Name-Harshita Totala Roll No- SEITA14

Div- A

Assignment-1
Searching and Sorting

AIM:

Consider a student database of SEIT class (at least 15 records). Database contains different fields of every student like Roll No, Name and SGPA.(array of structure)

a) Design a roll call list, arrange list of students according to roll numbers in ascending order (Use Bubble Sort)

b) Arrange list of students alphabetically. (Use Insertion sort)

c) Arrange list of students to find out first ten toppers from a class. (Use Quick sort)

d) Search students according to SGPA. If more than one student having same SGPA, then print list of all students having same SGPA.

e) Search a particular student according to name using binary search without recursion. (all the student records having the presence of search key should be displayed**)**

THEORY:

Introduction Sorting:

Sorting means bringing some orderliness in the data, which are otherwise not ordered. For example, to print the list of students in ascending order of their birth dates, we need to sort the student information in ascending order of date of birth (student information stored in a file may not be in this order) before generating the print report. In most of the computer applications we need to sort the data either in ascending or descending order as per the requirements of the application. Sorting techniques are broadly classified as internal sort techniques and external sort techniques.

Internal sort: Internal sort is also known as Memory Sort, and it is used to sort the small volume of data in computer memory. Following are the some of internal or memory sort techniques.

1. Bubble sort
2. Selection sort
3. Insertion sort
4. Bucket sort
5. Radix sort
6. Quick sort
7. Merge sort

Name-Harshita Totala Roll No- SEITA14

Div- A

External sort:

The term External sort is referred to sorting of voluminous data. For example, to create a temporary file sorted on some key from the input file containing thousands of records, we need to divide the input file in small partitions, which can be sorted on the key by using any one of the known internal sort techniques and then merging those partitions on the sorted key to generate new partition of bigger size and this process is repeated until we get only one large sorted partition of size equal to the number of records in input file. In practice most of the times it is not possible to bring the entire data from input file to the memory of computer for sorting and we have to use some secondary storage to store the intermediate results (partitions). Thus in external sort input and output sorted files are stored on secondary storage i.e. disk.

Bubble sort:

Basic step:

From the remaining unsorted list, compare the first element with other elements and interchange them if they are not in the required order.

For the list containing 'n' elements,

Number of passes : n - 1

Time complexity : $O(n^2)$

Space complexity : No extra space required

Worst case : List in reverse order

Comparisons : $O(n^2)$ Exchanges : $O(n^2)$

Best case : Already sorted list

Comparisons: $O(n^2)$, Exchanges: 0

Average case : List on random order

Comparisons : $O(n^2)$, Exchanges : $O(n^2)$

Comments : Good for small value of 'n'

Name-Harshita Totala Roll No- SEITA14

Div- A

ALGORITHM:

```
 begin BubbleSort(list)

   for all elements of list       if list[i] > list[i+1]
swap(list[i], list[i+1])       end if    end for

   return list

end BubbleSort
```

Insertion sort:

Basic step:

To insert a record R into sequence of ordered records R1, R2, …, Ri, in such a way that resulting sequence of size (i+1) is also ordered.

For the list containing 'n' elements,

Number of passes : 1

Time complexity : O(n*n)

Space complexity : No extra space required

Worst case : List in reverse order
Comparisons : O(n*n), Push downs : O(n*n)

Best case : Already sorted list
Comparisons : O(n), Push downs : 0

Average case : List on random order
Comparisons : O(n*n), Push downs : O(n*n)

Comments : Good if the input list has very few entries Left Out of Order (LOO)

ALGORITHM:

Step 1 – If it is the first element, it is already sorted. return

Name-Harshita Totala Roll No- SEITA14

Div- A

```
1;
Step 2 – Pick next element
Step 3 – Compare with all elements in the sorted sub-list Step
4 – Shift all the elements in the sorted sub-list that is
greater than the          value to be sorted
Step 5 – Insert the value
Step 6 – Repeat until list is sorted
```

QUICK SORT : Quick sort scheme developed by C.A.R. Hoare, has the best average behaviour in terms of Time & Space Complexity among all the internal sorting methods.

BASIC STEP : In Quick Sort the key Ki (pivot – generally the first element in an array) controlling the process is placed at the right position with respect to the elements in an array i.e. if key Ki is placed at position s(i) then

$$Kj < Ks(i) \text{ for } j < s(i) \text{ and}$$

$$Kj >= Ks(i) \text{ for } j > s(i)$$

Thus after positioning of the element Ki has been made, the original array is partitioned into two sub arrays, one consisting of elements K1 to (Ks(i) – 1) and the other partition (Ks(i) + 1) to Kn. All the elements in first sub array are less than Ks(i) & those in the second sub array are greater or equal to Ks(i). These two sub arrays can be sorted independently

DIVIDE & CONQUER STRATEGY :

From the above description of Quick Sort, it is evident that Quick Sort is a good example of **Divide & Conquer** strategy for solving the problem. In Quick Sort, we divide the array of items to be sorted into two partitions and then call the same procedure recursively to sort the two partitions. Thus we **divide** the problem in two smaller problems and **conquer** by solving the smallest problem, in this case sorting of partition containing only one element at the end, which is trivial one. Thus Quick Sort is the best example of divide & conquer strategy, where the problem is solved (conquered) by dividing the original problem into two smaller problems and then applying the same strategy recursively until the problem is so small that it can be solved trivially. Thus the conquer part of the Quick Sort looks like this.

Name-Harshita Totala Roll No- SEITA14

Div- A

Initial Step

< Pivot1	Pivot1	>= Pivot1

Next Step

< Pivot2	Pivot2	>= Pivot2	Pivot1	>= Pivot1

EXAMPLE :

Pass No : **< -------- Input Array Elements -------- > Pivot m n i j**

Position : 0 1 2 3 4 5 6 7 8 9

Pass 1 : **2 1 22 34 4 7 30 1 21 20** 2 0 9 1 9

Pass 2 : **1 1** 2 34 4 7 30 22 21 20 1 0 1 1 1

Pass 3 : 1 1 2 **34 4 7 30 22 21 20** 34 3 9 4 9

Pass 4 : 1 1 2 **20 4 7 30 22 21** 34 20 3 8 4 8

Pass 5 : 1 1 2 **7 4** 20 30 22 21 34 7 3 4 4 4

Pass 6 : 1 1 2 4 7 20 **30 22 21** 34 30 6 8 7 8

Name-Harshita Totala Roll No- SEITA14

Div- A

Pass 7 : 1 1 2 4 7 20 **21 22** 30 34 30 6 7 7 7

Sorted Array : 1 1 2 4 7 20 21 22 30 34

1) Time Complexity :

a) Worst Case : Already sorted array is the worst case for Quick sort. Let us assume that there are 'n' elements in an array, then the number of elements in each partition for each pass will be,

Pass No	***Partitions***	***No. of comparisions***
Pass 1	: [0], [n - 1]	(n - 1)
Pass 2	: [0], [0], [n - 2]	(n - 2)
Pass 3	: [0],.[0], [0], [n - 3]	(n - 3)
.		.
.		.
.		.
Pass (n-1)	: [0],.[0], [0], …[0],[1]	1

Total number of comparisons would be = (n - 1) + (n - 2) + (n - 3) + …+ 1

= (n - 1) (n) / 2

= (n**2 / 2) – (n / 2)

= O (n**2) (ignoring lower order terms)

b) Best Case : Best case of Quick Sort would be the case where input elements in the array are such that in every pass **pivot** element is positioned at the middle such that the partition is divided in two partitions of equal size Then for a large value of n, the number of comparisons in each pass would be,

After 1st Pass : n comparisons + number of comparisons required to sort two

Partitions of size n / 2.

= n + 2 * [n / 2]

After 2nd Pass = n + 2 * [n / 2 + 2 * (n / 4)]

= n + n + 4*[n / 4]

= 2n + 4*[n / 4]

After 3rd Pass = 2n + 4* [n / 4 + 2 * (n / 8)]

= 2n + n + 8*[n / 8]

= 3n + 8*[n / 8] = (log 8) * 8 + 8*[8/8] for 8 elements array

= **8*log8** since number of comparisons required to sort partition of size 1 would be zero.

. . .

. . .

. . .

After nth Pass = n*log(n)

c) Average case : Experimental results show that number of comparisons required to sort 'n' element array using Quick Sort are **O(nlogn)** to an average.

2) Space Complexity : In recursive quick sort, additional space for stack is required, which is of the order of **O(logn)** for **best and average case** and **O(n)** for **worst case.**

ALGORITHM:

```
void quicksort (int a[], int p, int q)
/* sorts the elements a[p]…,a[q] which reside in the global array a[1:n] into ascending
order.*/ {
1. declare int j;
2. if(left >right)
   return;
```

3. j=partition(a, left ,right);
4. quicksort(a, left,j-1);
5. quicksort(a, j+1,right);

}

int part(int a[], int left ,int right)

/* within the array a[left: right] elements are arranged such that if pivot p is stored at location q then all elements less than p are stored from left to q-1 and all elements greater than p are stored from q+1 to right. */

{

1. **declare int i,j,temp,v;**
2. **pivot=a[left]; i=left; j=right+1;**
3. **do**

```
{
    do
{
i++;
    } while(a[i]<pivot);
do
    { j--;
    } while(a[j]>pivot);
if(i<j)
        {
          temp=a[i];
a[i]=a[j];          a[j]=temp;
        }
    }while(i<j);
```

4. **a[left]=a[j];**
5. **a[j]=pivot;**
6. **// Display intermediate results**
   Print partition position as j
   Print array

7. **return(j);**

}

Searching Techniques:

1. Sequential or Serial or Linear Search
2. Binary Search
3. Fibonacci Search

Sequential or Serial or Linear search:

In this method, entities are searched one by one starting from the first to the end. Searching stops as soon as we reach to the required entity or we reach to the end. For example, consider the array containing 'n' integers and to search a given number in array we start searching from the 1st element of the array to the end of array. Searching will stop as soon as the element of the array is equal to the given number (number found) or we reach the end of the array (number not found). This method is generally used when the **data stored is not in the order of the key** on which we want to search the table.

Binary Search:

If the **array is sorted on the key** & we want to search for a key having given value, then **Binary search** technique can be used which is more efficient as compared to serial search. Binary search uses the '**divide & conquer**' strategy as given below: 1) Divide the list into two equal halves.

2) Compare the given key value with the middle element of the array or list.
   There are three possibilities.
   a) middle element = key : key found and search terminates
   b) middle element > key : the value which we are searching is possibly in the first half of the list
   c) middle element < key : the value which we are searching is possibly in the second half of the list
3) Now search the key either in first half of second half of the list (b or c) 4) Repeat steps 2 and 3 till key is found or search fails in case key does not exist.

Binary Search:

a) Non recursive int bin_search_nonrec (char a[][20] , int

n, char key[20])

/* this procedure searches a record of given name from set of names and returns the record no to the calling program */ {

Name-Harshita Totala Roll No- SEITA14

Div- A

1) **Declare int** first, last, middle, passcnt=0;
2) **initialize** first=1, last =cnt;
3) **while** (last >= first)

```
{
    middle= ( first  + last )/2        if
( strcmp(key, a[middle] ) >0)
            first = middle + 1;
    else if (( strcmp(key, a[middle]) < 0)
            last = middle – 1;
    else
        return ( middle );
}
```

4) **return**(-1); //not found

}//end bin_search_nonrec

Name-Harshita Totala Roll No- SEITA14

Div- A

PROGRAM-

```
#include<iostream>
#include<string.h>
using namespace std;

typedef struct Students{
    int roll_no;
    char name[20];
    float marks;
}stu;                                   //stu array of structure
student.

void input(stu s[20], int n) {
    int i;
    for (i = 0; i < n; i++){
        cout<<"\nStudent "<<i+1<<" Information ";
        cout<<"\nEnter Name of student : ";
        cin>>s[i].name;
        cout<<"Enter student Roll Number : ";
        cin>>s[i].roll_no;

       int a;
        for(a=0;a<i;a++)
        {
           if(s[i].roll_no==s[a].roll_no)               //to check
same roll-no entered or not
            {
            cout<<"Please enter a unique roll no- ";
            cin>>s[i].roll_no;
            break;
            }
        }
         cout<<"Enter SGPA of student : ";
        cin>>s[i].marks;
    }
    i--;

}
```

Name-Harshita Totala Roll No- SEITA14

Div- A

```
void display(stu s[20], int n){
    cout<<"\n\t "<<"NAME"<<"\t      "<<"ROLL NO."<<"\t\t"<<"SGPA";
    cout<<"\n";
    cout<<"\t----------------------------------------";
    for (int i = 0 ; i < n ; i++)
    {
        cout<<"\n";
        cout<<"\t "<<s[i].name<<"\t\t
"<<s[i].roll_no<<"\t\t"<<s[i].marks<<endl;
    }
}


//bubble sort to sort roll number in ascending order.
void bubble_sort(stu s[20], int n){
    stu temp;
    for(int i = 0 ; i < n ; i++)
    {
        for(int j = 0 ; j < n - 1  ; j++)
        {
            if (s[j].roll_no> s[j+1].roll_no)
            {
                temp = s[j];
                s[j] = s[j+1];
                s[j+1] = temp;
            }
        }
    }

}

// insertion sort to sort names in ascending order
void insertion_sort(stu s[20], int n)
{
    int i,j;
    stu k;
    for (i = 1; i < n; i++)
```

```
    {
        k= s[i];
        j = i - 1;
        while (j >= 0 && strcmp(s[j].name, k.name) >0)
        {
            s[j + 1]= s[j];
            j = j - 1;
        }
            s[j + 1]= k;
    }
}


// quick sort to sort marks in desending order.
int partition(stu s[20], int l,int h){
    int i,j;
     stu temp, pivot;

    pivot=s[l];
    i=l+1;
    j=h;

    do
    {
        while(s[i].marks > pivot.marks ){
         i++;
        }

        while(pivot.marks > s[j].marks){
        j--;
        }

        if(i<j)
        {
            temp=s[i];
            s[i]=s[j];
            s[j]=temp;
        }
    }while(i<j);
```

```
    temp=s[l];
    s[l]=s[j];
    s[j]=temp;

    return(j);
}
void quicksort(stu s[20], int l, int h){

    if(l<h){
        int p=partition(s,l,h);
        quicksort(s,l,p-1);
        quicksort(s,p+1,h);
    }
}


// linear search for marks if more than one student have same marks
print all of them.
void linear_search(stu s[20], int n, float key)
{
    int f=-1;
    cout<<"\n\t "<<"NAME"<<"\t      "<<"ROLL NO."<<"\t\t"<<"SGPA";
    cout<<"\n";
    cout<<"\t----------------------------------------";
    for(int i = 0; i < n; i++)
    {
        if (key == s[i].marks){
            cout<<"\n\t "<<s[i].name<<"\t\t
"<<s[i].roll_no<<"\t\t"<<s[i].marks<<endl;
            f=0;
        }
    }
    if(f==-1){
        cout<<"\n\t---------ENTERED MARKS NOT FOUND --------"<<endl;
    }
}
```

Name-Harshita Totala Roll No- SEITA14

Div- A

```
int binary_search(stu s[20], char x[20], int low, int high)
{
    int mid;
    while( low <= high){

       int mid = (low + high)/2;

       if(strcmp(x,s[mid].name) == 0){
           while(strcmp(x,s[mid].name)==0)
           {
               cout<<"\n\t "<<s[mid].name<<"\t\t
"<<s[mid].roll_no<<"\t\t"<<s[mid].marks<<endl;
               mid--;
           }
            while(strcmp(x,s[mid].name)==0)
           {
               cout<<"\n\t "<<s[mid].name<<"\t\t
"<<s[mid].roll_no<<"\t\t"<<s[mid].marks<<endl;
               mid++;
           }
           return 0;
       }
       else if(strcmp(x,s[mid].name) > 0){
           low=mid+1;
       }
       else{
           high=mid-1;
       }
    }
    return -1;

}



int main(){
    stu s[20];
    int ch, n , r;
    float k;
```

Name-Harshita Totala Roll No- SEITA14

Div- A

```
    char x[20];

    do{
        cout<<"\nChoice of operations are : ";
        cout<<"\n1) Create Student Database";
        cout<<"\n2) Display Records";
        cout<<"\n3) Bubble Sort (for roll number)";
        cout<<"\n4) Insertion sort (for name)";
        cout<<"\n5) Quick sort (for marks)";
        cout<<"\n6) Linear Search (for marks)";
        cout<<"\n7) Binary Search  (for name)";
        cout<<"\n8) Exit"<<endl;

        cout<<"\nEnter your choice of operation : ";
        cin>>ch;

        switch (ch){
            case 1:
                cout<<"\nEnter number of Records to be entered : ";
                cin>>n;
                input(s, n);
                cout<<endl;
                break;

            case 2:
            display(s,n);
            break;

            case 3:
                bubble_sort(s, n);
                display(s,n);
                break;


            case 4:
                insertion_sort(s, n);
                display(s,n);
                break;
```

```
            case 5:
                quicksort(s,0,n-1);
                display(s,n);
                cout<<endl;
                break;


            case 6:
                cout<<"Enter the marks you want to search- ";
                cin>>k;
                linear_search(s, n ,k);
                break;


            case 7:
            cout<<"Enter name you want to search- ";
            cin>>x;
            insertion_sort(s, n);
            r=binary_search(s,x,0,(n-1));
            if(r==-1)
            {
                cout<<"\n----STUDENT NAME NOT PRESENT !----"<<endl;
            }
            else
            {
                cout<<"\n----STUDENT NAME PRESENT !----"<<endl;
            }
            break;


            case 8:
                break;

            default:
                cout<<"\nINVALID CHOICE. CHOOSE AGAIN !!!!\n"<<endl;
            }

        } while (ch != 8);
}
```

Name-Harshita Totala Roll No- SEITA14

Div- A

Output-

Choice of operations are :

1) Create Student Database

2) Display Records

3) Bubble Sort (for roll number)

4) Insertion sort (for name)

5) Quick sort (for marks)

6) Linear Search (for marks)

7) Binary Search (for name)

8) Exit

Enter your choice of operation : 1

Enter number of Records to be entered : 4

Student 1 Information

Enter Name of student : Akash

Enter student Roll Number : 23

Enter SGPA of student : 6.78

Student 2 Information

Name-Harshita Totala Roll No- SEITA14

Div- A

Enter Name of student : Priya

Enter student Roll Number : 12

Enter SGPA of student : 4.56

Student 3 Information

Enter Name of student : Deep

Enter student Roll Number : 23

Please enter a unique roll no- 67

Enter SGPA of student : 8.90

Student 4 Information

Enter Name of student : Swati

Enter student Roll Number : 56

Enter SGPA of student : 7.21

Choice of operations are :

1) Create Student Database

2) Display Records

3) Bubble Sort (for roll number)

4) Insertion sort (for name)

5) Quick sort (for marks)

Name-Harshita Totala Roll No- SEITA14

Div- A

6) Linear Search (for marks)

7) Binary Search (for name)

8) Exit

Enter your choice of operation : 3

NAME	ROLL NO.	SGPA
Priya	12	4.56
Akash	23	6.78
Swati	56	7.21
Deep	67	8.9

Choice of operations are :

1) Create Student Database

2) Display Records

3) Bubble Sort (for roll number)

4) Insertion sort (for name)

5) Quick sort (for marks)

Name-Harshita Totala Roll No- SEITA14

Div- A

6) Linear Search (for marks)

7) Binary Search (for name)

8) Exit

Enter your choice of operation : 4

NAME	ROLL NO.	SGPA
Akash	23	6.78
Deep	67	8.9
Priya	12	4.56
Swati	56	7.21

Choice of operations are :

1) Create Student Database

2) Display Records

3) Bubble Sort (for roll number)

4) Insertion sort (for name)

5) Quick sort (for marks)

Name-Harshita Totala Roll No- SEITA14

Div- A

6) Linear Search (for marks)

7) Binary Search (for name)

8) Exit

Enter your choice of operation : 5

NAME	ROLL NO.	SGPA
Deep	67	8.9
Swati	56	7.21
Akash	23	6.78
Priya	12	4.56

Choice of operations are :

1) Create Student Database

2) Display Records

3) Bubble Sort (for roll number)

4) Insertion sort (for name)

Name-Harshita Totala Roll No- SEITA14

Div- A

5) Quick sort (for marks)

6) Linear Search (for marks)

7) Binary Search (for name)

8) Exit

Enter your choice of operation : 6

Enter the marks you want to search- 7.21

NAME	ROLL NO.	SGPA
Swati	56	7.21

Choice of operations are :

1) Create Student Database

2) Display Records

3) Bubble Sort (for roll number)

4) Insertion sort (for name)

5) Quick sort (for marks)

6) Linear Search (for marks)

7) Binary Search (for name)

8) Exit

Name-Harshita Totala Roll No- SEITA14

Div- A

Enter your choice of operation : 7

Enter name you want to search- Deep

Deep 67 8.9

----STUDENT NAME PRESENT !----

Choice of operations are :

1) Create Student Database

2) Display Records

3) Bubble Sort (for roll number)

4) Insertion sort (for name)

5) Quick sort (for marks)

6) Linear Search (for marks)

7) Binary Search (for name)

8) Exit

Enter your choice of operation : 8

Conclusion: Thus we had implemented bubble sort, insertion sort, quick sort and linear and binary search.