

1)Linear search:

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

```
int main(void) {
    int n, val, ind;
    scanf("%d",&n);
    scanf("%d",&val);
    int arr[n];
    for(int i=0;i<n;i++){
        scanf("%d",&arr[i]);
    }
    for(int i=0;i<n;i++){
        if(arr[i]==val)
            ind=i;
    }
    printf("%d",ind);
    return 0;
}
```

2)Binary Search:

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

```
int binarysearch(int arr[],int l, int h, int val){
    if(l<=h){
        int mid=0;
        mid=(l+h)/2;
        if(arr[mid]==val){
            return mid;
        }
        if(arr[mid]>val)
        {
            h=mid-1;
        }
        else{
            l=mid+1;
        }
    }
}
```

```
}
int main(void) {
    int n, val, ind;
    scanf("%d",&n);
    scanf("%d",&val);
    int arr[n];
    for(int i=0;i<n;i++){
        scanf("%d",&arr[i]);
    }
    ind=binarysearch(arr,0,n-1,val);
    printf("%d",ind);
}
```

```

        return 0;
    }

```

3)Bubble sort :

```

#include <stdio.h>
void bubblesort(int arr[],int n){
    for(int i=0;i<n-1;i++){
        for(int j=0;j<n-i-1;j++){
            if(arr[j]>arr[j+1]){
                int t=0;
                t=arr[j];
                arr[j]=arr[j+1];
                arr[j+1]=t;
            }
        }
    }
}
int main(void) {
    int n, val,ind;
    scanf("%d",&n);
    scanf("%d",&val);
    int arr[n];
    for(int i=0;i<n;i++){
        scanf("%d",&arr[i]);
    }
    bubblesort(arr,n-1);
    for(int i=0;i<n;i++){
        printf("%d",arr[i]);
    }

    return 0;
}

```

4)MergeSort

```

#include <stdio.h>

void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;

    /* create temp arrays */
    int L[n1], R[n2];

    for (i = 0; i < n1; i++)

```

```

        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    i = 0;
    j = 0;
    k = l;
    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 l,int h){
    int mid;
    if(l<h){
        mid=(l+h)/2;
        mergesort(arr,l,mid);
        mergesort(arr,mid+1,h);
        merge(arr,l,mid,h);
    }
}

int main(void) {
    int n, val,ind;
    scanf("%d",&n);
    scanf("%d",&val);

```

```

int arr[n];
for(int i=0;i<n;i++){
    scanf("%d",&arr[i]);
}
mergesort(arr,0,n-1);
for(int i=0;i<n;i++){
    printf("%d",arr[i]);
}

    return 0;
}

```

5)Quicksort:

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

```

void swap(int* a, int* b)
{
    int t = *a;
    *a = *b;
    *b = t;
}

```

```

int partition (int arr[], int low, int high)
{
    int pivot = arr[high]; // pivot
    int i = (low - 1);

    for (int j = low; j <= high - 1; j++)
    {

        if (arr[j] < pivot)
        {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}

```

```

void quicksort(int arr[],int l,int h){
    int mid;
    if(l<h){
        mid=(partition(arr,l,h));

        quicksort(arr,l,mid-1);
    }
}

```

```

        quicksort(arr,mid+1,h);
    }
}
int main(void) {
int n, val,ind;
scanf("%d",&n);
scanf("%d",&val);
int arr[n];
for(int i=0;i<n;i++){
    scanf("%d",&arr[i]);
}
quicksort(arr,0,n-1);
for(int i=0;i<n;i++){
    printf("%d",arr[i]);
}
    return 0;
}

```

6)MinMax Algo:

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

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

```
int array[100];
```

```
int min, max;
```

```
// Min Max using divide and conquer
```

```
void min_max(int low, int high)
```

```

{
    if (low == high)
    {
        min = max = array[low];
    }
    else
    {
        if( low == high - 1)
        {
            min = array[low] < array[high] ? array[low] : array[high];
            max = array[low] > array[high] ? array[low] : array[high];
        }
        else
        {
            int mid = (low + high) / 2;
            min_max(low, mid);
            int max1 = max;
            int min1 = min;
            min_max(mid + 1, high);

```

```

        min = min1 < min ? min1 : min;
        max = max1 > max ? max1 : max;
    }
}
}

// Main function
int main()
{
    int n;
    printf("Enter the number of elements in the array: ");
    scanf("%d", &n);
    printf("Enter the elements of the array: ");
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &array[i]);
    }
    min_max(0, n - 1);
    printf("Minimum: %d\n", min);
    printf("Maximum: %d\n", max);
    return 0;
}

```

7)Graph coloring:

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

```

int m, n;
int c = 0;
int count = 0;
int g[50][50];
int x[50];

```

```

void nextValue(int k);
void GraphColoring(int k);

```

```

void main()
{
    int i, j;
    int temp;
    printf("\nEnter the number of nodes: ");
    scanf("%d", &n);
    printf("\nEnter Adjacency Matrix:\n");
    for (i = 1; i <= n; i++)
    {
        for (j = 1; j <= n; j++)
        {
            scanf("%d", &g[i][j]);

```

```

    }
}
printf("\nPossible Solutions are\n");
for (m = 1; m <= n; m++)
{
    if (c == 1)
    {
        break;
    }
    GraphColoring(1);
}
printf("\nThe chromatic number is %d", m - 1);
printf("\nThe total number of solutions is %d", count);
}

```

```

void GraphColoring(int k)
{
    int i;
    while (1)
    {
        nextValue(k);
        if (x[k] == 0)
        {
            return;
        }
        if (k == n)
        {
            c = 1;
            for (i = 1; i <= n; i++)
            {
                printf("%d ", x[i]);
            }
            count++;
            printf("\n");
        }
        else
        {
            GraphColoring(k + 1);
        }
    }
}

```

```

void nextValue(int k)
{
    int j;
    while (1)
    {
        x[k] = (x[k] + 1) % (m + 1);

```

```

    if (x[k] == 0)
    {
        return;
    }
    for (j = 1; j <= n; j++)
    {
        if (g[k][j] == 1 && x[k] == x[j])
        {
            break;
        }
    }
    if (j == (n + 1))
    {
        return;
    }
}
}

```

8)N-Queen:

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

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

```
#define N 5
```

```

int isSafe(int board[N][N], int row, int col)
{
    int tempRow = row, tempCol = col;
    while (tempRow >= 0 && tempCol >= 0)
    {
        if (board[tempRow][tempCol] == 1)
            return 0;
        tempRow--;
        tempCol--;
    }
    tempRow = row;
    tempCol = col;
    while (tempCol >= 0)
    {
        if (board[tempRow][tempCol] == 1)
            return 0;
        tempCol--;
    }
    tempRow = row;
    tempCol = col;
    while (tempRow < N && tempCol >= 0)
    {
        if (board[tempRow][tempCol] == 1)
            return 0;
        tempRow++;
    }
}

```



```

        tempCol--;
    }
    return 1;
}

int nQueens(int board[N][N], int col)
{
    int i, j;
    if (col >= N)
        return 1;

    for (int i = 0; i < N; i++)
    {
        if (isSafe(board, i, col))
        {
            board[i][col] = 1;
            if (nQueens(board, col + 1))
                return 1;
            board[i][col] = 0;
        }
    }
    return 0;
}

int main()
{
    int board[N][N] = {0};
    int col = 0;
    int solutionExists = nQueens(board, col);
    if (solutionExists)
    {
        for (int i = 0; i < N; i++)
        {
            for (int j = 0; j < N; j++)
                printf("%d ", board[i][j]);
            printf("\n");
        }
    }
    else
        printf("No solution");

    return 0;
}

```

9)Dijkstra:

```

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

```

```

#define MAX 10
#define INF 9999

// Dijkstra's Algorithm
void dijkstra(int graph[MAX][MAX], int n, int start)
{
    int dist[MAX] = {0},
        visited[MAX] = {0},
        path[MAX] = {0},
        u, v, w, i, j, k, min;
    for (i = 0; i < n; i++)
    {
        dist[i] = graph[start][i];
        path[i] = start;
    }
    visited[start] = 1;
    for (i = 0; i < n - 1; i++)
    {
        min = INF;
        for (j = 0; j < n; j++)
        {
            if (!visited[j] && dist[j] < min)
            {
                min = dist[j];
                u = j;
            }
        }
        visited[u] = 1;
        for (j = 0; j < n; j++)
        {
            if (!visited[j] && graph[u][j] < INF)
            {
                w = graph[u][j];
                if (dist[u] + w < dist[j])
                {
                    dist[j] = dist[u] + w;
                    path[j] = u;
                }
            }
        }
    }
}

printf("\n Shortest Path from %d: ", start);
printf("\n");
for (i = 0; i < n; i++)
{
    printf(" %d <- ", path[i]);
}

```

```

    printf("\n");
}

// Main function
int main()
{
    int graph[MAX][MAX], i, j, n, start;
    printf("\n Enter the number of vertices: ");
    scanf("%d", &n);
    printf("\n Enter the adjacency matrix:\n");
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
        {
            scanf("%d", &graph[i][j]);
            if (graph[i][j] == 0)
                graph[i][j] = INF;
        }
    printf("\n Enter the starting node: ");
    scanf("%d", &start);
    djikstra(graph, n, start);
    return 0;
}

```

10)

Fractional_Knapsacks:

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

```
void knapsack(int n, float weight[], float profit[], float capacity)
```

```

{
    float x[20], tp = 0;
    int i, j, u;
    u = capacity;

    for (i = 0; i < n; i++)
        x[i] = 0.0;

    for (i = 0; i < n; i++)
    {
        if (weight[i] > u)
            break;
        else
        {
            x[i] = 1.0;
            tp = tp + profit[i];
            u = u - weight[i];
        }
    }
}

```

```
if (i < n)
```

```

        x[i] = u / weight[i];

        tp = tp + (x[i] * profit[i]);

        printf("\nMaximum profit is:- %f", tp);
    }

int main()
{
    float weight[20], profit[20], capacity;
    int num, i, j;
    float ratio[20], temp;

    printf("\nEnter the no. of objects:- ");
    scanf("%d", &num);

    printf("\nEnter the wts and profits of each object:- ");
    for (i = 0; i < num; i++)
    {
        scanf("%f %f", &weight[i], &profit[i]);
    }

    printf("\nEnter the capacity of knapsack:- ");
    scanf("%f", &capacity);

    for (i = 0; i < num; i++)
    {
        ratio[i] = profit[i] / weight[i];
    }

    for (i = 0; i < num; i++)
    {
        for (j = i + 1; j < num; j++)
        {
            if (ratio[i] < ratio[j])
            {
                temp = ratio[j];
                ratio[j] = ratio[i];
                ratio[i] = temp;

                temp = weight[j];
                weight[j] = weight[i];
                weight[i] = temp;

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

```

```
    }  
}  
  
knapsack(num, weight, profit, capacity);  
return (0);  
}
```