Insertion Sort

```
#include <stdio.h>

int n, i, j, A[20], key;

void main()
{
   printf("Enter the size of array: ");
   scanf("%d", &n);
   printf("Enter the elements of array: \n");
   for(i=0; i<n; i++)
   {
      printf("Enter value: ");
      scanf("%d", &A[i]);
   }
   printf("The unsorted array is: ");
   for(i=0; i<n; i++)
   {
      printf("%d\t",A[i]);
   }
   Insertion_Sort(A,n);
   printf("\nAfter sorting array is: ");
   for(i=0; i<n; i++)
   {
      printf("%d\t",A[i]);
   }
}


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

Floyd Warshall

```
#include <stdio.h>
#define INF 99999

int n, i, j, k;
int graph[20][20];

void floydWarshall();

void main()
{
   printf("Enter the number of vertices: ");
   scanf("%d", &n);

   printf("Enter the adjacency matrix (Enter %d for
infinity):\n", INF);
   for (i = 0; i < n; i++)
   {
      for (j = 0; j < n; j++)
      {
         scanf("%d", &graph[i][j]);
      }
   }

   floydWarshall();
}

void floydWarshall()
{
   int dist[20][20];

   for (i = 0; i < n; i++)
   {
      for (j = 0; j < n; j++)
      {
         dist[i][j] = graph[i][j];
      }
   }

   for (k = 0; k < n; k++)
   {
      for (i = 0; i < n; i++)
      {
         for (j = 0; j < n; j++)
         {
            if (dist[i][k] != INF && dist[k][j] != INF &&
dist[i][k] + dist[k][j] < dist[i][j]) {
               dist[i][j] = dist[i][k] + dist[k][j];
            }
         }
      }
   }

   printf("\nShortest Distance Matrix:\n");
   for (i = 0; i < n; i++)
   {
      for (j = 0; j < n; j++)
```

```
    {
      if (dist[i][j] == INF)
        printf("INF\t");
      else
        printf("%d\t", dist[i][j]);
    }
    printf("\n");
  }
}
```

Fractional Knapsack

```
#include <stdio.h>

struct Item
{
  int weight, value;
  float ratio;
};

int main() {
  int n, capacity;
  printf("Enter number of items: ");
  scanf("%d", &n);

  struct Item items[n];
  printf("Enter weight and value of each item:\n");
  for (int i = 0; i < n; i++)
  {
    scanf("%d %d", &items[i].weight,
&items[i].value);
    items[i].ratio = (float)items[i].value /
items[i].weight;
  }

  for (int i = 1; i < n; i++) {
    struct Item temp = items[i];
    int j = i - 1;
    while (j >= 0 && items[j].ratio < temp.ratio)
    {
      items[j + 1] = items[j--];
    }
    items[j + 1] = temp;
  }

  printf("Enter knapsack capacity: ");
  scanf("%d", &capacity);

  float totalValue = 0;
  printf("\nItems taken:\n");
  for (int i = 0; i < n && capacity > 0; i++)
  {
    if (items[i].weight <= capacity)
    {
      printf("Full item: W=%d V=%d\n",
items[i].weight, items[i].value);
      totalValue += items[i].value;
      capacity -= items[i].weight;
    }
    else
    {
      float fraction = (float)capacity /
items[i].weight;
      printf("Fraction %.2f of item: W=%d V=%d\n",
fraction, items[i].weight, items[i].value);
      totalValue += items[i].value * fraction;
      break;
    }
  }

  printf("Total value = %.2f\n", totalValue);
  return 0;
}
```

Kruskal' s Algorithm

```
#include <stdio.h>

struct Edge {
  int src, dest, weight;
};

struct Edge edges[20], result[20];
int n, m, i, j;
int parent[20];

void kruskalMST();
int find(int);
void unionSets(int, int);

void main() {
  printf("Enter the number of vertices: ");
  scanf("%d", &n);

  printf("Enter the number of edges: ");
  scanf("%d", &m);

  printf("Enter edges (source, destination,
weight):\n");
  for (i = 0; i < m; i++) {
    scanf("%d %d %d", &edges[i].src, &edges[i].dest,
&edges[i].weight);
  }

  kruskalMST();
}
```

```
int find(int v) {
   if (parent[v] == v)
      return v;
   return find(parent[v]);
}

void unionSets(int u, int v) {
   parent[v] = u;
}

void kruskalMST() {
   struct Edge key;
   int cost = 0, edgeCount = 0;

   for (i = 1; i < m; i++) {
      key = edges[ i];
      j = i - 1;
      while (j >= 0 && edges[j].weight > key.weight) {
         edges[j + 1] = edges[j];
         j = j - 1;
      }
      edges[j + 1] = key;
   }

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

   printf("\nMinimum Spanning Tree (MST):\n");
   printf("Edge \tWeight\n");

   for (i = 0; i < m && edgeCount < n - 1; i++) {
      int u = find(edges[i].src);
      int v = find(edges[i].dest);
      if (u != v) {
         result[edgeCount++] = edges[i];
         unionSets(u, v);
         cost += edges[i].weight;
         printf("%d - %d\t%d\n", edges[i].src,
edges[i].dest, edges[i].weight);
      }
   }

   printf("Total Minimum Cost: %d\n", cost);
}
```

Merge Sort

```
#include <stdio.h>

int n, A[20];

void Merge_Sort(int A[], int low, int high);
void Combine(int A[], int low, int mid, int high);

int main() {
   int i;
   printf("Enter the size of array: ");
   scanf("%d", &n);

   printf("Enter the elements of array:\n");
   for (i = 0; i < n; i++) {
      printf("Enter value: ");
      scanf("%d", &A[i]);
   }

   printf("\nThe unsorted array is: \n");
   for (i = 0; i < n; i++) {
      printf("%d\t", A[i]);
   }

   Merge_Sort(A, 0, n - 1);

   printf("\nAfter sorting, the array is: \n");
   for (i = 0; i < n; i++) {
      printf("%d\t", A[i]);
   }

   return 0;
}

// Recursive Merge Sort
void Merge_Sort(int A[], int low, int high) {
   if (low < high) {
      int mid = (low + high) / 2;
      Merge_Sort(A, low, mid);
      Merge_Sort(A, mid + 1, high);
      Combine(A, low, mid, high);
   }
}

void Combine(int A[], int low, int mid, int high) {
   int i = low, j = mid + 1, k = 0;
   int temp[high - low + 1];

   while (i <= mid && j <= high) {
      if (A[i] <= A[j])
         temp[k++] = A[i++];
      else
         temp[k++] = A[j++];
   }

   while (i <= mid)
      temp[k++] = A[i++];

   while (j <= high)
      temp[k++] = A[j++];

   for (i = low, k = 0; i <= high; i++, k++)
      A[i] = temp[k];
}
```

N – queen

```
#include <stdio.h>

int n, board[10][10];

void printSolution();
int isSafe(int row, int col);
int solveNQueen(int col);

void main() {
   printf("Enter the number of queens: ");
   scanf("%d", &n);

   for (int i = 0; i < n; i++)
      for (int j = 0; j < n; j++)
         board[i][j] = 0;

   if (solveNQueen(0))
      printSolution();
   else
      printf("Solution does not exist.\n");
}

int isSafe(int row, int col) {
   int i, j;

   for (i = 0; i < col; i++)
   {
      if (board[row][i])
         return 0;
   }

   for (i = row, j = col; i >= 0 && j >= 0; i--, j--)
   {
      if (board[i][j])
         return 0;
   }

   for (i = row, j = col; i < n && j >= 0; i++, j--)
      if (board[i][j])
         return 0;

   return 1;
}

int solveNQueen(int col) {
   if (col >= n)
      return 1;

   for (int i = 0; i < n; i++) {
      if (isSafe(i, col)) {
         board[i][col] = 1;

         if (solveNQueen(col + 1))
            return 1;

         board[i][col] = 0;
      }
   }

   return 0;
}

void printSolution() {
   printf("\nSolution:\n");
   for (int i = 0; i < n; i++)
   {
      for (int j = 0; j < n; j++)
      {
         if (board[i][j] == 1)
            printf("Q\t");
         else
            printf(".\t");
      }
      printf("\n");
   }
}
```

Naïve String

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

char text[100], pattern[20];
int n, m, i, j;

void naiveStringMatch();

void main() {
   printf("Enter the text: ");
   gets(text);

   printf("Enter the pattern: ");
   gets(pattern);

   naiveStringMatch();
}

void naiveStringMatch() {
   n = strlen(text);
   m = strlen(pattern);

   printf("Pattern found at positions: ");
   int found = 0;

   for (i = 0; i <= n - m; i++) {
      for (j = 0; j <= m-1; j++) {
         if (text[i + j] != pattern[j])
            break;
      }
```

```
      if (j == m) {
         printf("%d ", i);
         found = 1;
      }
   }

   if (!found)
      printf("No match found.");

   printf("\n");
}
```

Prims Algorithm

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

int n, i, j;
int graph[20][20], parent[20], key[20], mstSet[20];

void primMST();
int minKey();

void main() {
   printf("Enter the number of vertices: ");
   scanf("%d", &n);

   printf("Enter the adjacency matrix (0 for no
edge):\n");
   for (i = 0; i < n; i++) {
      for (j = 0; j < n; j++) {
         scanf("%d", &graph[i][j]);
      }
   }

   primMST();
}

int minKey() {
   int min = INT_MAX, minIndex;
   for (i = 0; i < n; i++) {
      if (mstSet[i] == 0 && key[i] < min) {
         min = key[i];
         minIndex = i;
      }
   }
   return minIndex;
}


void primMST() {
   for (i = 0; i < n; i++) {
      key[i] = INT_MAX;
      mstSet[i] = 0;
   }

   key[0] = 0;
   parent[0] = -1;

   for (i = 0; i < n - 1; i++) {
      int u = minKey();
      mstSet[u] = 1;

      for (j = 0; j < n; j++) {
         if (graph[u][j] && mstSet[j] == 0 && graph[u][j]
< key[j]) {
            parent[j] = u;
            key[j] = graph[u][j];
         }
      }
   }

   printf("\nMinimum Spanning Tree (MST):\n");
   printf("Edge \tWeight\n");
   for (i = 1; i < n; i++) {
      printf("%d - %d\t%d\n", parent[i], i,
graph[i][parent[i]]);
   }
}
```

Quick Sort

```
#include <stdio.h>

int n, A[20];

void Quick_Sort(int A[], int low, int high);
int Partition(int A[], int low, int high);

int main() {
   int i;
   printf("Enter the size of array: ");
   scanf("%d", &n);

   printf("Enter the elements of array:\n");
   for (i = 0; i < n; i++) {
      printf("Enter value: ");
      scanf("%d", &A[i]);
   }

   printf("\nThe unsorted array is: \n");
   for (i = 0; i < n; i++) {
      printf("%d\t", A[i]);
   }

   Quick_Sort(A, 0, n - 1);
```

```
   printf("\nAfter sorting, the array is: \n");
   for (i = 0; i < n; i++) {
      printf("%d\t", A[i]);
   }

   return 0;
}

void Quick_Sort(int A[], int low, int high) {
   if (low < high) {
      int pivotIndex = Partition(A, low, high);
      Quick_Sort(A, low, pivotIndex - 1);
      Quick_Sort(A, pivotIndex + 1, high);
   }
}

int Partition(int A[], int low, int high) {
   int pivot = A[high];
   int i = low - 1;

   for (int j = low; j < high; j++) {
      if (A[j] < pivot) {
         i++;
         int temp = A[i];
         A[i] = A[j];
         A[j] = temp;
      }
   }

   int temp = A[i + 1];
   A[i + 1] = A[high];
   A[high] = temp;

   return i + 1;
}
```

Selection Sort

```
#include <stdio.h>
int n, i, j, A[20], minIndex;

void Selection_Sort(int A[], int n);
void Swap(int A[], int i, int minIndex);


void main()
{
   int i, j;
   printf("Enter the size of array: ");
   scanf("%d", &n);
   printf("Enter the elements of array: \n");
   for(i=0; i<n; i++)
   {
      printf("Enter value: ");
      scanf("%d", &A[i]);
   }
   printf("The unsorted array is: ");
   for(i=0; i<n; i++)
   {
      printf("%d\t",A[i]);
   }
   Selection_Sort(A,n);
   printf("\nAfter sorting array is: ");
   for(i=0; i<n; i++)
   {
      printf("%d\t",A[i]);
   }
}


void Selection_Sort(int A[], int n)
{
   for(i= 0; i<=n-2; i++)
   {
      minIndex = i;
      for(j=i+1; j<= n-1; j++)
      {
         if(A[j]< A[minIndex])
         {
            minIndex = j;
         }
      }
      Swap(A, i, minIndex);
   }
}


void Swap(int A[], int i, int minIndex)
{
   int temp;
   temp = A[i];
   A[i] = A[minIndex];
   A[minIndex] = temp;
}
```