1. Searching and Sorting

TITLE:

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).

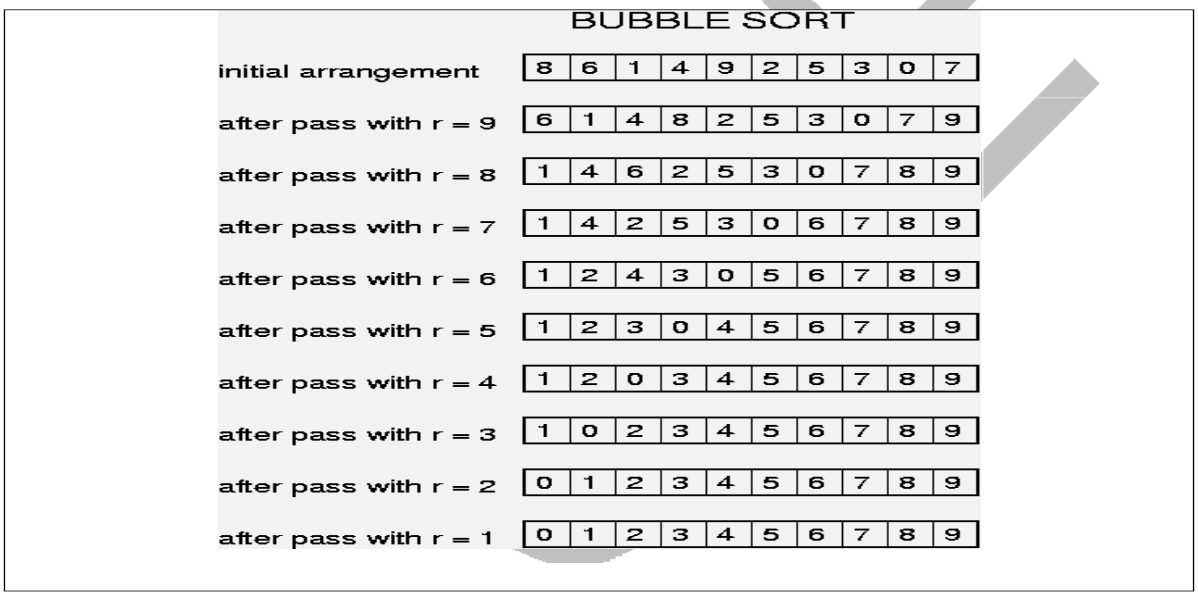
Objective:

To understand the concept of Searching and Sorting.

Theory:

1. Bubble Sort:

The basic idea underlying the bubble sort is to compare the adjacent elements and moves the largest element to the bottom of the list.



Algorithm: Bubble Sort:

Bubble Sort (a, n)

//Here 'a' is an array of stringss and 'n' represents the number of strings in the array 'a'.

Step 1: Initialize j=0;

Step 2: Repeat through step-9 while (j<n-1)

Step 3: Initialize i=j+1

Step 4: Repeat through step-8 while (i<n).

Step 5: Compare a[j] and a [i].

If (strcmp(a[j],a [i])>0)) then interchange a[j] and a [i].

Step 6: Increment the value of 'j' as i=i+1

Step 7: Increment the value of 'i' as j=j+1

Step 8: Exit

Implementation: BubbleSort:

```
void sortStrings(char arr[][MAX], int n)
{
   char temp[MAX];

   // Sorting strings using bubble sort
   for (int j=0; j<n-1; j++)
   {
      for (int i=j+1; i<n; i++)
      {
         if (strcmp(arr[j], arr[i]) > 0)
         {
            strcpy(temp, arr[j]);
            strcpy(arr[j], arr[i]);
            strcpy(arr[i], temp);
         }
      }
   }
}
void sortStrings(char arr[][MAX], int n)
```

2. Insertion Sort:

An insertion sort is on that sorts a set of records into an existing sorted file. The simple insertion sort may be viewed as a general selection sort in which priority queue is necessary.

But the selection sort & insertion sort are the more efficient than bubble sort. Selection sort requires fewer assignment than insertion sort but more comparisons. Consider for example in insertion sort initially if second elements is smaller than the first then contents of first and second elements are swapped and again so on until all the numbers in an array are properly sorted. E.g. 2 9 45 39 94 82 11 108here 9 is compared with 2 then they are sorted in proper order and again 5 is compared with 39.

Algorithm: - for function insertion sort (a)

Step 1: Start

Step 2: for i=0 to 10

Step 3: l=a[i] and j=i

Step 4: If j > 0 and a [j-1] > l then goto step then goto step 4a, 4b, 4c

4a: a[j] = a[j-1] and j-j-1a [j]

= l

Step 5: Print "pass" I + 1 th

For k = 0 to 10Print

a[k]

Step 6: End for loop

Step 7: Stop.

Implementation: insertion Sort:

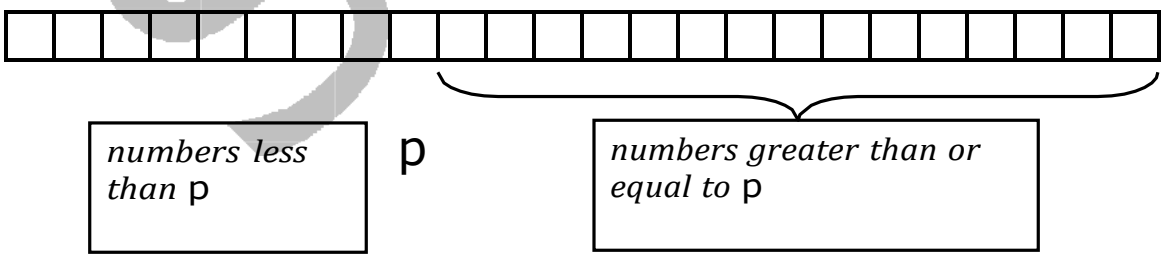
```
insertion( int a[], int n)
{
  for(k=1;k<n;k++)
 {
   y=a[k];
    For(i=k-1;i>=0 && y<a[i]; i--)
     {
        a[i+1]=a[i];
        a[i]=y;
      }
  }
Print a array
}  // end of function.
```

3. Quick sort:

Quick Sort also known as partition exchange sort. Original algorithm was developed by C.A.R. Hoare in 1962. Quicksort sorts by employing a divide and conquer strategy to divide a list into two sub-lists. It is one of the fastest sorting algorithms available. QuickSort is especially convenient with large arrays that contain elements in random order.

The steps are:

1. Pick an element, called a pivot, from the list.
2. Reorder the list so that all elements which are less than the pivot, come before the pivot and all elements greater than the pivot come, after it (equal values can go either way). After this partitioning, the pivot is in its final position. This is called the partition operation.
3. Recursively sort the sub-list of smaller elements and the sub-list of greater elements.



Thus quicksort is an recursive sorting algorithm which chooses an element of the list, called the pivot element, and then rearranges the list so that all of the elements smaller than the pivot are moved before it and all of the elements larger than the pivot are moved after it.

Advantage:

- Efficient sorting technique.
- Time complexity is O(nlogn).

Disadvantage:

- Worst case time complexity is $O(n^2)$.
- Time requirement depends on the position of the pivot in the list.

Analysis of Quicksort:

Time requirement of Quicksort depends on the position of pivot in the list, how pivot is dividing list into sublists. It may be equal division of list or may be it will not divide also.

Best case:

In Best case, we assume that list is equally divided means, list1 is equally divided in two sublists, these two sublists in four sublists, and so on. So the total no. of elements at particular level (l) will be 2^{l-1} so total number of steps will be $\log_2 n$. The no. of comparison at any level will be maximum n. so we say run time of Quicksort will be $\Omega(n\log_2 n)$.

Worst Case:

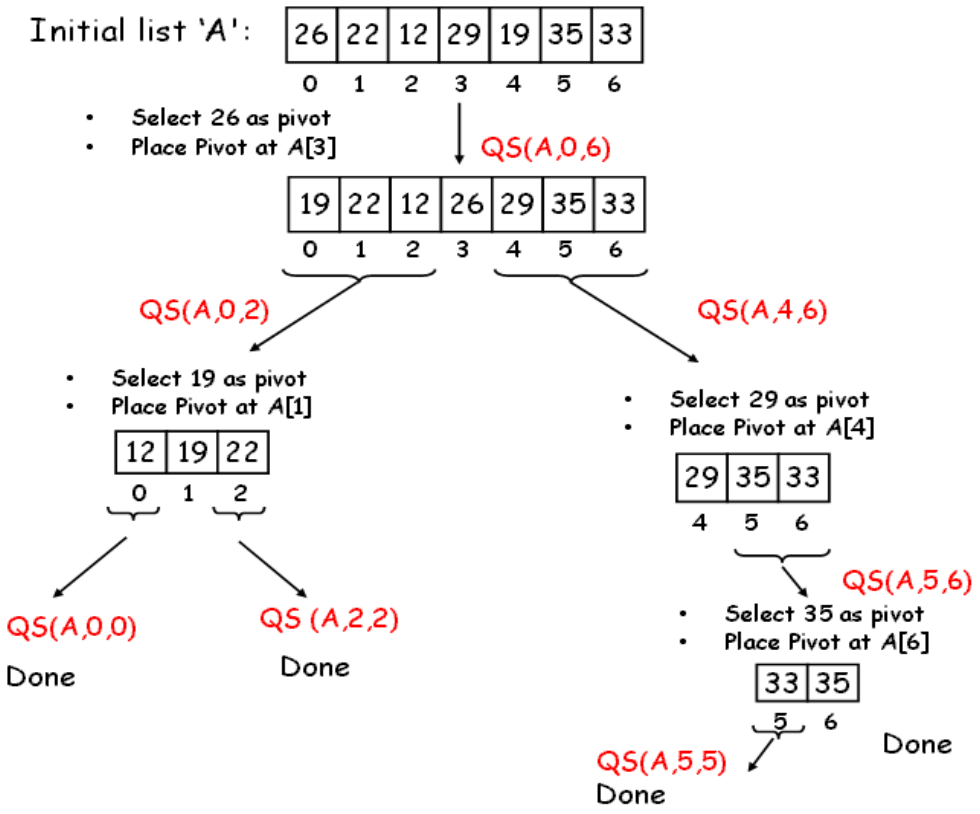
Suppose list of elements are already in sorted order. When we find the pivot then it will be first element. So here it produces only 1 sublist which is on right side of first element and starts from second element. Similarly other sublists will be created only at right side. The no. of comparisons for first element is n, second element requires n-1 comparisons and so on.. So the total no. of comparisons will be:

$n + n-1 + \ldots. 2 + 1 = n (n-1) / 2 = O(n^2)$.

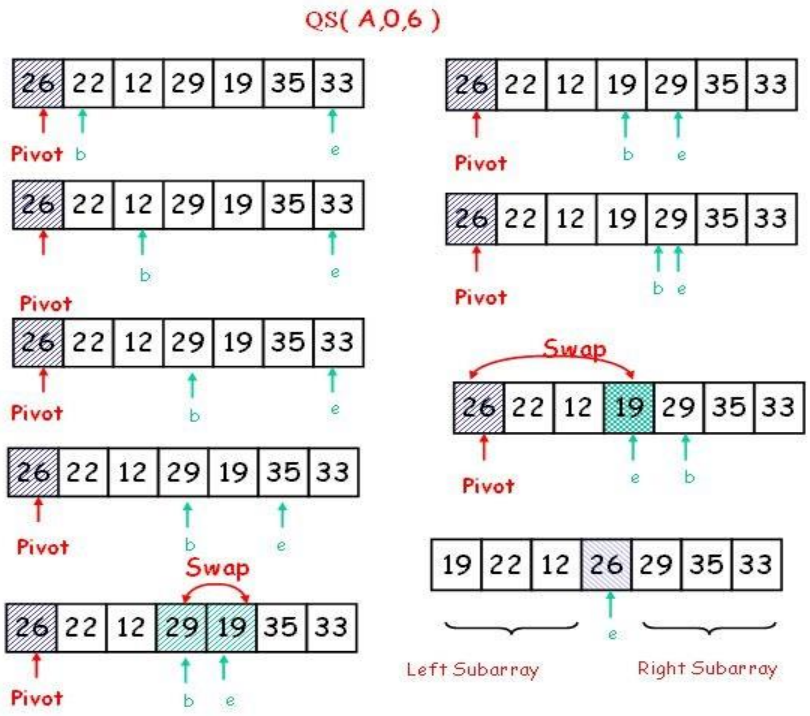
Average Case:

The average case performance of QuickSort is $\Theta(n\log n)$

Example:



Step by Step Process for QS (A, 0, 6)



Implementation:

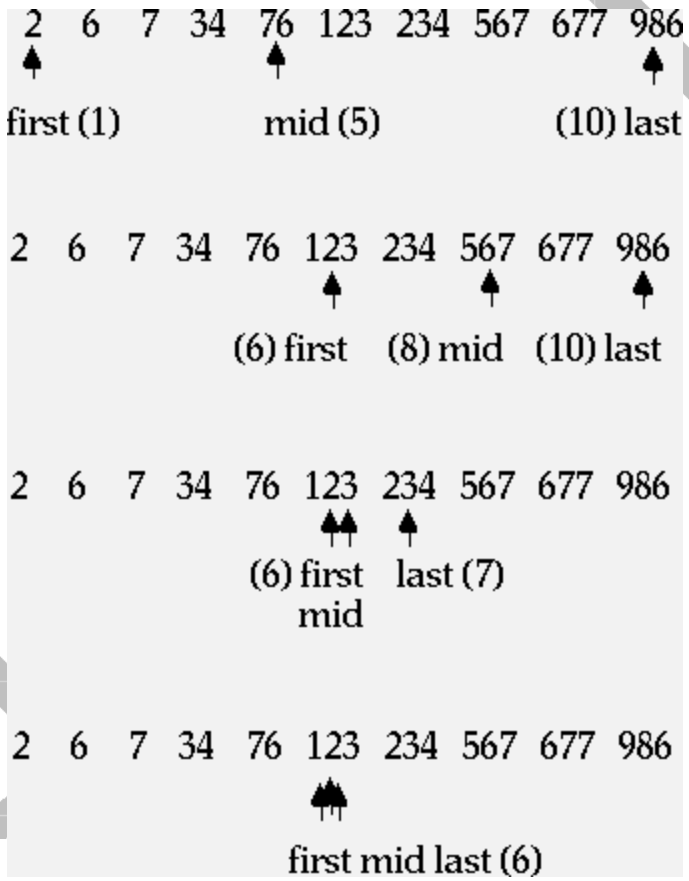
```
void quick_sort(int a[],int l,int u)
{
   int j;
   if(l<u)
   {
      j=partition(a,l,u);
      quick_sort(a,l,j-1);
      quick_sort(a,j+1,u);
   }
}

int partition(int a[],int l,int u)
{
   int v,i,j,temp;
   v=a[l];
   i=l;
   j=u+1;

   do
   {
      do
         i++;

      while(a[i]<v&&i<=u);

      do
         j--;
      while(v<a[j]);

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

   a[l]=a[j];
   a[j]=v;

   return(j);
}
```

JSCOE

1. Binary Search:

In binary searching we compare the item with the element in the center of the list and then restrict our attention to only the first or second half of the list, depending on whether the 'item' comes before or after the central one. If the 'item' is in the first half of the list, the procedure is repeated on the first of the list. In this way at each step we reduce the length of the list to be searched by half. And this process continues until either the required 'item' is found or the search intervals become empty.

E.g. target item=123.



Algorithm: Binary Search:

Binary Search (a, n, item)

//Here 'a' is an array of 'n' number of elements and 'item' represents the item to be searched in array 'a'.

Step 1: Set lower = 0;

Step 2: Set upper = n-1;

Step 3: Repeat through step-5

while (lower<= upper).

Step 4: mid= (lower + upper)/2;

Step 5: Compare a [mid] and item.

If (a [mid] = =item) then return (mid) and go to step-7.

Else if (a [mid]> item) then

Upper= mid-1;

Else lower= mid+1;

Step 6: If item not found then just print a message- " Item not found" and return(-1).

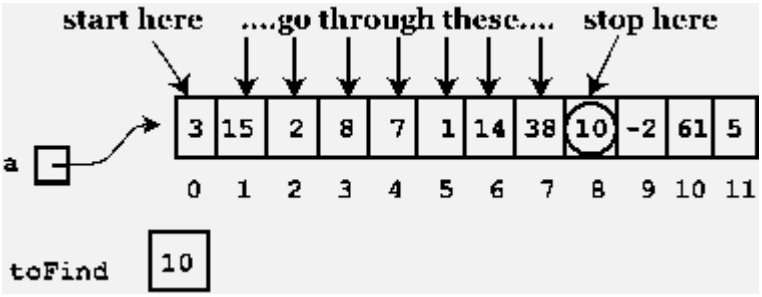
Step 7: Exit.

Implementation of Binary Search:

```
Binary Search (int a [], int n, int item)
{
        int mid, lower, upper;
        lower= 0;
        upper= n-1;
        while (lower<= upper)
        {
                mid = (lower+ upper)/2;
                if (a[mid] = = item)
                        return (mid);
                else if ( a[mid]> item)
                        upper =mid -1;
                else
                        lower = mid +1;
        }
        return (-1);
}
```

2. Linear Search:

The simplest form of a search is the sequential searching or linear searching. In sequential searching we search for a target 'item' in the list by examining the records in the list one by one. Initially the process starts with the first record. If this first comparison results in false then the algorithm proceeds to the next record. This process continues until either the desired record is found or the list is exhausted.



Algorithm: Linear Search:

SequentialSearch (a, n, item)

//Here 'a' is an array of 'n' number of strings and 'item' represents the item to be searched in array 'a'.

Step 1: Initialize i=0;

Step 2: Repeat through step-4 while (i<n).

Step 3: Compare a[i] and item.

If (strcmp(a[i],item)==0) then return (i) and go to step-6.

Step 4: Increment the value of 'i' as i=i+1.

Step 5: If item not found then just print a message- " Item not found" and return(-1).

Step 6: Exit.

Implementation of Linear Search:

```
SequentialSearch (char a [][MAX], int n, int item)
{
        int i;
        for (i=0;i<n;i++)
        {
                If(strcmp(a[i],item)==0))
                        return (i+1);
        }
        return (-1);
}
```

CONCLUSION: In this way, we have studied how to implement different searching and sorting methods.

FAQ:

1. What is searching?
2. What is sorting?
3. List the advantages and disadvantages of Binary Search?
4. Explain the analysis of Binary Search with respect to best case, worst case & Average case?
5. Analyze Bubble sort with respect to Time complexity?
6. Give applications of Binary Search?
7. What is pivot element in Quick sort?
8. Differentiate between quick sort and insertion sort.
9. Explain Binary Search with example.