1.N-Queens Problem

```
#include <stdio.h>
#include <math.h>
int m[20], count;
int main()
{
   int n,i,j;
   void queen(int row, int n);
   printf ("Enter the number of Queen: ");
   scanf("%d", &n);
   queen(1, n);
   return 0;
}
void print(int n){
   int i, j;
   printf("\nSolution %d:\n",++count);
   for(i=1; i<=n; ++i)
      printf("\t%d", i);
   for(i=1; i<=n; ++i){
      printf("\n%d", i);
      for(j=1;j<=n;++j){
         if(m[i]==j)
            printf("\tQ");
         else
            printf("\t-");
      }
   }
}
```

```
int place(int row, int column)
{
   int i;
   for(i=1;i<=row-1;i++){
      if(m[i]==column)
         return 0;
      else if(abs(m[i]-column)==abs(i-row))
         return 0;
   }
   return 1;
}
void queen(int row, int n){
   int column;
   for(column=1;column<=n;++column){
      if(place(row, column)){
         m[row]=column;
         if(row==n)
            print(n);
         else
            queen(row+1,n);
      }
   }
}
```

2. Prims Algorithm

```
#include <stdio.h>
#define MAX 20
```

```
#define INF 999 // Use a constant for infinity
int main() {
    int n, i, j, cost[MAX][MAX], visited[MAX] = {0};
    int edge = 1, min, totalCost = 0;
    int node1, node2, u, v;
    printf("Enter the number of nodes: ");
    scanf("%d", &n);
    printf("Enter the adjacency matrix: \n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            scanf("%d", &cost[i][j]);
            if (cost[i][j] == 0) {
                cost[i][j] = INF;
            }
        }
    }
    visited[0] = 1;
    while (edge < n) {
        min = INF;
        for (i = 0; i < n; i++) {
            for (j = 0; j < n; j++) {
                if (cost[i][j] < min && visited[i]) {
                    min = cost[i][j];
                    node1 = u = i;
                    node2 = v = j;
                }
            }
        }
```

```
      if (visited[u] == 0 || visited[v] == 0) {
         printf("Edge %d: (%d - %d) = %d\n", edge++, node1 + 1, node2 +
1, min);
         totalCost += min;
         visited[node2] = 1;
      }
      cost[node1][node2] = cost[node2][node1] = INF;
   }
   printf("The minimum cost = %d\n", totalCost);
   return 0;
}
```

3. Kruskal's Algorithm

```
#include <stdio.h>
#define MAX 100
typedef struct {
   int u, v, w;
} Edge;
Edge edges[MAX];
int parent[MAX], n, e;
int find(int i) {
   return (parent[i] == i) ? i : find(parent[i]);
}
void unionSets(int u, int v) {
   parent[u] = v;
}
void sortEdges() {
```

```
    for (int i = 0; i < e - 1; i++) {
        for (int j = 0; j < e - i - 1; j++) {
            if (edges[j].w > edges[j + 1].w) {
                Edge temp = edges[j];
                edges[j] = edges[j + 1];
                edges[j + 1] = temp;
            }
        }
    }
}
void kruskal() {
    int mst_weight = 0;
    sortEdges();
    for (int i = 0; i < n; i++) parent[i] = i;
    for (int i = 0; i < e; i++) {
        int u = find(edges[i].u);
        int v = find(edges[i].v);
        if (u != v) {
            printf("Edge (%d, %d) -> Weight: %d\n", edges[i].u, edges[i].v,
edges[i].w);
            mst_weight += edges[i].w;
            unionSets(u, v);
        }
    }
    printf("Total weight of MST: %d\n", mst_weight);
}
int main() {
    printf("Enter the number of vertices: ");
```

```
    scanf("%d", &n);
    printf("Enter the number of edges: ");
    scanf("%d", &e);
    // Input edges
    for (int i = 0; i < e; i++) {
        printf("Enter edge (u, v, w): ");
        scanf("%d %d %d", &edges[i].u, &edges[i].v, &edges[i].w);
    }
    // Run Kruskal's algorithm
    kruskal();
    return 0;
}
```

4. Longest common subsequence (dynamic programming)

```
#include <stdio.h>
#include <string.h>
#define MAX 100
// Function to find the length of the LCS
int lcs(char *X, char *Y, int m, int n) {
    int L[MAX][MAX];
    // Build the LCS table in bottom-up manner
    for (int i = 0; i <= m; i++) {
        for (int j = 0; j <= n; j++) {
            if (i == 0 || j == 0)
                L[i][j] = 0;
            else if (X[i - 1] == Y[j - 1])
```

```
            L[i][j] = L[i - 1][j - 1] + 1;
        else
            L[i][j] = (L[i - 1][j] > L[i][j - 1]) ? L[i - 1][j] : L[i][j - 1];
      }
   }
   return L[m][n]; // Length of the LCS is in L[m][n]
}
int main() {
   char X[MAX], Y[MAX];
   // Input two strings
   printf("Enter first string: ");
   scanf("%s", X);
   printf("Enter second string: ");
   scanf("%s", Y);
   int m = strlen(X);
   int n = strlen(Y);
   // Find and print the length of LCS
   printf("Length of LCS: %d\n", lcs(X, Y, m, n));
   return 0;
}
```

5. margesort Algorithm

```
#include <stdio.h>
void merge(int arr[], int l, int m, int r) {
   int i = l, j = m + 1, k = 0;
   int temp[r - l + 1];  // Temporary array
   // Merge the two halves
   while (i <= m && j <= r) {
```

```
        if (arr[i] <= arr[j])
            temp[k++] = arr[i++];
        else
            temp[k++] = arr[j++];
    }
    // Copy remaining elements from left half
    while (i <= m)
        temp[k++] = arr[i++];
    // Copy remaining elements from right half
    while (j <= r)
        temp[k++] = arr[j++];
    // Copy the sorted elements back to original array
    for (i = l, k = 0; i <= r; i++, k++)
        arr[i] = temp[k];
}
void mergeSort(int arr[], int l, int r) {
    if (l < r) {
        int m = (l + r) / 2;
        mergeSort(arr, l, m);    // Sort left half
        mergeSort(arr, m + 1, r); // Sort right half
        merge(arr, l, m, r);      // Merge the sorted halves
    }
}
int main() {
    int n;
    // Input size of array
    printf("Enter number of elements: ");
    scanf("%d", &n);
```

```
    int arr[n];
    // Input array elements
    printf("Enter the elements: ");
    for (int i = 0; i < n; i++)
        scanf("%d", &arr[i]);
    // Sorting the array using merge sort
    mergeSort(arr, 0, n - 1);
    // Print the sorted array
    printf("Sorted array: ");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    return 0;
}
```