**CHAROTAR UNIVERSITY OF SCIENCE & TECHNOLOGY**

**FACULTY OF TECHNOLOGY AND ENGINEERING**

Department of Computer Engineering

**Subject Name**: Data Structures & Algorithm **Semester**: IV

**Subject Code**: CE245 **Academic year**: 2019-20

**PRACTICAL 1**

**AIM:**

Write a program to store roll numbers of student in array who attended training program in random order. Write function for

a) Searching whether particular student attended training program or not using linear search and sentinel search.

b) Searching whether particular student attended training program or not using binary search.

**PROGRAM:**

**Linear Search:**

#include<stdio.h>

#include<conio.h>

void main()

{

int arr[10];

int i;

printf("Enter an array: ");

for(i=0;i<5;i++)

{

scanf("%d",&arr[i]);

}

printf("Array is: ");

for(i=0;i<5;i++)

{

printf("%d ",arr[i]);

}

int j;

printf("\n Enter Number to find: ");

scanf("%d",&j);

int count=0;

for( i=0;i<5;i++)

{

if(j==arr[i])

{

count++;

}

}

if(count==0)

{

printf("\nNumber not found");

}

else

{

printf("\nitem found");

}

}

**Sentinel Search:**

#include<stdio.h>

#include<conio.h>

void main()

{

int arr[5];

int i;

printf("Enter an array: ");

for(i=0;i<5;i++)

{

scanf("%d",&arr[i]);

}

for(i=0;i<5;i++)

{

printf("%d ",arr[i]);

}

int last=arr[4];

int j;

printf("\nEnter Number to find: ");

scanf("%d",&j);

arr[4]=j;

int count=0;

while(arr[i]=!arr[4])

{

i++;

}

if(i<4 || last==arr[4])

{

printf("Number found");

}

else

{

printf("Number not found");

}

}

**B)**

**Binary Search:**

#include<stdio.h>

#include<conio.h>

void main()

{

int arr[10];

int i;

printf("Enter an array");

for(i=0;i<10;i++)

{

scanf("%d",&arr[i]);

}

for(i=0;i<10;i++)

{

printf("%d",arr[i]);

}

int mid;

int lb=0;

int ub=9;

int num;

printf("\n Enter no. to find");

scanf("%d",&num);

while(lb<=ub)

{

mid=(lb+ub)/2;

if(arr[mid]==num)

{

printf("\n item matched");

break;

}

else if(num<arr[mid])

{

ub=mid-1;

}

else

{

lb=mid+1;

}

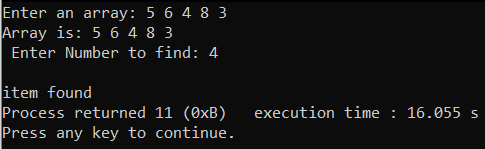
printf("\n item not matched");

}

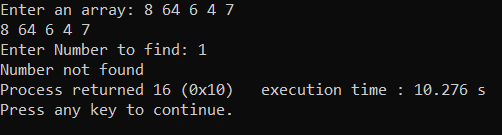
}

**OUTPUT:**

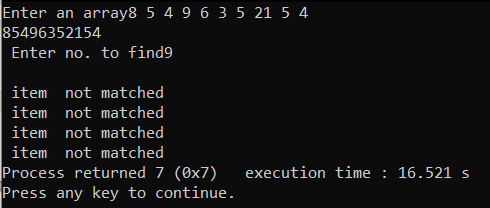
**Linear Search:**



**Sentinel Search:**



**Binary Search:**



**CONCLUSION:**

In this practical we learned about types of search

(1)linear search or sentinel search

(2)binary search.

**PRACTICAL 2**

**AIM:**

Mark purchased Books from books store of standard 1 to 7. He purchased 4 books for each standard(for std.1 books are 1.1,1.2,1.3,1.4 and for std. 2 books are 2.1,2.2,2.3,2.4 and so on..). When he reached home, he opens the bag and sees that all the books got mixed. So, how he will sort all the books, according to the standards and theirpreference in that particular standard. (ex. : preference in std. 1 is 1.1<1.2<1.3

2.1: SELECTION SORT that arranges in descending order

2.2: INSERTION SORT that arranges in ascending order

**PROGRAM:**

**Selection Sort:**

#include<stdio.h>

#include<conio.h>

void main()

{

double arr[]={1.1,1.2,1.3,1.4,2.1,2.2,2.3,2.4,3.1,3.2,3.3,3.4,4.1,4.2,4.3,4.4,5.1,5.2,5.3,5.4,6.1,6.2,6.3,6.4,7.1,7.2,7.3,7.4};

int i,j,k,min;

double temp;

for (i=0;i<27;i++)

{

min=i;

for(j=i+1;j<28;j++)

{

if(arr[min]>arr[j])

{

min=j;

}

}

temp=arr[i];

arr[i]=arr[min];

arr[min]=temp;

}

for(k=0;k<28;k++)

{

printf("%f ",arr[k]);

printf(" ");

}

}

**Insertion Sort:**

#include<stdio.h>

#include<conio.h>

void main()

{

double arr[]={1.1,1.2,1.3,1.4,2.1,2.2,2.3,2.4,3.1,3.2,3.3,3.4,4.1,4.2,4.3,4.4,5.1,5.2,5.3,5.4,6.1,6.2,6.3,6.4,7.1,7.2,7.3,7.4};

int i,j,k,min;

double temp;

for (i=1;i<28;i++)

{

temp=arr[i];

j=i-1;

while(j>=0 && arr[j]>temp)

{

arr[j+1]=arr[j];

j--;

}

arr[j+1]=temp;

}

for(i=0;i<28;i++)

{

printf("%f ",arr[i]);

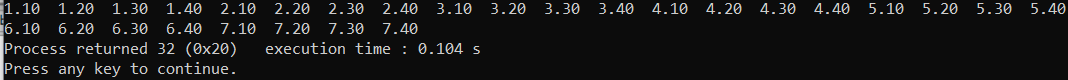
printf(" ");

}

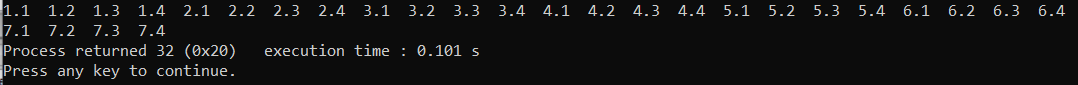
}

**OUTPUT:**

**Selection Sort:**



**Insertion Sort:**



**CONCLUSION:**

In this practical we learned about insertion sort and selection sort in ascending and descending .

**PRACTICAL 3**

**AIM:**

Implement a menu driven program that performs following sorting algorithms

3.1 QUICK SORT that arranges in ascending order

3.2 MERGE SORT that arranges in descending order

**PROGRAM:**

**Quick Sort:**

#include<stdio.h>

// A utility function to swap two elements

void swap(int\* a, int\* b)

{

int t = \*a;

\*a = \*b;

\*b = t;

}

int partition (int arr[], int low, int high)

{

int pivot = arr[high]; // pivot

int i = (low - 1); // Index of smaller element

for (int j = low; j <= high- 1; j++)

{

if (arr[j] < pivot)

{

i++; // increment index of smaller element

swap(&arr[i], &arr[j]);

}

}

swap(&arr[i + 1], &arr[high]);

return (i + 1);

}

/\* The main function that implements QuickSort

arr[] --> Array to be sorted,

low --> Starting index,

high --> Ending index \*/

void quickSort(int arr[], int low, int high)

{

if (low < high)

{

/\* pi is partitioning index, arr[p] is now

at right place \*/

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

void printArray(int arr[], int size)

{

int i;

for (i=0; i < size; i++)

printf("%d ", arr[i]);

//printf("n");

}

// Driver program to test above functions

int main()

{

int arr[] = {10, 7, 8, 9, 1, 5};

int n = sizeof(arr)/sizeof(arr[0]);

quickSort(arr, 0, n-1);

printf("Sorted array: ");

printArray(arr, n);

return 0;

}

**Merge Sort:**

#include<stdlib.h>

#include<stdio.h>

void merge(int arr[], int l, int m, int r)

{

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

/\* create temp arrays \*/

int L[n1], R[n2];

/\* Copy data to temp arrays L[] and R[] \*/

for (i = 0; i < n1; i++)

L[i] = arr[l + i];

for (j = 0; j < n2; j++)

R[j] = arr[m + 1+ j];

i = 0; // Initial index of first subarray

j = 0; // Initial index of second subarray

k = l; // Initial index of merged subarray

while (i < n1 && j < n2)

{

if (L[i] <= R[j])

{

arr[k] = L[i];

i++;

}

else

{

arr[k] = R[j];

j++;

}

k++;

}

/\* Copy the remaining elements of L[], if there

are any \*/

while (i < n1)

{

arr[k] = L[i];

i++;

k++;

}

while (j < n2)

{

arr[k] = R[j];

j++;

k++;

}

}

void mergeSort(int arr[], int l, int r)

{

if (l < r)

{

// Same as (l+r)/2, but avoids overflow for

// large l and h

int m = l+(r-l)/2;

mergeSort(arr, l, m);

mergeSort(arr, m+1, r);

merge(arr, l, m, r);

}

}

void printArray(int A[], int size)

{

int i;

for (i=0; i < size; i++)

printf("%d ", A[i]);

printf("\n");

}

int main()

{

int arr[] = {12, 11, 13, 5, 6, 7};

int arr\_size = sizeof(arr)/sizeof(arr[0]);

printf("Given array is \n");

printArray(arr, arr\_size)

mergeSort(arr, 0, arr\_size - 1);

printf("\nSorted array is \n");

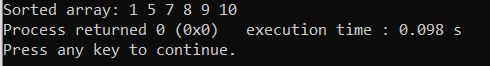
printArray(arr, arr\_size);

return 0;

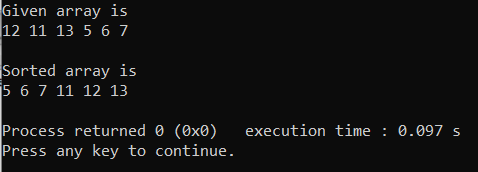
}

**OUTPUT:**

**Quick Sort:**



**Merge Sort:**



**CONCLUSION:**

In this practical we learned about marge sort and quick sort .

**PRACTICAL 4**

**AIM:**

Perform following programs using Stack data structure:

4.1 Sometimes a program requires two stack containing the same type of items. If the two stacks are stored in separate arrays. Then one stack might overflow while there was considerable unused space in the other. A neat way to avoid the problem is to put all the space in one array and let one stack grow from one end of the array and the other stack start at the other end and grow in opposite direction i.e., toward the first stack, in this way, if one stack turns out to be large and the other small, then they will still both fit, and there will be no overflow until all the space is actually used. Declare a new structure type Double stack that includes the array and the two indices top A and top B, and write functions Push A, Push B, Pop A and Pop B to handle the two stacks with in one Double Stack.

4.2 Implement Tower of Hanoi Problem using Recursion.

**PROGRAM:  
4.1**

#include<iostream>

#include<stdlib.h>

using namespace std;

class twoStacks

{

int \*arr;

int size;

int top1, top2;

public:

twoStacks(int n) // constructor

{

size = n;

arr = new int[n];

top1 = -1;

top2 = size;

}

void push1(int x)

{

if (top1 < top2 - 1)

{

top1++;

arr[top1] = x;

}

else

{

cout << "Stack Overflow";

exit(1);

}

}

void push2(int x)

{

if (top1 < top2 - 1)

{

top2--;

arr[top2] = x;

}

else

{

cout << "Stack Overflow";

exit(1);

}

}

int pop1()

{

if (top1 >= 0 )

{

int x = arr[top1];

top1--;

return x;

}

else

{

cout << "Stack UnderFlow";

exit(1);

}

}

int pop2()

{

if (top2 < size)

{

int x = arr[top2];

top2++;

return x;

}

else

{

cout << "Stack UnderFlow";

exit(1);

}

}

};

int main()

{

twoStacks ts(5);

ts.push1(5);

ts.push2(10);

ts.push2(15);

ts.push1(11);

ts.push2(7);

cout << "Popped element from stack1 is " << ts.pop1();

ts.push2(40);

cout << "\nPopped element from stack2 is " << ts.pop2();

return 0;

}

**4.2:**

#include<stdio.h>

#include<conio.h>

void towerOfHanoi(int n, char from\_rod, char to\_rod, char aux\_rod)

{

if (n == 1)

{

printf("\n Move disk 1 from rod %c to rod %c", from\_rod, to\_rod);

return;

}

towerOfHanoi(n-1, from\_rod, aux\_rod, to\_rod);

printf("\n Move disk %d from rod %c to rod %c", n, from\_rod, to\_rod);

towerOfHanoi(n-1, aux\_rod, to\_rod, from\_rod);

}

int main()

{

int n = 4; // Number of disks

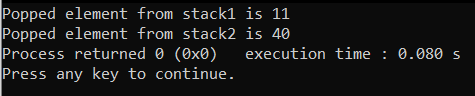
towerOfHanoi(n, 'A', 'C', 'B');

return 0;

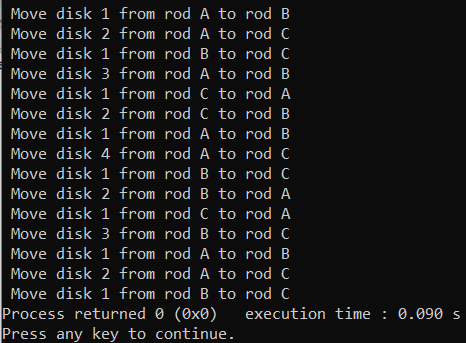
}

**OUTPUT:**

**4.1**



**4.2**



**CONCLUSION:**

In this practical we learned about stack and tower of Hanoi using recursion.

**PRACTICAL 5**

**AIM:**

We are developing software for a call center. When a client calls, his/her call should be

stored until there is a free service representative to pick the call. Calls should be

processed in the same order they are received. Select appropriate data structure to build

call center software system.

**PROGRAM:**

#include<iostream>

using namespace std;

int main()

{

int arr1[10];

static int max1=10,rear=-1,front1=-1,ele,num,i;

//cout<<sizeof(arr1)/sizeof(int);

cout<<"The total size of the array is 10!!";

while(1)

{

cout<<"\nEnter (1 to INSERT, 0 to DELETE , 9 to EXIT ) :";

cin>>num;

cout<<"======================================================";

if(num==1)

{

if(rear>max1)

{

cout<<"\nThe QUEUE IS FULL!";

cout<<"\n=======================================================";

}

else

{

rear=rear+1;

cout<<"\nEnter the Element you want to enter:";

cin>>ele;

arr1[rear]=ele;

front1++;

}

}

else if(num==0)

{

if(front1>=0)

{

ele=arr1[front1];

arr1[front1]=arr1[front1 + 1];

cout<<"\nDELETED item is:"<<ele;

cout<<"\n======================================================";

}

else

{

cout<<"\nTHERE IS NO ELEMENT TO DELETE!!";

cout<<"\n======================================================";

}

}

else if(num==9)

{

break;

}

else

{

for (i=front1+1;i<rear;i++)

{

cout<<"\n";

cout<<" "<<arr1[i];

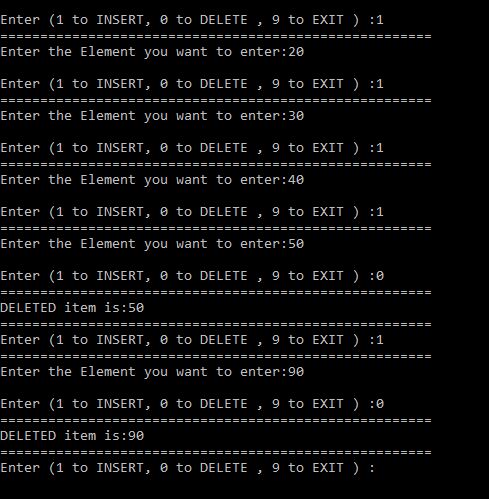
}

}

}

}

**OUTPUT:**

****

**CONCLUSION:**

In this practical we came to know that in queue it works on the principle of ’FIRST IN FIRST OUT’