Insertion Sort Algorithm

```
#include <stdio.h>
void insertionSort(int arr[], int n)
{
  int i, key, j;
  for (i = 1; i < n; 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;
  }
}
void printArray(int arr[], int n)
{
  int i;
  for (i = 0; i < n; i++)
    printf("%d ", arr[i]);
  printf("\n");
}
int main()
{
  int arr[] = {12, 11, 13, 5, 6};
  int n = sizeof(arr) / sizeof(arr[0]);
  printArray(arr, n);
  insertionSort(arr, n);
  printArray(arr, n);
  return 0;
}
```

Merge Sort Algorithm

```
#include <stdio.h>

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

    int L[n1], R[n2];

    for (int i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (int j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    int 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 r)
{
    if (l < r)
    {
        int m = l + (r - l) / 2;

        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);

        merge(arr, l, m, r);
    }
}
```

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

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

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

    mergeSort(arr, 0, arr_size - 1);

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

    return 0;
}
```

Quick Sort Algorithm

```
#include <stdio.h>

#define N 5

int arr[N];

int partition(int low, int high) {
    int pivot = arr[high], i = low - 1, temp;
    int j; // Declared outside the loop
    for (j = low; j < high; j++) {
        if (arr[j] < pivot) {
            i++;
            temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
    temp = arr[i + 1];
    arr[i + 1] = arr[high];
    arr[high] = temp;
    return i + 1;
}

void quickSort(int low, int high) {
    int pi;
    if (low < high) {
        pi = partition(low, high);
        quickSort(low, pi - 1);
        quickSort(pi + 1, high);
    }
}

int main() {
    int i; // Declared outside the loop
    printf("Enter 5 numbers: ");
    for (i = 0; i < N; i++) scanf("%d", &arr[i]);

    quickSort(0, N - 1);

    printf("Sorted array: ");
    for (i = 0; i < N; i++) printf("%d ", arr[i]);
    return 0;
}
```

Dijkstra Algorithm

```
#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];
    int i = (low - 1);

    for (int j = low; j < high; 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 low, int high)
{
    if (low < high)
                    {
        int pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

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

int main()
{
    int arr[] = {10, 7, 8, 9, 1, 5};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("Original array:\n");
    printArray(arr, n);
    quickSort(arr, 0, n - 1);
    printf("Sorted array:\n");
    printArray(arr, n);

    return 0;
}
```

Prims Algorithm

```
#include <stdio.h>
#include <limits.h>
#define V 5
int minKey(int key[], int mstSet[])
{
   int min = INT_MAX, minIndex, v;
   for (v = 0; v < V; v++) {
      if (mstSet[v] == 0 && key[v] < min)
     {
         min = key[v];
         minIndex = v;
      }
   }
   return minIndex;
}

void printMST(int parent[], int graph[V][V])
 {
   int i;
   for (i = 1; i < V; i++)
  {
      printf("%d - %d   %d\n", parent[i], i, graph[i][parent[i]]);
   }
}

void primMST(int graph[V][V])
{
   int parent[V], key[V], mstSet[V];
   int count, u, v,i;
   for (i = 0; i < V; i++)
  {
      key[i] = INT_MAX;
      mstSet[i] = 0;
   }
   key[0] = 0;
   parent[0] = -1;
   for (count = 0; count < V - 1; count++)
  {
      u = minKey(key, mstSet);
      mstSet[u] = 1;
      for (v = 0; v < V; v++) {
         if (graph[u][v] && mstSet[v] == 0 && graph[u][v] < key[v])
        {
            key[v] = graph[u][v];
            parent[v] = u;
         }
      }
   }
   printMST(parent, graph);
}

int main() {
   int graph[V][V] = {
      {0, 2, 0, 6, 0},
      {2, 0, 3, 8, 5},
      {0, 3, 0, 0, 7},
      {6, 8, 0, 0, 9},
      {0, 5, 7, 9, 0}
   };
   primMST(graph);
   return 0;
}
```

Fractional Knapsack problem

```
#include <stdio.h>

#define N 5

struct Item {
    int weight;
    int value;
};

int compare(const void *a, const void *b) {
    double r1 = ((struct Item *)a)->value / (double)((struct Item *)a)->weight;
    double r2 = ((struct Item *)b)->value / (double)((struct Item *)b)->weight;
    return (r1 < r2) - (r1 > r2);
}

double fractionalKnapsack(struct Item arr[], int W) {
    int i;
    double totalValue = 0.0;
    for (i = 0; i < N; i++) {
        if (arr[i].weight <= W) {
            totalValue += arr[i].value;
            W -= arr[i].weight;
        } else {
            totalValue += arr[i].value * ((double) W / arr[i].weight);
            break;
        }
    }
    return totalValue;
}

int main() {
    struct Item arr[N] = {{60, 10}, {100, 20}, {120, 30}, {90, 10}, {70, 15}};
    int W = 50;  // Knapsack capacity
    qsort(arr, N, sizeof(arr[0]), compare);
    printf("Maximum value in Knapsack = %.2f\n", fractionalKnapsack(arr, W));
    return 0;
}
```

Bellman Ford Algorithm

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

#define V 5
#define E 8

struct Edge {
    int u, v, weight;
};

void bellmanFord(struct Edge edges[], int src) {
    int dist[V], i, j;

    for (i = 0; i < V; i++)
        dist[i] = INT_MAX;
    dist[src] = 0;

    for (i = 1; i < V; i++) {
        for (j = 0; j < E; j++) {
            int u = edges[j].u;
            int v = edges[j].v;
            int weight = edges[j].weight;
            if (dist[u] != INT_MAX && dist[u] + weight < dist[v])
                dist[v] = dist[u] + weight;
        }
    }

    for (i = 0; i < E; i++) {
        int u = edges[i].u;
        int v = edges[i].v;
        int weight = edges[i].weight;
        if (dist[u] != INT_MAX && dist[u] + weight < dist[v])
            printf("Graph contains negative weight cycle\n");
    }

    printf("Vertex Distance from Source %d\n", src);
    for (i = 0; i < V; i++)
        printf("%d \t\t %d\n", i, dist[i]);
}

int main() {
    struct Edge edges[E] = {
        {0, 1, -1}, {0, 2, 4}, {1, 2, 3}, {1, 3, 2}, {1, 4, 2}, {3, 2, 5}, {3, 1, 1}, {4, 3, -3}
    };
    bellmanFord(edges, 0);
    return 0;
}
```

Travelling salesperson problem

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

#define N 4

int dist[N][N] = {
   {0, 10, 15, 20},
   {10, 0, 35, 25},
   {15, 35, 0, 30},
   {20, 25, 30, 0}
};

int dp[1 << N][N];

int tsp(int mask, int pos) {
   int city, newAns, ans;
   if (mask == (1 << N) - 1) {
      return dist[pos][0]; // Return to the starting point
   }
   if (dp[mask][pos] != -1) return dp[mask][pos];

   ans = INT_MAX;
   for (city = 0; city < N; city++) {
      if ((mask & (1 << city)) == 0) {
         newAns = dist[pos][city] + tsp(mask | (1 << city), city);
         ans = (ans < newAns) ? ans : newAns;
      }
   }
   return dp[mask][pos] = ans;
}

int main() {
   int i, j;
   for (i = 0; i < (1 << N); i++)
      for (j = 0; j < N; j++)
         dp[i][j] = -1;

   printf("Minimum cost: %d\n", tsp(1, 0));
   return 0;
}
```

N-queen problem

```
#include <stdio.h>
#define N 4

int board[N][N];
int isSafe(int r, int c)
{
    int i, j;
    for (i = 0; i < r; i++)
    {
        if (board[i][c]) return 0;
    }
    for (i = r - 1, j = c - 1; i >= 0 && j >= 0; i--, j--)
    {
        if (board[i][j]) return 0;
    }
    for (i = r - 1, j = c + 1; i >= 0 && j < N; i--, j++)
    {
        if (board[i][j]) return 0;
    }
    return 1;
}

int solve(int r)
{
    int c, i, j;
    if (r == N) {
        for (i = 0; i < N; i++)
        {
            for (j = 0; j < N; j++)
            {
                printf("%d ", board[i][j]);
            }
            printf("\n");
        }
        return 1;
    }
    for (c = 0; c < N; c++)
    {
        if (isSafe(r, c))
        {
            board[r][c] = 1;
            if (solve(r + 1)) return 1;
            board[r][c] = 0;
        }
    }
    return 0;
}

int main()
{
    solve(0);
    return 0;
}
```

Rabin Karp Algorithm

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

#define d 256
#define q 101
int hash(char s[], int l, int m)
 {
  int h = 0;
  int i;
  for (i = 0; i < l; i++)
    h = (h * d + s[i]) % m;
  return h;
}
void rk(char t[], char p[]) {
  int m = strlen(p);
  int n = strlen(t);
  int ph = hash(p, m, q);
  int th = hash(t, m, q);
  int h = 1;
  int i, j;
  for (i = 0; i < m - 1; i++)
    h = (h * d) % q;

  for (i = 0; i <= n - m; i++) {
    if (ph == th) {
      j = 0;
      while (j < m && t[i + j] == p[j])
        j++;
      if (j == m)
        printf("Pattern found at index %d\n", i);
    }
    if (i < n - m) {
      th = (d * (th - t[i] * h) + t[i + m]) % q;
      if (th < 0)
        th += q;
    }
  }
}

int main() {
  char t[100], p[100];
  printf("Enter text: ");
  scanf("%s", t);  // Takes input for text
  printf("Enter pattern: ");
  scanf("%s", p);  // Takes input for pattern
  rk(t, p);
  return 0;
}
```