**TITLE OF LAB: (PROBING PEAK IN ARRAYS)**

**LAB REPORT NO.05**



**Spring 2022**

**CSE-210L Data Structures and Algorithm Lab**

Submitted by

Name: **Safi Ullah Khan**

Registration No. **20PWCSE1943**

Class Section: **B**

“On my honor, as student of University of Engineering and Technology, I have neither given nor received unauthorized assistance on this academic work.”

Student Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Submitted to:

**Dr. Khurram Shehzad Khattak**

(Friday, July 29th, 2022)

**Department of Computer Systems Engineering**

**University of Engineering and Technology, Peshawar**

**OBJECTIVES OF THE LAB**

‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐

In this lab, we will learn about some basic techniques and algorithms to probe peak in.

* One Dimensional Array
* Two-Dimensional Array

‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐‐

**Sorting Algorithms:**

A Sorting Algorithm is used to rearrange a given array or list elements according to a comparison operator on the elements. The comparison operator is used to decide the new order of element in the respective data structure.

**Peak finding:**

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.

## **Task 01**

Probe peak element in one dimensional array and analyze its worst, best and average case complexity.

**Screenshot of Input:**

#include<iostream>

using namespace std;

int Peak\_Search(int \*arr,int start,int end); //O(1)

int main() //O(1)

{

int size,i,peak; //O(1)

cout<<"Enter Size of Array : "; //O(1)

cin>>size; //O(1)

int arr[size]; //O(1)

cout<<"Enter element of array : "; //O(1)

for(int i=0;i<size;i++) //O(N)

{

cin>>arr[i]; //O(1)

}

peak=Peak\_Search(arr,0,size-1); //O(1)

cout<<peak<<" Is peak: \n"; //O(1)

return 0; //O(1)

}

int Peak\_Search(int \*arr,int start,int end) //O(1)

{

int mid,i; //O(1)

mid=(start+end+1)/2; //O(1)

if((arr[mid]>arr[mid+1] && mid==start) || (arr[mid]>arr[mid-1] && mid==end)) //O(1)

{

return arr[mid]; //O(1)

}

else if(arr[mid] > arr[mid-1] && arr[mid] > arr[mid+1]) //O(1)

{

return arr[mid]; //O(1)

}

else if(arr[mid]<=arr[mid+1]) //O(1)

{

return Peak\_Search(arr,mid+1,end); //O(1)

}

else if(arr[mid]<=arr[mid-1]) //O(1)

{

return Peak\_Search(arr,start,mid-1); //O(1)

}

}

// 22\*O(1) // 1\*O(n) // 0\*O(n/2)

**Since Order of Complexity is O(n) in this case**

**Screenshot of Output:**



## **Task 02**

Probe peak element in two-dimensional array and analyze its worst, best and average case complexity.

**Screenshot of Input:**

#include<iostream>

using namespace std;

const int MAX=100; //O(1)

int findPeak(int arr[][MAX],int rows, int columns); //O(1)

int findpeakrec(int arr[][MAX],int rows, int columns, int mid); //O(1)

int findMAX(int arr[][MAX],int rows,int mid,int &max); //O(1)

int main ()

{

int arr[][MAX]={{1,8,6,9},{14,13,120,11},{19,9,1000,31},{16,28,59,20}}; //O(1)

int rows=4,columns=4; //O(1)

for(int i=0;i<4;i++) //O(1)

{

for(int j=0;j<4;j++) //O(1)

{

cout<<arr[i][j]<<"\t"; //O(1)

if(j==3) //O(1) {

cout<<endl<<endl; //O(1) } }

cout<<endl; //O(1)

cout<<"The Peak Value In the Multi Dimensional Array :: //O(1) "<<findPeak(arr,rows,columns);

return 0; //O(1)

}

int findPeak(int arr[][MAX],int rows, int columns) //O(1)

{

return findpeakrec(arr,rows,columns,columns/2); //O(1)

}

int findpeakrec(int arr[][MAX],int rows, int columns, int mid) //O(1)

{

int max=0; //O(1)

int max\_index; //O(1)

max\_index=findMAX (arr,rows,mid,max); //O(1)

if(mid==0 || mid==columns-1) //O(1)

{

return max; //O(1)

}

if(max>=arr[max\_index][mid-1] && max>=arr[max\_index][mid+1]) //O(1)

{

return max; //O(1)

}

if(max<arr[max\_index][mid-1]) //O(1)

{

return findpeakrec(arr,rows,columns,0); //O(N/2)

}

if(max<arr[max\_index][mid+1]) //O(1)

{

return findpeakrec(arr,rows,columns,mid); //O(N/2)

}

}

int findMAX(int arr[][MAX],int rows,int mid,int &max) //O(1)

{

int max\_index=0; //O(1)

for(int i=0;i<rows;i++) //O(N)

{

if(max<arr[i][mid]) //O(1)

{

max=arr[i][mid]; //O(1)

max\_index=i; //O(1)

}

}

return max\_index; //O(1)

}

// 32\*O(1) // 1\*O(n) // 2\*O(n/2)

**Since Order of Complexity is O(nLogn) in this case.**

**Screenshot of Output:**

