LAB 3

OBJECTIVE: To Implement Recursion Algorithm.

THEORY:

Recursion is a fundamental concept in programming where a function solves a complex problem by breaking it down into smaller, more manageable sub-problems. The function accomplishes this by calling itself. To avoid an infinite loop, recursion requires two key components:

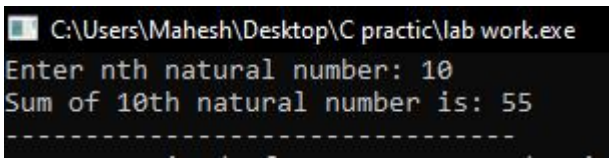
- **Base Case:** This is the condition under which the recursion ends. It provides a simple, straightforward solution to the smallest possible sub-problem.
- **Recursive Case:** This part of the function includes the self-call, breaking down the larger problem into smaller instances of the same problem.

PROGRAM:

To demonstrate the Recursion, here are some problems to solve using recursion method in c programming.

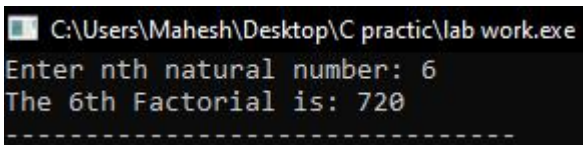
1. Sum of n natural numbers.

```
#include<stdio.h>
int sum(int n){
        if(n==1 || n==0) return n;
        else return n + sum(n-1);
}
int main(){
        int n;
        printf("Enter nth natural number: ");
        scanf("%d",&n);
        if(n>=0)        printf("Sum of %dth natural number is: %d",n,sum(n));
        else    printf("\nCan't Accept negative number..!!!");
        return 0;
}
```



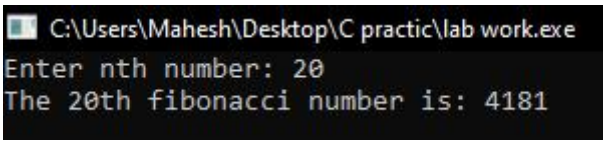
2. Fatorial of n.

```
#include<stdio.h>
int factorial(int n){
        if(n==1 || n==0) return 1;
        else return n * factorial(n-1);
}
int main(){
        int n;
        printf("Enter nth natural number: ");
        scanf("%d",&n);
        if(n>=0)        printf("The %dth Factorial is: %d",n,factorial(n));
        else    printf("\nCan't Accept negative number..!!!");
        return 0;
}
```



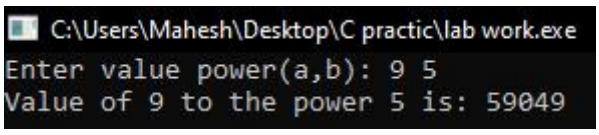
3. Nth fibonacci number.

```
#include<stdio.h>
int fibo(int n){
        if(n==1) return 0;
        else if(n==2 || n==3) return 1;
        else return fibo(n-2) + fibo(n-1);
}
int main(){
        int n;
        printf("Enter nth number: ");
        scanf("%d",&n);
        if(n>0) printf("The %dth fibonacci number is: %d",n,fibo(n));
        else    printf("\nEntered number must be greater then equal to 1..!!!");
        return 0;
}
```



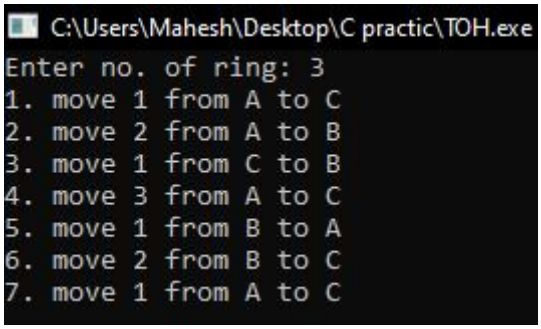
4. Power(a,b)

```
#include<stdio.h>
int power(int a, int b){
        if(b==0) return 1;
        else if(b==1) return a;
        else return a * power(a,b-1);
}
int main(){
        int a,b;
        printf("Enter value power(a,b): ");
        scanf("%d%d",&a,&b);
        if(b>=0)        printf("Value of %d to the power %d is: %d\n",a,b,power(a,b));
        else    printf("\nThe value of b can't be negative..!!!");
        return 0;
}
```



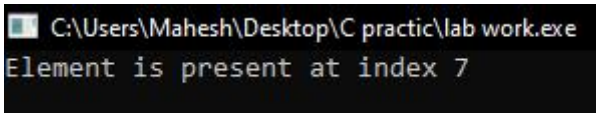
5. Tower of Hanoi (TOH)

```
#include<stdio.h>
int count=1;
void TOH(int n, char A, char B, char C){
        if(n>0){
                TOH(n-1,A,C,B);
                printf("%d. move %d from %c to %c\n",count++,n,A,C);
                TOH(n-1,B,A,C);
        }
}
int main(){
        int n;
        printf("Enter no. of ring: ");
        scanf("%d",&n);
        TOH(n,'A','B','C');
        return 0;
}
```



6. Binary search.

```
#include <stdio.h>

int binarySearch(int arr[], int low, int high, int key) {
    if (high >= low) {
        int mid = low + (high - low) / 2;

        // Check if key is at mid
        if (arr[mid] == key)
            return mid;

        // If key is smaller than mid, search in the left subarray
        if (arr[mid] > key)
            return binarySearch(arr, low, mid - 1, key);

        // Otherwise, search in the right subarray
        return binarySearch(arr, mid + 1, high, key);
    }

    return -1;   // Key not found
}

int main() {
    int arr[] = {1, 2, 5, 7, 10, 15, 20, 25, 30, 40};
    int n = sizeof(arr) / sizeof(arr[0]);
    int key = 25;

    int result = binarySearch(arr, 0, n - 1, key);
    if (result != -1)
        printf("Element is present at index %d\n", result);
    else
        printf("Element is not present in array\n");

    return 0;
}
```



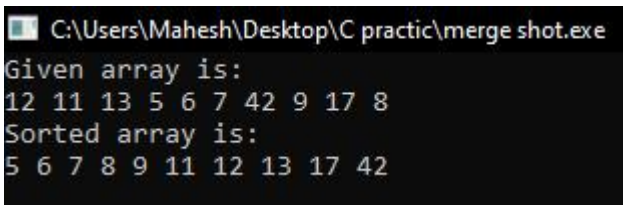
7. Marge Sort

```
#include <stdio.h>

void merge(int arr[], int left, int mid, int right) {
        int n1 = mid - left + 1;
        int n2 = right - mid;
        int i,j,k;

        int L[n1], R[n2]; //temporary arrays
```

```
    for (i = 0; i < n1; i++)
        L[i] = arr[left + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[mid + 1 + j];

    i = 0, j = 0, k = left;

    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}
```

```
void mergeSort(int arr[], int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;

        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);

        merge(arr, left, mid, right);
    }
}

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

int main() {
    int arr[] = {12, 11, 13, 5, 6, 7, 42, 9, 17, 8};
    int arr_size = sizeof(arr) / sizeof(arr[0]);

    printf("Given array is: \n");
    printArray(arr, arr_size);

    mergeSort(arr, 0, arr_size - 1);

    printf("Sorted array is: \n");
    printArray(arr, arr_size);

    return 0;
}
```

……



RESULTS AND DISCUSSION:

The experiment was successful to demonstrate and implement the Recursion Algorithm in C programming. This program helps in C programming language.

CONCLUSION:

This laboratory exercise provided a hands-on experience in C program. Students gained practical knowledge of implementing algorithms in C programming and are now better equipped to undertake more complex programming tasks in the future.