof(a[0]);

int i;

for(i=0;i<len;i++) {="" if(i="=0&amp;&amp;a[i]">a[i+1])

printf("\nPeak element found at position %d",i+1);

see more

Reply • Share ›

Akshay • 2 years ago

This approach is totally wrong, as you cannot deny the fact that even if the mid element has greater value on one side, the other side can also have a peak element. It does not even satisfy the provided test case of {10, 20, 15, 2, 23, 90, 67}.
It should be done with linear approach.

1 • Reply • Share ›

Load more comments

Subscribe Add Disqus to your site Privacy DISQUS

Google™ Custom Search

-
-
-
-
    - Interview Experiences
    - Advanced Data Structures
    - Dynamic Programming
    - Greedy Algorithms
    - Backtracking
    - Pattern Searching
    - Divide & Conquer
    - Mathematical Algorithms
    - Recursion
    - Geometric Algorithms
-

- ## Popular Posts

    - All permutations of a given string
    - Memory Layout of C Programs
    - Understanding "extern" keyword in C
    - Median of two sorted arrays
    - Tree traversal without recursion and without stack!
    - Structure Member Alignment, Padding and Data Packing
    - Intersection point of two Linked Lists
    - Lowest Common Ancestor in a BST.

  - Check if a binary tree is BST or not
  - Sorted Linked List to Balanced BST
- Follow @GeeksforGeeks

- ## Recent Comments

  - Nikhil kumar

    public class...

    Print missing elements that lie in range 0 – 99 · 5 minutes ago
  - Ashish Aggarwal

    Try Data Structures and Algorithms Made Easy -...

    Algorithms · 27 minutes ago
  - Vlad

    Thanks. Very interesting lectures.

    Expected Number of Trials until Success · 1 hour ago
  - cfh

    My implementation which prints the index of the...

    Longest Even Length Substring such that Sum of First and Second Half is same · 1 hour ago
  - Gaurav pruthi

    forgot to see that part ;)

    Bloomberg Interview | Set 1 (Phone Interview) · 2 hours ago
  - saeid aslami

    thanks

    Greedy Algorithms | Set 7 (Dijkstra's shortest path algorithm) · 2 hours ago
-