S.No: 1	Exp. Name: ***C program that implements the Selection sort***	**Date: 2025-02-01**

Aim:

Write a C program that implements the Selection Sort algorithm to sort a given list of integers in ascending order. The program should display the elements of the array before and after sorting. Additionally, the program should also display the total number of steps (comparisons and swaps) performed during the sorting process.

Input Format:

- The first line contains an integer n (1 ≤ n ≤ 20), representing the number of elements in the array.
- The second line contains n space-separated integers, representing the elements to be sorted.

Output Format:

- Print the elements of the array before sorting, separated by a space.
- Print the elements of the array after sorting, separated by a space.
- Print the total number of steps (comparisons and swaps) required to sort the array.

Refer to visible test case to strictly match with input/output layout.

Source Code:

```
selectionSortSteps.c
```


ID: 323103310032
2023-2027-CSE-A
Gayatri Vidya Parishad College of Engineering (Autonomous)

```
#include<stdio.h>

void selectionSort(int arr[], int n, int* steps) {
    for(int i=0; i<n-1;i++){
                int minIndex =i;
                for(int j=i+1;j<n;j++){
                        (*steps)++;
                        if(arr[j]<arr[minIndex]){
                                minIndex=j;
                        }
                }
                if(minIndex != i){
                        int temp = arr[minIndex];
                        arr[minIndex] = arr[i];
                        arr[i]=temp;
                        (*steps)++;
                }
        }
}


int main() {
    int n, i, steps = 0;

    // Input number of elements
    scanf("%d", &n);

    int arr[n];

    // Input elements
    for(i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    // Print the array before sorting
    for(i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }

    // Sort the array using Selection Sort and count steps
    selectionSort(arr, n, &steps);

    // Print the array after sorting
    printf("\n");
    for(i = 0; i < n; i++) {
        printf("%d ", arr[i]);
```

```
    }

    // Print the number of steps (comparisons and swaps)
    printf("\n%d", steps);

    return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
6
99 34 67 21 4 56
99 34 67 21 4 56
4 21 34 56 67 99
20

Test Case - 2
User Output
3
5 3 8
5 3 8
3 5 8
4

S.No: 2	Exp. Name: ***Factorial using recursion***	**Date: 2025-01-26**

ID: 323103310032

Aim:

Write a recursive function factorial that accepts an integer n as a parameter and returns the factorial of n, or $n!$

A factorial of an integer is defined as the product of all integers from 1 through that integer inclusive. For example, the call of factorial(4) should return 1 * 2 * 3 * 4, or 24. The factorial of 0 and 1 are defined to be 1.

You may assume that the value passed is non-negative and that its factorial can fit in the range of type int.

Input format:

The first line is the integer that represents the number of test cases.
Each test case will contain a single integer n where $n >= 0$.

Output format:

For each input case, generate and print the factorial of the integer.

Source Code:

factorialcalc.c

```
#include <stdio.h>
int factorial(int n)
{

    if(n==0){
                return 1;
        }
        return n*factorial(n-1);

}
int main()
{
   int T, no;
   scanf("%d",&T);
   while(T--)
   {
     scanf("%d",&no);
     printf("%d\n",factorial(no));
   }
 return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
6
4
24
3
6
11
39916800
7
5040
1
1
0
1

Test Case - 2
User Output
4
5
120
8
40320
9
362880
3
6

S.No: 3	Exp. Name: ***Factorial using non recursive approach***	**Date: 2025-01-26**

Aim:

Write a C program to calculate the factorial of small positive integers using non recursive approach

Input Format:

Single line of input contains an integer N representing the value to find the factorial

Output Format:

Print the factorial result of N

Source Code:

factorial.c

```
#include<stdio.h>

int factorial(int n){
        int fact =1;
        for(int i=n; i>1;i--){
                fact*=i;
        }
        return fact;
}

void main(){
        int n;
        scanf("%d",&n);
        printf("%d", factorial(n));
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
2
2

Test Case - 2

User Output
3
6

Test Case - 3
User Output
4
24

Test Case - 4
User Output
5
120

Test Case - 5
User Output
1
1

S.No: 4	Exp. Name: ***C program that implements the Insertion sort***	**Date: 2025-01-26**

Aim:

Write a C program that implements the Insertion sort to sort a given list of integers in ascending order.

Input Format:

- The first line of the input contains an integer n representing the number of elements.
- The second line contains n space-separated integers representing the elements to be sorted.

Output Format:

- The first line will contain the array before sorting.
- The second line will contain the array after sorting using Insertion Sort.

Source Code:

insertionSort.c

```
#include<stdio.h>

void sort(int arr[], int n){
        for(int i=1; i<n;i++){
                int key =arr[i];
                int j= i-1;
                while(j>=0 && arr[j]>key){
                        arr[j+1]=arr[j];
                        j--;
                }
                arr[j+1] = key;
        }
}

void main(){
        int n;
        printf("Enter no of elements: ");
        scanf("%d", &n);
        int arr[n];
        printf("Enter the elements: ");
        for(int i=0; i<n; i++){
                scanf("%d", &arr[i]);
        }
        printf("Array before sort: ");
        for(int i=0; i<n; i++){
                printf("%d ", arr[i]);
        }
        printf("\n");
        sort(arr,n);
        printf("Array after insertion sort: ");
        for(int i=0; i<n;i++){
                printf("%d ", arr[i]);
        }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter no of elements:
6
Enter the elements:

1 5 4 2 6 8
Array before sort: 1 5 4 2 6 8
Array after insertion sort: 1 2 4 5 6 8

Test Case - 2
User Output
Enter no of elements:
8
Enter the elements:
5 2 10 36 95 14 10 23
Array before sort: 5 2 10 36 95 14 10 23
Array after insertion sort: 2 5 10 10 14 23 36 95

S.No: 6	Exp. Name: ***GCD using recursion with Count***	**Date: 2025-02-01**

Aim:

Write a C program to find the Greatest Common Divisor (GCD) of two numbers using recursion and count how many steps are required to execute the recursive function.

The GCD of two numbers is the largest number that divides both of them without leaving a remainder.

Input Format:

- The input consists of two integers, a and b ($0 \leq a, b \leq 100$), separated by a space, for which the GCD is to be found.

Output Format:

- Print the GCD of the two integers.
- Print the number of recursive steps taken to find the GCD.

Source Code:

gcdCount.c

```
#include<stdio.h>

int gcd(int a, int b, int* steps){
        if(b==0){
                return a;
        }
        else{
                (*steps)++;
                return gcd(b, a%b, steps);
        }
}

void main(){
        int a,b,steps=1;
        scanf("%d %d", &a, &b);
        int g = gcd(a,b, &steps);
        printf("%d\n%d\n", g, steps);
}
```

Execution Results - All test cases have succeeded!

ID: 323103310032
2023-2027-CSE-A
Gayatri Vidya Parishad College of Engineering (Autonomous)

Test Case - 1
User Output
48 18
6
4

Test Case - 2
User Output
7 11
1
6

S.No: 7	Exp. Name: ***Quick Sort Algorithm with Count***	**Date: 2025-02-01**

Aim:

Write a C program to implement the Quick Sort algorithm using the divide and conquer strategy. The algorithm should recursively divide an array into smaller sub-arrays based on a chosen pivot element, and then sort these sub-arrays independently. Additionally, the program should count and display the number of steps (comparisons and swaps) required to execute the Quick Sort.

Input Format:

- The input consists of an integer n (1 ≤ n ≤ 100), representing the number of elements in the array.
- The next line contains n space-separated integers which are the elements of the array to be sorted.

Output Format:

- Print the unsorted array.
- Print the sorted array.
- Print the number of steps (comparisons and swaps) required to perform the Quick Sort.

Source Code:

quickSortCount.c

```c
#include <stdio.h>
#include <stdlib.h>

// Swap function
void swap(int *a, int *b, int *steps) {
        int temp = *a;
        *a=*b;
        *b=temp;
        (*steps)++;
}

// Partition function to place the pivot in the correct position
int partition(int arr[], int low, int high, int *steps) {
    int pivort =arr[high];
        int i=low-1;
        for(int j=low;j<high;j++){
                (*steps)++;
                if(arr[j]<pivort){
                        i++;
                        swap(&arr[i], &arr[j],steps);
                }
        }
        swap(&arr[i+1], &arr[high],steps);
        return i+1;
}

// Quick sort function with recursion
void quick_sort(int arr[], int low, int high, int *steps) {
    if(low<high){
                int index=partition(arr, low, high, steps);
                quick_sort(arr,low,index-1,steps);
                quick_sort(arr,index+1,high,steps);
        }
}

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

int main() {
    int n, steps = 0;
```

```
    // Prompt user for the number of elements
    scanf("%d", &n);

    // Allocate memory for the array
    int *arr = (int *)malloc(n * sizeof(int));
    if (arr == NULL) {
        printf("Memory allocation failed");
        return 1;
    }

    // Input array elements from the user
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    // Print unsorted array
    print_array(arr, n);

    // Perform quick sort
    quick_sort(arr, 0, n - 1, &steps);

    // Print sorted array
    print_array(arr, n);

    // Print the number of steps
    printf("%d\n", steps);

    // Free allocated memory
    free(arr);

    return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
5
5 4 3 2 1
5 4 3 2 1
1 2 3 4 5
18

Test Case - 2
User Output
1
100
100
100
0

Test Case - 3
User Output
5
-10 -56 -25 47 -98
-10 -56 -25 47 -98
-98 -56 -25 -10 47
14

S.No: 8	Exp. Name: ***Check connected graph status using DFS in undirected graphs***	**Date: 2025-02-01**

Aim:

You are given **directed graph**, and your task is to check if the given graph is strongly connected or not.

Input Format:

- The first line of input is an integer V, Which represents the number of vertices in the graph.
- The first line of input is an integer E, Which represents the number of edges in the graph.
- The next E lines each contain two integers u and v representing a directed edge from vertex u to vertex v (0-indexed).

Output Format:

- Print **"Strongly connected"** if the graph is strongly connected.
- Print **"Not strongly connected"** if the graph is not strongly connected.

Source Code:

SCGraph.c

ID: 323103310032

Gayatri Vidya Parishad College of Engineering (Autonomous)

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

#define MAX 1000

int adj[MAX][MAX], rev[MAX][MAX], visited[MAX], V;

void dfs(int v, int graph[MAX][MAX]){
        visited[v]=1;
        for(int i=0; i<V; i++){
                if(graph[v][i]&& !visited[i])
                        dfs(i,graph);
        }
}
int isStronglyConnected(){
        memset(visited,0,sizeof(visited));
        dfs(0,adj);
        for(int i=0; i<V; i++)
                if(!visited[i])return 0;
        memset(visited,0,sizeof(visited));
        dfs(0, rev);
        for(int i=0; i<V;i++)
                if(!visited[i])return 0;
        return 1;
}

int main(){
        int E,u,v;
        scanf("%d %d", &V, &E);
        memset(adj, 0,sizeof(adj));
        memset(rev, 0, sizeof(rev));

        for(int i=0; i<E;i++){
                scanf("%d %d", &u, &v);
                adj[u][v]=1;
                rev[v][u]=1;
        }
        printf(isStronglyConnected() ? "Strongly connected\n" : "Not
strongly connected\n");
        return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
4
4
0 1
1 2
2 3
3 0
Strongly connected

Test Case - 2
User Output
4
3
0 1
1 2
2 3
Not strongly connected