Name: Akshita Pathak

Roll No: 102203796

Subgroup: 2CO18

DA Lab Assignment-1

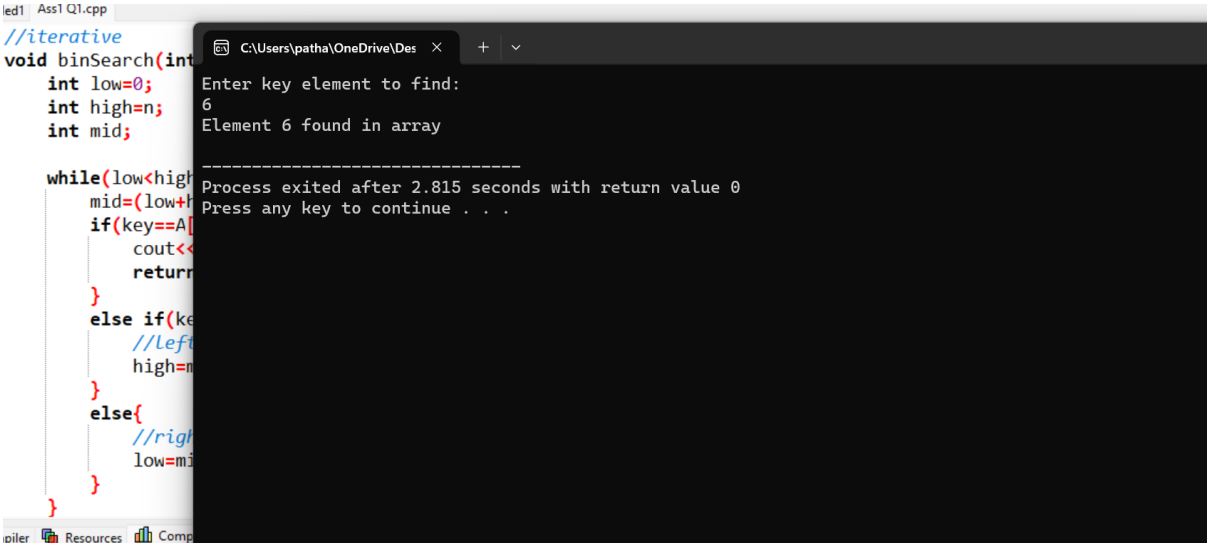
Write a program to implement iterative as well as recursive versions of the following algorithms:

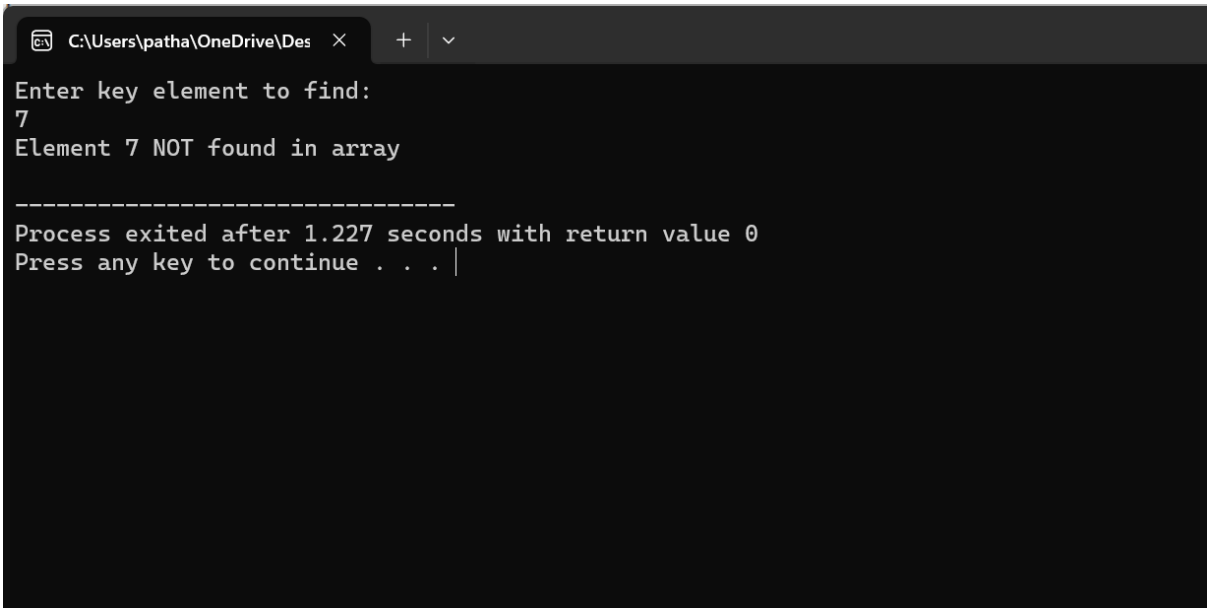
- **Binary search**

```
//Write a program to implement iterative as well as recursive versions of- Binary search
#include<iostream>
using namespace std;

//iterative
void binSearch(int A[],int n,int key){
    int low=0;
    int high=n-1;
    int mid;

    while(low<=high){
        mid=(low+high)/2;
        if(key==A[mid]){
            cout<<"Element "<<key<<" found in array \n";
            return;
        }
        else if(key<A[mid]){
            //left
            high=mid-1;
        }
        else{
            //right
            low=mid+1;
        }
    }

    cout<<"Element "<<key<<" NOT found in array \n";
}

//recursive
int rbinSearch(int A[],int n,int low,int high,int key){
    int mid;

    if(low<=high){
        mid=(low+high)/2;
        if(key==A[mid]){
```

```
                return 1;

        }
        else if(key<A[mid]){
                //left
                return rbinSearch(A,n,low,mid-1,key);
        }
        else{
                return rbinSearch(A,n,mid+1,high,key);
        }
    }
}

int main()
{
    int A[]={2,4,6,8,10};
    int n=5;

    int key;
    cout<<"Enter key element to find: \n";
    cin>>key;

//  binSearch(A,n,key);


    int res=rbinSearch(A,n,0,n-1,key);
    if(res==1){
        cout<<"Element "<<key<<" found in array \n";
    }
    else{
        cout<<"Element "<<key<<" NOT found in array \n";
    }

    return 0;
}
```

Output:


ed1 Ass1 Q1.cpp
//iterative
void binSearch(int
int low=0;
int high=n;
int mid;
while(low<high
mid=(low+h
if(key==A[
cout<<
return
}
else if(ke
//left
high=m
}
else{
//righ
low=mi
}
}
piler Resources Comp
C:\Users\patha\OneDrive\Des
Enter key element to find:
6
Element 6 found in array
Process exited after 2.815 seconds with return value 0
Press any key to continue . . .


C:\Users\patha\OneDrive\Des
Enter key element to find:
7
Element 7 NOT found in array
Process exited after 1.227 seconds with return value 0
Press any key to continue . . .

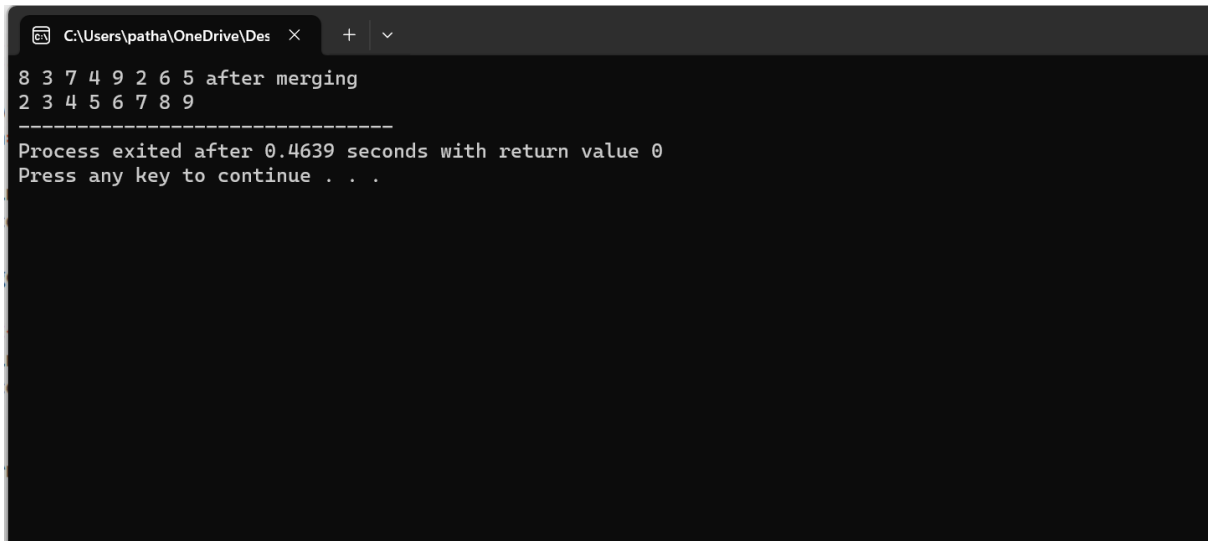
- **Merge sort**

```
//Write a program to implement iterative as well as recursive versions of- Merge sort
#include<iostream>
using namespace std;

void merge(int A[],int low,int mid,int high){
	int i,j,k;
	i=low;
	j=mid+1;
	k=low;

	int B[100];

	while(i<=mid && j<=high){
		if(A[i]<A[j]){
			B[k++]=A[i++];
		}
		else{
			B[k++]=A[j++];
		}
	}

	//remaining elements
	for(;i<=mid;i++){
		B[k++]=A[i];
	}
	for(;j<=high;j++){
		B[k++]=A[j];
	}

	//copy back
```

```
        for(i=low;i<=high;i++){
                A[i]=B[i];
        }
}

//iterative
void iMergeSort(int A[],int n)
{
        int pass;
        int low,mid,high;
        for(pass=2;pass<=n;pass=pass*2){
                for(int i=0;i+pass-1<n;i=i+pass){
                        low=i;
                        high=i+pass-1;
                        mid=(low+high)/2;
                        merge(A,low,mid,high);
                }
        }
        if(pass/2<n){
                merge(A,0,pass/2,n-1);
        }
}

//recursive
void rMergeSort(int A[],int low,int high){
        if(low<high){
                int mid=(low+high)/2;
                rMergeSort(A,low,mid);
                rMergeSort(A,mid+1,high);
                merge(A,low,mid,high);
        }
```

```
}


int main()
{
	int A[]={8,3,7,4,9,2,6,5};
	int n=8;

	for(int i=0;i<n;i++){
		cout<<A[i]<<" ";
	}
//	iMergeSort(A,n);
	rMergeSort(A,0,n-1);

	cout<<"after merging \n";
	for(int i=0;i<n;i++){
		cout<<A[i]<<" ";
	}

	return 0;
}
```

OUTPUT:



- **Quick sort**

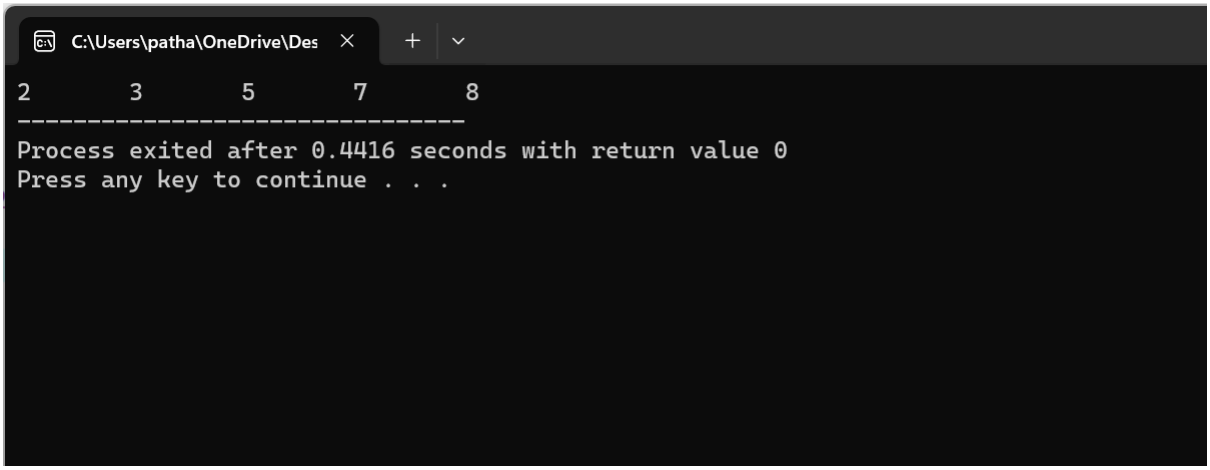
//Write a program to implement iterative as well as recursive versions of- Quick sort

```
#include<iostream>
using namespace std;

void swap(int *x,int *y){
        int temp;
        temp=*x;
        *x=*y;
        *y=temp;
}

int partition(int A[], int l, int h){
        int pivot=A[l];                 //first element must be pivot element

        int i=l;
        int j=h;

        do{
                do{
```

```
                i++;
            }while(A[i]<=pivot);

            do{
                j--;
            }while(A[j]>pivot);

            if(i<j){
                swap(&A[i],&A[j]);
            }

        }while(i<j);

        //if i>j then swap pivot with j
        swap(&A[l],&A[j]);

        return j;                   //j gives partioning position
}

void quickSort(int A[], int l, int h){
        if(l<h){
            int j;
            j=partition(A,l,h);
            quickSort(A,l,j);
            quickSort(A,j+1,h);
        }

}

int main()
{
```

```
    int n=5;
    int A[]={8,5,7,3,2};
    quickSort(A,0,n);

    for(int i=0; i<n; i++){
        cout<<A[i]<<"    ";
    }


    return 0;
}
```

Output:



- **Maximum sub-array sum**

//Write a program to implement iterative as well as recursive versions of- Maximum sub-array sum

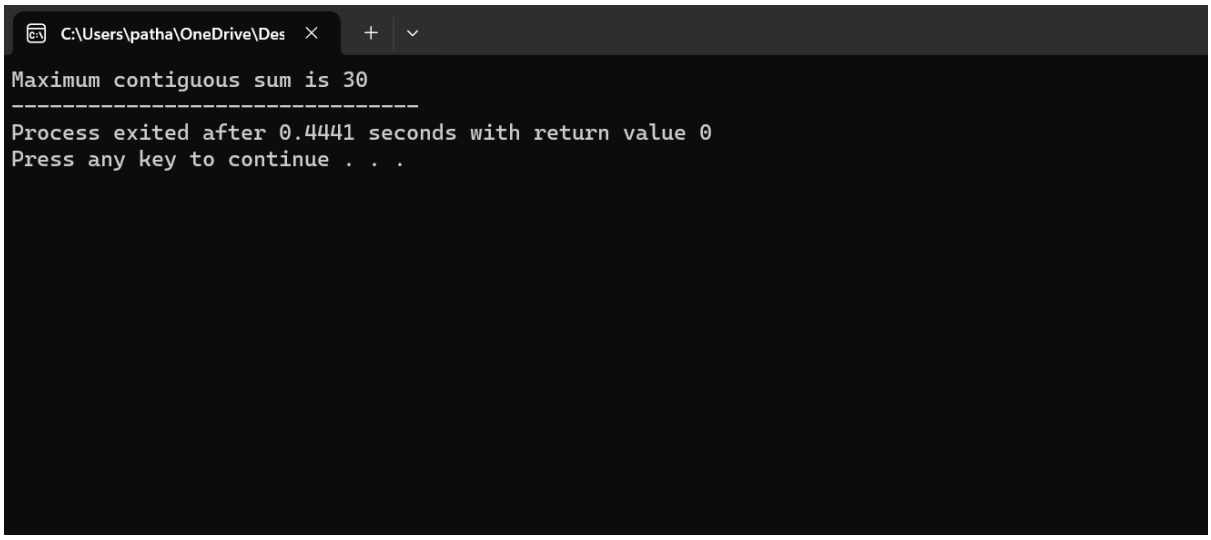
```
#include<iostream>
using namespace std;

//find maximum of two integers
int max(int a, int b) {
    if(a>b){
        return a;
```

```
	}
	else{
		return b;
	}
}

//find maximum of three integers
int max(int a, int b, int c) {
	return max(max(a, b), c);
}

//Find the maximum possible sum in arr[] such that arr[m] is part of it
int maxCrossingSum(int arr[], int l, int m, int h){
	int sum=0;
	int left_sum=INT_MIN;
	for (int i=m; i>=l; i--) {
		sum=sum+arr[i];
		if(sum>left_sum)
			left_sum = sum;
	}
	int right_sum=INT_MIN;
	for (int i=m; i<=h; i++) {
		sum=sum+arr[i];
		if(sum>right_sum)
			right_sum = sum;
	}

	// Return sum of elements on left and right of mid returning only left_sum + right_sum
	return max(left_sum+right_sum-arr[m], left_sum, right_sum);
}
```

```
// Returns sum of maximum sum subarray
int maxSubArraySum(int arr[], int l, int h) {
	// Invalid Range: low is greater than high
	if (l > h)
		return INT_MIN;
	// Base Case: Only one element
	if (l == h)
		return arr[l];

	// Find middle point
	int m = (l + h) / 2;

	/* Return maximum of following three possible cases
			a) Maximum subarray sum in left half
			b) Maximum subarray sum in right half
			c) Maximum subarray sum such that the subarray
	crosses the midpoint */
	return max(maxSubArraySum(arr, l, m - 1),
			maxSubArraySum(arr, m + 1, h),
			maxCrossingSum(arr, l, m, h));
}


int main()
{
	int arr[] = { 2, 3, 4, 5, 7 };
	int n = sizeof(arr) / sizeof(arr[0]);
	int max_sum = maxSubArraySum(arr, 0, n - 1);
	cout << "Maximum contiguous sum is " << max_sum;

	return 0;
```

}

OUTPUT:


C:\Users\patha\OneDrive\Des
Maximum contiguous sum is 30

Process exited after 0.4441 seconds with return value 0
Press any key to continue . . .