TOPIC ➔ SORTING AND SEARCHING

CONTENTS:

1) **Sorting Techniques**

   [Bubble, Insertion, Quick, Bucket, Merge, Selection, Shell]

2) **Searching Techniques**

   [Index, Sequential, Binary]

❖ **INTRODUCTION:**

Computer systems are often used to store large amounts of data from which individual records can be retrieved according to some search criterion. Thus the efficient storage of data to facilitate fast searching is an important issue. Sorting is one of the most important operations performed by computers. In the days of magnetic tape storage before modern databases, it was almost certainly the most common operation performed by computers as most database updating was done by sorting transactions and merging them with a master file. It's still important for presentation of data extracted from databases: most people prefer to get reports sorted into some relevant order before wading through pages of data. We shall investigate the performance of some searching and sorting algorithms and the data structures, which they use.

❖ **SORTING:**

Sorting means arranging a set of data in some order. There are different methods that are used to sort the data in ascending or descending order. These methods can be divided into two categories.

❖ **INTERNAL SORTING:**

If all the data that is to be sorted can be accommodated at a time in memory then internal sorting methods are used. There are different types of internal sorting methods. Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quick Sort are the examples of internal sorting.

❖ **EXTERNAL SORTING:**

When the data to be sorted is so large that some of the data is present in the memory and some is kept in auxiliary memory (hard disk, floppy, tapes etc.) then external sorting methods are used.

1. **SELECTION SORT:**

Selection sort is a simple sorting algorithm, specifically an in-place comparison sort. It is a technique to arrange the data in proper order. This type of sorting is called "***Selection Sort***" because it works by repeatedly selecting smallest order element.

If we want to sort array in increasing order (i.e. the smallest element at the beginning of the array and the largest element at the end). And vice - versa for decreasing order (i.e. the largest element at the beginning of the array and the smallest element at the end).

The process of selection sort is explained here (Working of selection sort).

1	**44**	**33**	**33**	**22**	11	11	11	11	11	11	11
2	**33**	44	44	44	**44**	**44**	**33**	22	22	22	22
3	55	**55**	55	55	**55**	55	55	**55**	**44**	33	33
4	22	22	**22**	33	33	**33**	44	**44**	55	**55**	44
5	11	11	11	**11**	22	22	**22**	33	**33**	**44**	55

This process continues until all the elements are sorted.
Write a C Program for Selection Sort.

ALGORITHM OF SELECTION SORT.

sel_sort(arr, n)
Where arr -> represents the list of elements
n -> represents the size of the list
Step – 1: i =0 [initially]
Step – 2: Repeat through Step – 7 while (i < n-1)
Step – 3: j = i + 1
Step – 4: Repeat through Step – 6 while (j < n)
Step – 5: if(array[i] > array[j])
i) temp = array[i]
ii) array[i] = array[j]
iii) array[j] = temp
Step – 6: j = j + 1
Step – 7: i = i + 1
Step – 8 : Exit

/* PROGRAM FOR SELECTION SORT */

```
#include<stdio.h>
#include<conio.h>

void sel_sort(int arr[],int);
void main()
{
    int arr[20],i,n;
    clrscr();
    printf("Enter the limit : ");
    scanf("%d",&n);

    for(i=0;i<n;i++)
    {
        printf("Enter [%d] element for list : ", i);
        scanf("%d",&arr[i]);
    }

    sel_sort(arr,n);
```

```
    printf("\nSORTED LIST\n");
    for(i=0;i<n;i++)
    {
        printf(" %d",arr[i]);
    }
    getch();
}

void sel_sort(int array[],int n)
{
    int temp,i,j;
    for(i=0;i<n-1;i++)
    {
        for(j=i+1;j<n;j++)
        {
            if(array[i] > array[j])
            {
                temp = array[i];
                array[i] = array[j];
                array[j] = temp;
            }
        }
    }
}
```

2. <u>INSERTION SORT:</u>

Insertion sort is very simple and efficient algorithms for the smallest lists. Its mechanism is very simple just take one element from the list one by one and inserting them in their correct position into a new sorted list.

The name insertion sorting means that sorting is occurred by inserting a particular element at a proper position.

The process of insertion sort is explained here (Working of insertion sort).

Pass	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	Process
	65	50	30	35	25	45	Original Array
1	**50**	65	30	35	25	45	50 is inserted
2	**30**	50	65	35	25	45	30 is inserted
3	30	**35**	50	65	25	45	35 is inserted
4	**25**	30	35	50	65	45	25 is inserted
5	25	30	35	**45**	50	65	45 is inserted

This process continues until all the elements are sorted.

Explanation:

Pass 1: a[1] is compared with a[0] means 65 with 50. Since 50 < 65, so 50 is inserted at position 0. 65 is shifted one position to the right. The sorted sub array is: a[0], a[1].

Pass 2: a[2] is inserted into its proper place in a[0], a[1]. Since 30 < 50, so it is inserted at position 0 and 50 and 65 are shifted to the right by one position. The sorted sub array is: a[0], a[1], a[2].

Pass 3: a[3] is inserted into its proper place in a[0], a[1], a[2]. So that the sub array a[0], a[1], a[2], a[3] is sorted.

Pass 4: a[4] is inserted into its proper place in a[0], a[1], a[2], a[3]. The element a[4] is 25. This is smaller than a[0] (means 30) and other elements are moved one position to the right. a[0] ….. a[4] is sorted.

Pass 5: a[5] is compared with a[0] (means 25), next with a[1] (means 30), next with a[2] (means 35), next with a[3] (means 50). Since 35 < 50, it takes the position 3, and 50 and 65 are shifted to the right by one location.

Write a C Program for Insertion Sort.

ALGORITHM OF INSERTION SORT.

insertion_sort(a, n)
Where a -> represents the list of elements
n -> represents the size of the list
Step – 1: i =0 [initially]
Step – 2: Repeat through Step – 10 while (i < n)
Step – 3: j = 0
Step – 4: Repeat through Step – 9 while (j < i)
Step – 5: k = 0
Step – 6: Repeat through Step – 8 while (k < j)
Step – 7: if(a[j] < a[k])
i. temp = a[j]
ii. a[j] = a[k]
iii. a[k] = temp
Step – 8: k = k + 1
Step – 9: j = j + 1
Step – 10: i = i + 1
Step – 11: Exit

/* PROGRAM FOR INSERTION SORT */

```
#include<conio.h>
#include<stdio.h>

void insertion_sort(int [],int);
void main()
{
```

```
    int a[6] = {65,50,30,35,25,45},i;
    clrscr();
    insertion_sort(a,6);
    for(i=0;i<6;i++)
        printf("%d\n",a[i]);
    getch();
}
void insertion_sort(int a[],int n)
{
    int i,j,k,temp;
    for(i=0;i<n;i++)  //insert value from array in sort order
    {
        for(j=0;j<=i;j++)        //to sort from the inserted value
        {
            for(k=0;k<=j;k++)        //sub loop of sort
            {
                if(a[j] < a[k])
                {
                    temp = a[j];
                    a[j] = a[k];
                    a[k] = temp;
                }       //end of if condition
            }       //end of k loop
        }       //end of j loop
    }       //end of i loop
}   //end of function
```

3. BUBBLE SORT:

The simplest sorting method is "***Bubble Sort***". In these techniques, we continually compare two items from the list. If they are in wrong order then swap it and then after sorting is completed. This process is used frequently until no swaps are needed. It means that sorting is completed and whole list is sorted. From its name "bubble", the smallest elements at the top of the list and largest elements at the bottom of the list. It is known as comparison sort or exchange sort because it continually compares two adjacent elements from the list.

The process of bubble sort is explained here (Working of bubble sort).

1	**44**	33	33	**33**	22	22	22	**22**	11
2	**33**	44	**44**	**22**	33	33	**33**	**11**	22
3	55	**55**	**22**	44	44	**44**	**11**	33	33
4	22	**22**	55	55	**55**	**11**	44	44	44
5	11	11	11	11	**11**	55	55	55	55

This process continues until all the elements are sorted.

Write a C Program for Bubble Sort.

ALGORITHM OF BUBBLE SORT.

Bubble_sort(arr, n)

Where arr -> represents the list of elements
n -> represents the size of the list

Step – 1: i =0 [initially]
Step – 2: Repeat through step – 7 while (i < n)
Step – 3: j = 0
Step – 4: Repeat through step – 6 while (j < n-1)
Step – 5: if(array[j] > array[j+1])
i) temp = array[j]
ii) array[j] = array[j+1]
iii) array[j+1] = temp
Step – 6: j = j + 1
Step – 7: i = i + 1
Step – 8 : Exit

/* PROGRAM FOR BUBBLE SORT */

```
#include<stdio.h>
#include<conio.h>

void bubble_sort(int arr[],int n);
void main()
{
    int arr[20],i,n;
    clrscr();
    printf("Enter the limit for array : ");
    scanf("%d",&n);

    for(i=0;i<n;i++)
    {
        printf("Enter [%d] element for list : ",i);
        scanf("%d",&arr[i]);
    }

    bubble_sort(arr,n);
    printf("\nSORTED LIST\n");

    for(i=0;i<n;i++)
    {
        printf(" %d",arr[i]);
    }

    getch();
}

void bubble_sort(int array[],int n)
{
    int temp,i,j;
```

```
        for(i=0;i<n;i++)
        {
                for(j=0;j<n-1;j++)
                {
                        if(array[j] > array[j+1])
                        {
                                temp = array[j];
                                array[j] = array[j+1];
                                array[j+1] = temp;
                        }
                }
        }
}
```

4. <u>QUICK SORT:</u>

Quick Sort is very popular sorting method. The name comes from the fact that, in general, quick sort can sort a list of data elements significantly faster than any of the common sorting algorithms. Quick sort treats an array as a list of elements. When sort begins it selects the list's middle element as the list pivot. The sort then divides the list into two sub-lists one with elements that are less than the list pivot and the second with elements greater than or equal to the list pivot. The sort then recursively invokes itself with both the lists. Each time when the sort is invoked, it further divides the elements into smaller sub-list. Quick sort is also known as *<u>Partition Exchange Sort.</u>*

1	2	3	4	5	6	7	8	9
6	**2**	**1**	**3**	**4**	**5**	**8**	**7**	**0**

Consider we have above list with 9 elements.

(a)

Step – 1:

low = first = 1
high = last = 9
pivot = array[(low + high)/2]
= array[(1 + 9)/2]
= array[5]
= 4

1	2	3	4	5	6	7	8	9
6	**2**	**1**	**3**	**4**	**5**	**8**	**7**	**0**

Pivot

Step – 2: low <= high, i.e. 1 <= 9
Step – 3: array[low] < pivot is false because 6 > 4.
Step – 4: low = 1 [remains previous value]
Step – 5: array[high] > pivot is false because 0 < 4.
Step – 6: high = 9 [remains previous value]
Step – 7: low < high, i.e. 1 < 9
temp = array[low] = array[1] = 6

array[1] = array[high] = array[9] = 0
array[9] = temp = 6
low = low + 1
= 1 + 1
low = 2
high = high – 1 = 9 - 1 = 8

Now the list becomes

1	2	3	4	5	6	7	8	9
0	**2**	**1**	**3**	**4**	**5**	**8**	**7**	**6**

Pivot

(b)
Step – 2: low <= high i.e. 2 < 8.

Step – 3: array[low] < pivot i.e. array[2] < pivot i.e. 2 < 4
Step – 4: **(1)** low = low + 1 means 2 + 1 = 3

Step – 3: array[low] < pivot i.e. array[3] < pivot i.e. 1 < 4
Step – 4: **(2)** low = low + 1 means 3 + 1 = 4

Step – 3: array[low] < pivot i.e. array[4] < pivot i.e. 3 < 4
Step – 4: **(3)** low = low + 1 means 4 + 1 = 5

Step – 5: array[high] > pivot i.e. array[8] > pivot i.e. 7 > 4
Step – 6: **(1)** high = high – 1 means 8 – 1 = 7

Step – 5: array[high] > pivot i.e. array[7] > pivot i.e. 8 > 4
Step – 6: **(2)** high = high – 1 means 7 – 1 = 6

Step – 5: array[high] > pivot i.e. array[6] > pivot i.e. 5 > 4
Step – 6: **(3)** high = high – 1 means 6 – 1 = 5

Now the list is broken into two sub lists:

1	2	3	4	5
0	**2**	**1**	**3**	**4**

1	2	3	4
5	**8**	**7**	**6**

If we continue above process, finally we will get a sorted list.

1	2	3	4	5	6	7	8	9
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**

Write a C Program for Quick Sort.

```
/* PROGRAM FOR QUICK SORT */
#include<stdio.h>
#include<conio.h>
void quick_sort(int[],int,int);

void main()
{
    int value[100],i,n;
    clrscr();
    printf("Enter size of array : ");
    scanf("%d",&n);

    for(i=0;i<n;i++)
    {
        printf("Enter Number : ");
        scanf("%d",&value[i]);
    }
    quick_sort(value,0,n-1);

    printf("Sorted List \n");
    for(i=0;i<n;i++)
        printf(" %d ",value[i]);
    getch();
}

void quick_sort(int a[],int first,int last)
{
    int temp, low, high, mid;
    low = first;
    high = last;
    mid = a[(first+last)/2];
    while(low <= high)
    {
        while(a[low] < mid)
            low ++;
        while(a[high] > mid)
            high --;
        if(low <= high)
        {
            temp = a[low];
            a[low] = a[high];
            a[high] = temp;
            low++;
            high--;
        }

    }
```

```
    if(first < high)
        quick_sort(a,first,high);
    if(low < last)
        quick_sort(a,low,last);
}
```

5. MERGE SORT:

Merging lists means combining two sorted lists into one sorted list. For this the elements from both the sorted lists are compared. The smaller of both the elements is then stored in the third array. The sorting is complete when all the elements from both the lists are placed in the third list.

Suppose array l1 and l2 contains 5 elements each (i.e. l1 = {11, 2, 9, 13, 57} and l2 = {25, 17, 1, 90, 3} and after sorting each array separately it will be like l1 = {2, 9, 11, 13, 57} and l2 = {1, 3, 17, 25, 90}). Then merge sort algorithm works as follows:

- The array l1 and l2 are sorted using any algorithm (sorting method).
- The 0^{th} element from the first array is 2, is compared with 0^{th} element of second array 1. Since 1 is smaller than 2, 1 is placed in the third array. New array will be 1.
- The 0^{th} element from the first array 2, is compared with 1^{st} element of second array 3. Since 2 is smaller than 3, 2 is placed in the third array. New array will be 1 2.
- The 1^{st} element from the first array is 9, is compared with 1^{st} element of second array 3. Since 3 is smaller than 9, 3 is placed in the third array. New array will be 1 2 3.
- The 1^{st} element from the first array is 9, is compared with 2^{nd} element of second array 17. Since 9 is smaller than 17, 9 is placed in third array. New array will be 1 2 3 9.
- The same procedure is repeated till end of one of the arrays is reached. Now remaining elements from the other array are placed directly into the third list as are already in sorted order.

Write a C Program for Merge Sort.

```
/* PROGRAM FOR MERGE SORT */
#include<stdio.h>
#include<conio.h>
void main()
{
    int l1[5] = {11,2,9,13,57},i;
    int l2[5] = {25,17,1,90,3},j;
    int merge[10],k,temp;
    clrscr();
    for(i=0;i<5;i++)
        printf("Value of first list %d\n",l1[i]);
    printf("\n");
    for(j=0;j<5;j++)
        printf("Value of second list %d\n",l2[j]);

    //Sorting Steps
    for(i=0;i<5;i++)
    {
        for(j=0;j<5;j++)
        {
```

```
                if(l1[i] < l1[j])
                {
                        temp = l1[i];
                        l1[i] = l1[j];
                        l1[j] = temp;
                }
                if(l2[i] < l2[j])
                {
                        temp = l2[i];
                        l2[i] = l2[j];
                        l2[j] = temp;
                }
        }
}

for(i=0;i<5;i++)
        printf("Sorted value of first list %d\n",l1[i]);
printf("\n");
for(j=0;j<5;j++)
        printf("Sorted value of second list %d\n",l2[j]);

//Merge sort steps
i  = j = k = 0;
while(i<5 && j<5)
{
        if(l1[i] < l2[j])
        {
                merge[k] = l1[i];
                i++;
                k++;
        }
        else
        {
                merge[k] = l2[j];
                j++;
                k++;
        }
}
while(i<5)
{
        merge[k] = l1[i];
        i++;
        k++;
}
while(j<5)
{
        merge[k] = l2[j];
        j++;
        k++;
```

```
    }
    printf("Value of Merge Sort\n");
    for(k=0;k<10;k++)
        printf(" %d ",merge[k]);
    getch();
}
```

❖ **SEARCHING:**

Searching is an operation which finds the location of a given element in a list. the search is said to be successful or unsuccessful depending on whether the element that is to be searched is found or not. There are mainly two standard searching methods – Linear Search and Binary Search.

1. LINEAR SEARCH:

To search (locate, find) an element from the unsorted array list we are using this simplest technique. It simply traverses from top to bottom in the array and searches for the key (target) value from the list and displays output as well. It is also called as sequential searching method.

In this technique searching element (value) is compared with the element of the list, if match is found then the search it terminated. Otherwise next element from the list is fetched and compared with the searching element and this process is continued till the searching element is found or list is completely traversed. At last, if searching element is not found from the array, it displays a message that key element is not found in the list.

Write a C Program for Sequential (Linear) Search.

```
/* PROGRAM FOR SEQUENTIAL (LINEAR) SEARCH */
#include<stdio.h>
#include<conio.h>
#define size 10

void lnr_search(int[],int);
void main()
{
    int lst[size], key, i;
    char ans = 'y';
    clrscr();
    for(i=0;i<size;i++)
    {
        printf("\nEnter number [%d] : ",i);
        scanf("%d",&lst[i]);
    }
    while(ans == 'y' || ans == 'Y')
    {
        printf("\nEnter the number to search : ");
        scanf("%d",&key);
        lnr_search(lst,key);
        printf("\nDo you want to continue : ");
        fflush(stdin);
```

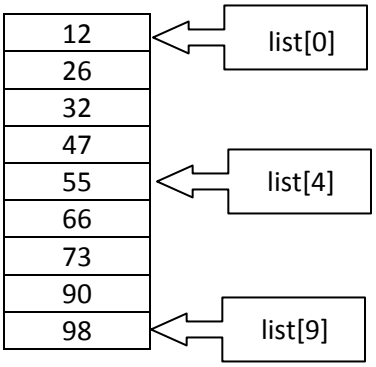
```
            scanf("%c",&ans);
        }
        getch();
}



void lnr_search(int a[],int s_key)
{
    int i,found = 0;
    for(i=0;i<size;i++)
    {       if(a[i] == s_key)
            {
                    printf("\n Search is successful");
                    printf("\nKey = %d, Found at %d position", s_key,i);
                    found = 1;
                    break;
            }
    }
    if(!found)
            printf("\nUnsuccessful");
}
```

2. BINARY SEARCH:

This method of searching is very efficient. The prerequisite of binary search is data from which we want to search must be in sorted order. In this method, to search an element we compare it with middle element of the list, if it match with the element which we wants to search then search will be successful and we stop otherwise the list is divided into two halves, one from the first element to the middle element and other from the middle element to the last element. As a result all the elements in the first half are smaller than middle element while the entire elements in the second half are greater than the center element.

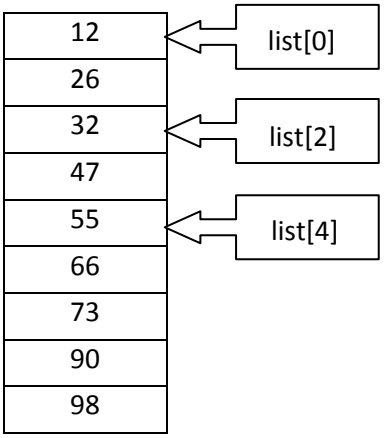
The searching will process either of the two halves depending upon the target element is greater or smaller than the center element. If the element is smaller than the middle element, that process will be for the first half otherwise for the second half.

1	2	3	4	5	6	7	8	9
12	**26**	**32**	**47**	**55**	**66**	**73**	**90**	**98**



low = 0 mid = 4 high = 9 k = 32

1. Initially
   - list[low] = list[0] =12
   - list[high] = list[9] = 98
   - mid = (low + high) / 2 = (0 + 9) / 2 = 4
   - list[mid] = list[4] = 55



2. Since 32 < 55 means k < list[mid] so that
   - high = mid – 1
     high = 4 – 1.
     high = 3.
   - mid = (low + high) / 2
     mid = (0 + 3) / 2 i.e. 3/2
     mid = 1.
   - list[mid] = list[1] = 26

12(low) 26 32 47(high)

3. Since 32 > 26 so that
   - low = mid + 1
     low = 1 + 1
     low = 2
   - mid = (low + high) / 2
     mid = (2 + 3) / 2 i.e. 5 /2
     mid = 2
   - list[mid] = list[2] = 32

```
if (list[mid] == k)
{
        printf("The number exits on %d of the list " , mid + 1);
        flag = 1;
        break;
}
```

Here the condition will be fulfilled and it will give message like
The number exits on 3 of the list.

/* PROGRAM FOR BINARY SEARCH */

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int low=0, mid, high=9, i, k, flag = 0;
        int list[10] = {12,26,32,47,55,66,73,81,90,98};
        clrscr();

        printf("Enter the number to be searched : ");
        scanf("%d",&k);
        mid = (low + high) / 2;

        if(list[low] == k)
        {
                printf("The number exits on the first place");
                getch();
                exit();
        }
        if(list[high] == k)
        {
                printf("The number exits on the last place");
                getch();
                exit();
        }
        while(low <= high)
        {
                if(list[mid] == k)
                {
                        printf("\nThe number exits on %d of the list", mid+1);
                        flag = 1;
                        break;
                }
                else if(list[mid] > k)
                {
                        high = mid - 1;
                        mid = (low + high) / 2;
                }
                else if(list[mid] < k)
                {
                        low = mid + 1;
                        mid = (low + high) / 2;
                }
        }
        if(flag ==0)
                printf("\nThis number is not in the list ");
        getch();
}
```