LAB NO. 3

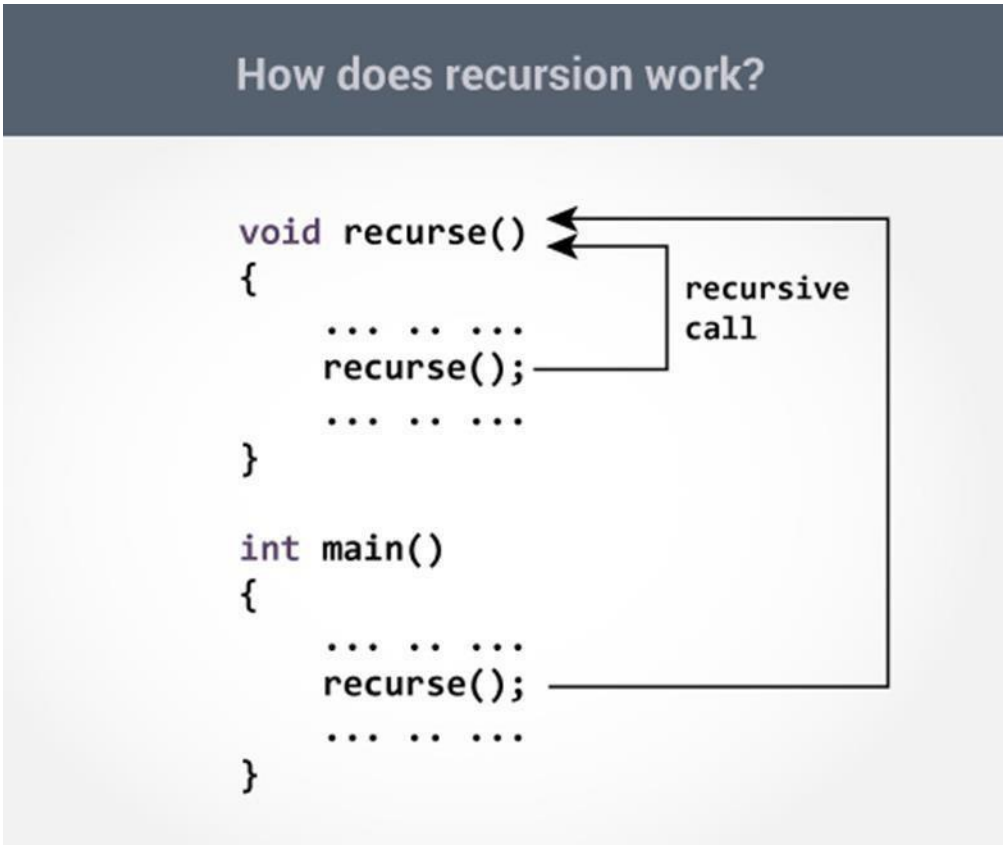
Recursion & Searching Algorithms

OBJECTIVE:

- To understand & implement the working of recursive functions in C++.
- To understand & Implement searching algorithms in C++.

RECURSION:

A function that calls itself is known as recursive function. And, this technique is known as recursion. The figure below shows how recursion works by calling itself over and over again.



The recursion continues until some condition is met.

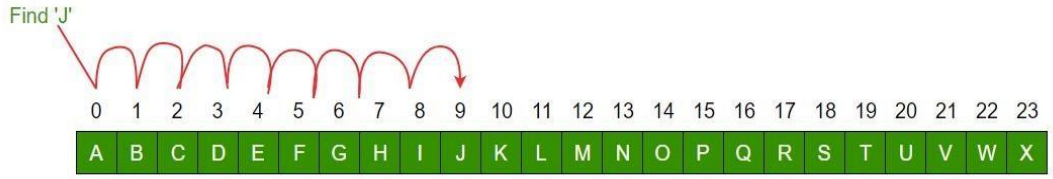
To prevent infinite recursion, if...else statement (or similar approach) can be used where one branch makes the recursive call and other doesn't.

Example: C++ Program to assign data to members of a structure variable and display it.

```
#include <iostream>
using namespace std;

int factorial(int);

int main( )
{    int n;
   cout<<"Enter a number to find factorial: ";
cin >> n;
cout << "Factorial of " << n <<" = " << factorial(n);
 return 0;
}

int factorial(int n)
{
   if (n > 1)
   {
      return n*factorial(n-1);
   }
else
{
      return 1;
   }
}
```

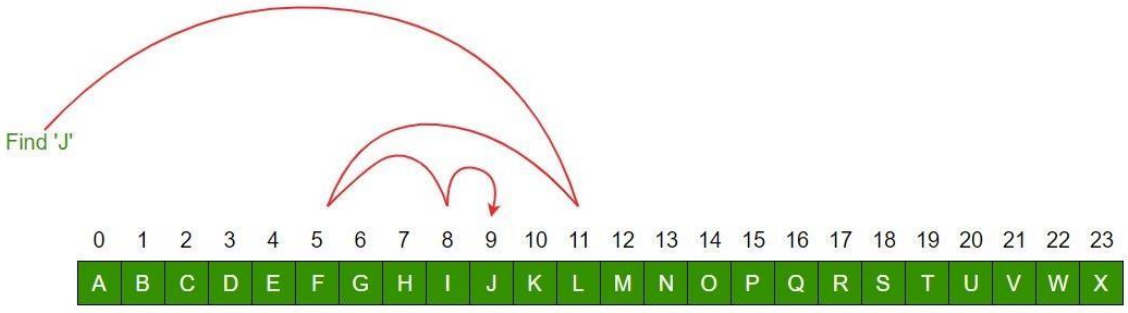
LINEAR SEARCH:

The simplest form of a search is the linear search. This technique is meant for searching a particular item in an unsorted data set in a sequential manner until the desired data item is found. Linear search is easy to write and efficient for short lists, but inefficient for long ones. To find any element in a long array (list), there are far more efficient methods, provided that the array is sorted. A program for linear search is as follows:



BINARY SEARCH:

Binary search is a simple method of accessing a particular item in a sorted (ordered) data set. A search for a particular item with a certain key value resembles the search for a name in telephone directory or a word in a dictionary. The approximate middle item of the data set is located, and its key value is examined. If its value is too high, then the key of the middle element of the first half of the set is examined and procedure is repeated on the first half until the required item is found. If the value is too low, then the key of the middle entry of the second half of the data set is tried and the procedure is repeated on the second half. This process continues until the desired key is found or search interval becomes empty. The binary search algorithm is based on binary search tree. A queue is different from a stack in that a queue works on a First in First out (FIFO) basis. In general, items are added to the end of a queue (In the way that we join at the end of a queue for a bus) and items are removed from the front of a queue. (The people at the front of the queue for the bus can get on the bus first.)



Example: Linear Search Iterative & Recursive

```
#include <iostream>
using namespace std;
// Linearly search x in arr[].  If x is present then return its
// location,  otherwise return -1

int search(int arr[], int nsize, int x)
{
int i;
   for (i=0; i<nsize; i++)
if (arr[i] == x)
return i+1;
return -1;
}
// Recursive
int recursiveLinearSearch(int array[],int key,int size) {
size=size-1;
if(size <0) {
        return -1;
 }
else if(array[size]==key) {
        return 1;
 }
else {
        return recursiveLinearSearch(array,key,size);
 }
}
// Display
void display(int arr[],int nsize){
for(int i=0;i<nsize;i++){
        cout<<arr[i]<<",";

 }
 cout<<"\b \b"<<endl;
}

int main() {  int item, nsize=16;
 int arr[nsize] = {2,3,4,5,6,2,3,44,5,3,5,3,4,7,8,99};
display(arr, nsize);
cout << "Enter the item you want to find: ";
 cin >> item;
```

```
   cout << "Item is found at location " <<search(arr,nsize,item)<< endl;
  /* int key;
   cout<<"Enter Key To Search  in Array: ";
cin>>key;
int result;
 result=recursiveLinearSearch(arr,key,nsize);
if(result==1) {
          cout<<"Key Found in Array  ";
          }
          else
          {
          cout<<"Key NOT Found in Array  ";
        }
   */
          return 0;
          }
```

Example: Binary Search

```
 #include <iostream>
using namespace std;

int binarySearch(int arr[], int start, int end, int key)
{
   while (start <= end)
   {
      int mid = (start +end)/2;
      // Check if x is present at mid
if (arr[mid] == key)
return mid;
    // If x greater, ignore left half
if (arr[mid] < key)
start = mid + 1;
   // If x is smaller, ignore right half
else
end = mid - 1;
   }

   // if we reach here, then element was  not present
```

```
  return -1;
}
int main(void)
{
   int arr[] = {2, 3, 4, 10, 40};      int n = 5     int key = 10;
   int result = binarySearch(arr, 0, n-1, key);     if(result == -1){
  cout<<"Element is not present in array"<<endl;
 }
else
{
    cout<<"Element is present at index "<<result;
 }
return 0;
}
```

LAB TASKS:

Task 1:

Write a program to find out a number among all other numbers entered by user using linear search technique.

Task 2:

Write a program to find out a number among all other numbers entered by user using Binary search technique.

Task 3:

Find Sum of Fibonacci Series using Recursive Function.

In Lab Tasks:

Task 1:

Given a sorted array of integers, find index of first or last occurrence of a given number. If the element is not found in the array, report that as well.

Task 2:

Given a sorted array of integers, find floor and ceil of a given number in it. The floor and ceil map the given number to the largest previous or the smallest following integer, respectively.

Task 3:

Given a circularly sorted array of integers, find the number of times the array is rotated. Assume there are no duplicates in the array and the rotation is in anti-clockwise direction.

Input : arr = [9, 10, 2, 5, 6, 8]

Output: The array is rotated 2 times