**Name**: Ravi Jamanbhai Makwana

**Roll No**: 21BCP418

Div-6, G-12

**Assignment-10**

**Aim: (Question)**

Suppose you are given an array 𝐴 with 𝑛 entries, with each entry holding a distinct number. You are told that the sequence of values 𝐴[1], 𝐴[2],...,𝐴[𝑛] is unimodal. That is, for some index 𝑝 between 1 and 𝑛, the values in the array entries increase up to position 𝑝 in 𝐴 and then decrease the remainder of the way until position 𝑛. (So, if you were to draw a plot with the array position 𝑗 on the 𝑥-axis and the value of the entry 𝐴[𝑗] on the 𝑦-axis, the plotted points would rise until 𝑥-value 𝑝, where they’d achieve their maximum value, and then fall from there on). You’d like to find the “peak entry” 𝑝 without having to read the entire array - in fact, by reading as few entries of 𝐴 as possible. Show how to find the entry 𝑝 by reading at most 𝑂(𝑙𝑜𝑔 𝑛) entries of 𝐴.

**Idea:**

Using Binary Search, check if the middle element is the peak element or not. If the middle element is the peak element, then return the mid value. If not then check if the element on the right side is greater than the middle element then there is always a peak element on the right side. If the element on the left side is greater than the middle element then there is always a peak element on the left side.

**Algorithm:**

1. Assign the data to the array.

2. Call findPeakEntry() function with ‘arr’ the array of data, start and end index in the argument list.

3. Assign the mid of subpart of the array.

4. Check if the value at mid is greater than both of its neighbours then return mid as peak.

5. Otherwise, check if the value at the right of mid is greater than mid then send second sub-part of the array into findPeakEntry ().

6. Otherwise, check if the value at the left of mid is greater than mid then send first sub-part of the array into findPeakEntry ().

7. Return to main and display the peak value.

8. Exit.

**Program:**

/\*

The solution is achieved using concept of binary searching throughout the array,

which is done using recursion

We take the middle element of the array, and check if it is the peak element we're trying to find

[The peak element has the following PROPERTY: Value of elements right before and after the peak element,

will have lesser value than the peak element.

This is a unique property for the peak element.]

If we find it, we return its value, otherwise,

we recursively call the function accordingly to the right or left half of the array.

\*/

#include <bits/stdc++.h>

#include <vector>

using namespace std;

int findPeakEntry(vector <int> v, int low, int high){

    int mid = (low+high)/2;

    //if we have yet not reached the peak, and are on the left of the peak(where values are rising)

    if (v[mid] > v[mid-1] && v[mid] < v[mid+1]){

        return findPeakEntry(v,mid+1,high);

    }

    //if we have gone to the right side of the peak (where values are falling)

    else if (v[mid] < v[mid-1] && v[mid] > v[mid+1]){

        return findPeakEntry(v,low,high-1);

    }

    //we found the peak element, who follows the above mentioned property

    else{

        return v[mid];

    }

}

int main(){

    vector <int> v;

    v = {3,9,16,45,91,156,984,784,653,641,599,481,411,321,222,198,47,43,22,1};

    int n = v.size();

    int peakEntry = findPeakEntry(v,0,n-1);

    cout<<"The Peak Entry of the array is: "<<peakEntry;

    return 0;

}

**Output:**



**Analysis:**

**Time Complexity Analysis:**

**Time complexity**: **O(log N):** Where N is the number of elements in the input array.

**Auxiliary Space**: **O(log N)**: As recursive call is there; hence implicit stack is used.