1. Naïve string

```c
#include<stdio.h>
#include<string.h>
void search(char text[],char pattern[])
{
    int j;
    int textLength = strlen(text);
    int patternLength = strlen(pattern);
    for(int i = 0 ;i<textLength - patternLength;i++){
        for(j=0 ;j<textLength;j++){
            if(pattern[j]!=text[j+i])
            break;
        }
        if(j==patternLength)
        printf("pattern found at index %d\n",i);

    }
}
int main(){
    char text[] = "abacdabacda";
    char pattern[] = "aba";
    search(text,pattern);
    return 0;
}
```

25. Graph coloring

```c
#include<stdio.h>
#define vertex 4
int main(){
int matrix[vertex][vertex] = {
{0,1,1,0},
{1,0,0,1},
{1,0,0,1},
{0,1,1,0}
};
int colors[] = {1,2};
char vertices[] = {'A','B','C','D'};
for(int i=0;i<vertex;i++){
    printf("The color of vertex %c is %s\n",vertices[i],
    colors[i] == 1? "Red" : "Green"
    );
}


}
```

26. Binary search

```c
#include<stdio.h>
void search(int key, int arr[], int size){
    int low =0 , high = size-1 ,mid;
    int flag = 0 ;
    mid = (low+high) - low/2;
    while(low<=high){
        if(arr[mid] == key){
            flag =1;
            break;
        }
        else if (key < arr[mid]){
            high = mid-1;
        }
        else if(key > arr[mid]){
            low = mid +1;
        }
    }
    if(flag == 1)
    printf("Element is found");
    else
    printf("Element is not found");

}
int main(){
    int element,size;
    printf("Enter the size of the array\n");
    scanf("%d",&size);
    int arr[size];
    printf("Enter the element to be searched\n");
    scanf("%d",&element);
    A:printf("Enter the array elements: \n");
    for(int i=0 ;i<size;i++){
        scanf("%d",&arr[i]);
    }
    for(int i=0;i<size;i++){
        if(arr[i] > arr[i+1]){
            printf("Enter the elements in ascending order\n");
            goto A;
        }
    }
    search(element,arr,size);
    return 0;
}
```

70. Selection sort

```c
#include<stdio.h>
void selectionSort(int arr[] , int size){
```

```
    int i,j,min,temp;
    for(i = 0;i<size;i++){
        min = i;
        for(j=i+1;j<size;j++){
            if(arr[i] > arr[j])
            min = j;
            temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }

    }
    printf("The sorted array is\n");
    for(int k=0;k<size;k++){
        printf("%d ",arr[k]);
    }
}
void insertionSort(int arr[] , int size){
int key,i,j;
for(i=1 ;i<size;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 sorted elements are insertion sort are \n");
for(int k=0;k<size;k++){
    printf("%d ",arr[k]);
}
}
int main(){
    int size;
    printf("Enter the size of the array\n");
    scanf("%d",&size);
    int arr[size];
    printf("Enter the unsorted array elements\n");
    for(int i=0;i<size;i++){
        scanf("%d",&arr[i]);
    }
    insertionSort(arr,size);
    return 0;
}
```

71. Insertion srot (check the selection sort code)

72. LCS

```c
#include<stdio.h>
#include<string.h>
int LCS(char firstString[10],char secondString[10]){
    int i,j,hold[10][10];
    int m = strlen(firstString);
    int n = strlen(secondString);
    for(i = 0 ;i<=m;i++){
        for(j=0;j<=n;j++){
            if(i==0 || j==0){
                hold[i][j] = 0;
            }
            else if(firstString[i-1] == secondString[j-1])
            hold[i][j] = hold[i-1][j-1]+1;
            else
            hold[i][j] = (hold[i-1][j] > hold[i][j-1])?hold[i-1][j] :
   hold[i][j-1];
        }
    }
    return hold[m][n];
}
int main(){
    char firstString[10];
    char secondString[10];
    printf("Enter the first String\n");
    fgets(firstString,sizeof(firstString),stdin);
    printf("Enter the second string\n");
    fgets(secondString,sizeof(secondString),stdin);
    printf("the length of the longest common subsequent is %d
   ",LCS(firstString,secondString));
         return 0;
     }
```

102. Quick srot

```c
    #include<stdio.h>
    void quicksort(int arr[], int low, int high){
        int i,j,pivot;
        if(low<high){
            i= low;
            j=high;
            pivot = low;
            int temp=0;
            while(i<j){
                while(arr[i] <= arr[pivot])
                i++;
                while(arr[j] > arr[pivot])
                j--;
```

```
                if(i<j){
                    temp = arr[i];
                    arr[i] = arr[j];
                    arr[j] = temp;

                }

            }
            temp = arr[j];
            arr[j] = arr[pivot];
            arr[pivot] = temp;

            quicksort(arr,low,j-1);
            quicksort(arr,j+1,high);
        }


    }
    int main(){
        int n;
        printf("Enter the size of the array\n");
        scanf("%d",&n);
        int arr[n];
        printf("enter the array elements\n");
        for(int i=0;i<n;i++){
            scanf("%d",&arr[i]);
        }
        quicksort(arr,0,n-1);
        printf("The elements after sort are \n");
        for(int i=0;i<n;i++){
            printf("%d\n",arr[i]);
        }
        return 0;
    }
```

150. Binary search

```
#include<stdio.h>
void search(int key, int arr[], int size){
    int low =0 , high = size-1 ,mid;
    int flag = 0 ;
    mid = (low+high) - low/2;
    while(low<=high){
        if(arr[mid] == key){
            flag =1;
            break;
        }
        else if (key < arr[mid]){
```

```
            high = mid-1;
        }
        else if(key > arr[mid]){
            low = mid +1;
        }
    }
    if(flag == 1)
    printf("Element is found");
    else
    printf("Element is not found");

}
int main(){
    int element,size;
    printf("Enter the size of the array\n");
    scanf("%d",&size);
    int arr[size];
    printf("Enter the element to be searched\n");
    scanf("%d",&element);
    A:printf("Enter the array elements: \n");
    for(int i=0 ;i<size;i++){
        scanf("%d",&arr[i]);
    }
    for(int i=0;i<size;i++){
        if(arr[i] > arr[i+1]){
            printf("Enter the elements in ascending order\n");
            goto A;
        }
    }
    search(element,arr,size);
    return 0;
}
```

151. Job scheduling
152. Fractional knapsack

```
153.    #include<stdio.h>
154.
155.    void knapsack(int n, int weights[], int profit[], int capacity) {
156.        int i;
157.        float tp = 0.0;
158.        float vectorResult[n];
159.        int u = capacity;
160.
161.        for(i = 0; i < n; i++) {
162.            vectorResult[i] = 0.0;
163.        }
164.
165.        for(i = 0; i < n; i++) {
```

```
166.             if(weights[i] > u)
167.                 break;
168.             else {
169.                 vectorResult[i] = 1.0;
170.                 tp += profit[i];
171.                 u -= weights[i];
172.             }
173.         }
174.
175.         if(i < n) {
176.             vectorResult[i] = (float)u/weights[i];
177.             tp += vectorResult[i] * profit[i];
178.         }
179.
180.         for(i = 0; i < n; i++) {
181.             printf("Result Vector is %.2f\n", vectorResult[i]);
182.         }
183.         printf("The max profit is %.2f\n", tp);
184.     }
185.
186.     int main() {
187.         int n, capacity;
188.         printf("enter the no of items \n");
189.         scanf("%d", &n);
190.         int weights[n], profit[n];
191.         float ratio[n];
192.         printf("Enter the weights\n");
193.         for(int i = 0; i < n; i++) {
194.             scanf("%d", &weights[i]);
195.         }
196.         printf("Enter the profits\n");
197.         for(int i = 0; i < n; i++) {
198.             scanf("%d", &profit[i]);
199.         }
200.         printf("Enter the capacity of knapsack\n");
201.         scanf("%d", &capacity);
202.
203.         for(int i = 0; i < n; i++) {
204.             ratio[i] = (float)profit[i]/weights[i];
205.         }
206.
207.         int temp = 0, temp2 = 0, temp3 = 0;
208.
209.         for(int i = 0; i < n; i++) {
210.             for(int j = i + 1; j < n; j++) {
211.                 if(ratio[i] < ratio[j]) {
212.                     temp = ratio[i];
213.                     ratio[i] = ratio[j];
```

```
                    ratio[j] = temp;

                    temp = profit[i];
                    profit[i] = profit[j];
                    profit[j] = temp;

                    temp = weights[i];
                    weights[i] = weights[j];
                    weights[j] = temp;
                }
            }
        }
        knapsack(n, weights, profit, capacity);
        return 0;
    }
```

229. Floyd warshall

```
#include<stdio.h>
#define nV 4
#define INF 999




void warshall(int matrix[nV][nV]){
    int i,j,k;
    int current[nV][nV];

    // Copy the input matrix to current
    for(i=0; i<nV; i++){
        for(j=0; j<nV; j++){
            current[i][j] = matrix[i][j];
        }
    }

    // Floyd-Warshall Algorithm
    for(k=0; k<nV; k++){
        for(i=0; i<nV; i++){
            for(j=0; j<nV; j++){
                if(current[i][k] + current[k][j] < current[i][j])
                    current[i][j] = current[i][k] + current[k][j];  // Fixed:
Update current[i][j] not current[i][k]
            }
        }
    }
```

```
    // Print the result matrix
    printf("\nShortest distances between every pair of vertices:\n");
    for(i=0; i<nV; i++){
        for(j=0; j<nV; j++){
            if(current[i][j] == INF)
                printf("%4s", "INF");
            else
                printf("%4d", current[i][j]);
        }
        printf("\n");  // Add newline after each row
    }
}

int main(){
    int matrix[nV][nV] = {
        {0,2,INF,1},
        {INF,0,5,3},
        {6,INF,0,8},
        {INF,2,INF,0}
    };
    warshall(matrix);
    return 0;
}
```