Data Structures
UNIT-1
Searching and Sorting

Data Structures

- The logical or mathematical model of a particular organization of data is called data structures.

- Data structures is the study of logical relationship existing between individual data elements, the way the data is organized in the memory and the efficient way of storing, accessing, and manipulating the data elements.

- Choice of a particular data model depends on two considerations: it must be rich enough in structure to mirror the actual relationships of the data in the real world.

- On the other hand, the structure should be simple enough that one can effectively process the data when necessary.

Data Structures can be classified as:

- ***Primitive data structures***
- ***Non-Primitive data structures.***

Primitive data structures are the basic data structures that can be directly manipulated/operated by machine instructions.
Some of these are character, integer, real, pointers etc.

Non-primitive data structures are derived from primitive data structures, they cannot be directly manipulated/operated by machine instructions, and these are group of homogeneous or heterogeneous data items.
Some of these are Arrays, stacks, queues, trees, graphs etc.

Data structures are also classified as

Linear data structures
Non-Linear data structures.

In the Linear data structures processing of data items is possible in linear fashion, i.e., data can be processed one by one sequentially.
Example of such data structures are:

Array
Linked list
Stacks
Queues

A data structure in which insertion and deletion is not possible in a linear fashion is called as nonlinear data structure. i.e., which does not show the relationship of logical adjacency between the elements is called as non-linear data structure. Such as trees, graphs and files.

Data structure operations:

- The particular data structures that one chooses for a given situation depends largely on the

- Frequency with which specific operations are performed.

The following operations play major role in the processing of data.

i) Traversing.
ii) Searching.
iii) Inserting.
iv) Deleting.
v) Sorting.
vi) Mergin

SEARCHING

1)Linear Search:

- Linear search is a technique of finding whether a Key number is present in the array or not. In this technique the array is traversed from the beginning of the array by comparing the key element with each array elements till the key element is found or the comparison is done with all the elements of the array.

Write a C program to search for an element in an array using linear search technique.

```
#include <stdio.h>
void main()
{  int num;
   int i,  keynum, found = 0;
   printf("Enter the number of elements ");
   scanf("%d", &num);
   int array[num];
   printf("Enter the elements one by one \n");
   for (i = 0; i < num; i++)
   {
      scanf("%d", &array[i]);
   }
   printf("Enter the element to be searched ");
   scanf("%d", &keynum);
   /*  Linear search begins */
   for (i = 0; i < num ; i++)
   {
      if (keynum == array[i] )
      {
         found = 1;
         break;
      }
   }
   if (found == 1)
```

```
        printf("Element is present in the array at position %d",i+1);
    else
        printf("Element is not present in the array\n");
}
```

2) BINARY SEARCH

- In Binary search technique, the elements of the arrays must be sorted (ascending order or descending order).

- The beg is set to the first position of the array, The end is set to the last position of the array and the whole array's mid position is calculated and the mid value is set.

- The element of the array pointed by mid is compared with the key element. If the Key element is found than the search is completed with a successful message or the key element is compared with the element in the mid position whether the key element is lesser than the mid element, if true then the end position is set to mid-1. Because if at all the element is present, then it will be in the first half of the array, else, if the key element is larger than the mid element then the beg position is set to mid+1, because if at all the element is present, then it will be in the second half of the array. This process is continued till the search is successful or till the beg is greater than the end.

Write a C program to search for an element in an array using binary search technique

```
#include<stdio.h>

int main()
{
    int arr[50],i,n,x,flag=0,first,last,mid;

    printf("Enter size of array:");
```

```
    scanf("%d",&n);
    printf("\nEnter array element(ascending order)\n");

    for(i=0;i<n;++i)
        scanf("%d",&arr[i]);

    printf("\nEnter the element to search:");
    scanf("%d",&x);

    first=0;
    last=n-1;

    while(first<=last)
    {
        mid=(first+last)/2;

        if(x==arr[mid]){
            flag=1;
            break;
        }
        else
            if(x>arr[mid])
                first=mid+1;
            else
                last=mid-1;
    }

    if(flag==1)
        printf("\nElement found at position %d",mid+1);
    else
        printf("\nElement not found");

    return 0;
}
```

SORTING

Definition and techniques

- Sorting is the technique of arranging the elements in ascending or descending order.

1) Selection Sort:

- The selection sort is based on the minimum/maximum technique, By means of a nested for loops, a pass through the array is made to locate the minimum val***u***e. Once this is found, it is placed in the first position of the array (position 0). Another pass through the remaining elements is made to the next smallest element, which is placed in the second position (position 1), and so on.

- Once the next-to-last element has been compared with the last one, all the elements of the array have been sorted into ascending order.

Write a C program to sort the elements in the array using selection Sort technique.

```
#include<stdio.h>
int main()
{
   int a[30],i,j,n,temp,min;
   printf(“Enter the array size”);
   scanf("%d",&n);
   Printf(“Enter the array elements”);
   for(i=0;i<n;i++)
   {
      scanf("%d",&a[i]);
   }
   for(i=0;i<n;i++)
   {
      min=i;
      for(j=i+1;j<n;j++)
      {
         if(a[j]<a[min])
         min=j;
      }
      temp=a[i];
```

```
        a[i]=a[min];
        a[min]=temp;
    }
    printf("Sorted elements are");
    for(i=0;i<n;i++)
    {
        printf("%d\n",a[i]);
    }
    return 0;
}
```

2) Bubble Sort:

- In Bubble sort each element is compared with the adjacent element. If the first element is larger than the second then the position of the elements are interchanged, otherwise it is not changed.

- Then next element is compared with the adjacent element and same process is repeated for all the elements in the array.

- During the first pass, the largest element occupies the last position. During the next pass the same process is repeated leaving the largest element.

Write a C program to sort the elements in the array using Bubble Sort technique.

```
#include<stdio.h>
int main()
{
       int i,n,temp,j,a[30];
         Printf(“Enter the array size”);
         scanf("%d",&n);
         Printf(“Enter array elements”);
         for(i=0;i<n;i++)
        {
           scanf("%d",&a[i]);
        }
        for(i=1;i<n;i++)
       {
          for(j=0;j<n-1;j++)
```

```
        {
            if(a[j]>a[j+1])
            {
                temp=a[j];
                a[j]=a[j+1];
                a[j+1]=temp;

            }
        }
    }
      Printf(“Sorted elements of the array are”);
    for(i=0;i<n;i++)
    {
       printf("%d\n",a[i]);
    }
    return 0;
}
```

3) Merge Sort:

- Merge sort is one of the popular sorting algorithm
- It is based on the principle of Divide and Conquer algorithm
- The problem is divided into multiple sub problems and each sub problem is solved individually. And finally sub problems are combined to form the final solution.

Write a C program to sort the elements in the array using Merge Sort technique.

```
#include<stdio.h>
void mergesort(int, int);
void mergearray(int, int, int, int);
int arrSort[100];
int main()
{
   int i,n;
   printf("Enter the size of the array:");
   scanf("%d",&n);
   printf("Enter the array elements:");
   for(i=0;i<n;i++)
```

```
    scanf("%d",&arrSort[i]);
    mergesort(0,n-1);
    printf("order of sorted elements:");
    for(i=0;i<n;i++)
    printf("%d",arrSort[i]);
return 0;
}
void mergesort(int i,int j)
{
  int m;
  if (i < j)
  {
    m = (i + j) / 2;
    mergesort(i, m);
    mergesort(m + 1, j);
    mergearray(i, m, m + 1, j);
  }
}
void mergearray(int a, int b, int c, int d)
{
  int t[50];
  int i = a, j = c, k = 0;

  while (i <= b && j <= d)
  {
    if (arrSort[i] < arrSort[j])
      t[k++] = arrSort[i++];
    else
      t[k++] = arrSort[j++];
  }
  while (i <= b)
    t[k++] = arrSort[i++];

  while (j <= d)
    t[k++] = arrSort[j++];

  for (i = a, j = 0; i <= d; i++, j++)
    arrSort[i] = t[j];
}
```

4) Insertion Sort:

- Insertion sort is a sorting algorithm that places an unsorted element at its suitable place in each iteration.
- Insertion sort works similarly as we sort cards in our hand in a card game.
- We assume that the first card is already sorted then, we select an unsorted card. If the unsorted card is greater than the card in hand, it is placed on the right otherwise, to the left. In the same way, other unsorted cards are taken and put in their right place.
- A similar approach is used by insertion sort.

Write a C program to sort the elements in the array using Insertion Sort technique.

```
#include <stdio.h>
int main() {S
    int n, i, j, key;
    int arr[64];
    printf("Enter number of elements\n");
    scanf("%d", &n);
    printf("Enter %d integers\n", n);
    for (i = 0; i < n; i++)
    {
```

```
        scanf("%d", &arr[i]);
    }
    for(i=1;i<n;i++)
    {
        key=arr[i];
        j=i-1;
        while(j>=0 && arr[j]>key)
        {
            arr[j+1]=arr[j];
            j=j-1;
        }
        arr[j+1]=key;
    }
    printf("the elements after sorting are");
    for (i = 0; i < n; i++)
    {
        printf("%d\n", arr[i]);
    }

    return 0;
}
```

5) Quick Sort:

- Quicksort is a sorting algorithm based on the divide and conquer approach where an array is divided into sub arrays by selecting a pivot element (element selected from the array).

- While dividing the array, the pivot element should be positioned in such a way that elements less than pivot are kept on the left side and elements greater than pivot are on the right side of the pivot.

- The left and right sub arrays are also divided using the same approach. This process continues until each sub array contains a single element.

- At this point, elements are already sorted. Finally, elements are combined to form a sorted array.

Write a C program to sort the elements in the array using Quick Sort technique.

```
#include<stdio.h>
int main()
{
  int i,n,a[25];
  printf("Enter the size of the array:");
  scanf("%d",&n);
  printf("Enter elements:");
  for(i=0;i<n;i++)
    scanf("%d",&a[i]);

  quicksort(a,0,n-1);
```

```
  printf("Order of Sorted elements: ");
  for(i=0;i<n;i++)
    printf(" %d",a[i]);
}
void quicksort(int number[25],int first,int last)
{
  int i, j, pivot, temp;

  if(first<last)
  {
    pivot=first;
    i=first;
    j=last;

    while(i<j)
    {
      while(number[i]<=number[pivot]&&i<last)
        i++;
      while(number[j]>number[pivot])
        j--;
      if(i<j)
      {
        temp=number[i];
        number[i]=number[j];
        number[j]=temp;
      }
```

```
    }

    temp=number[pivot];
    number[pivot]=number[j];
    number[j]=temp;
    quicksort(number,first,j-1);
    quicksort(number,j+1,last);

  }
return 0;
}
```