1. Minimum Cost Spanning Tree Kruskal's Algorithm

```
#include <stdio.h>
int cost[10][10], n;
void kruskal() {
   int par[10];
   int a = 0, b = 0, u = 0, v = 0, min, mincost = 0, ne = 0;
   for (int i = 0; i < n; i++)
      par[i] = -1;
   printf("The minimum spanning tree edges are...\n");
   while (ne < n - 1) {
      min = 999;
      for (int i = 0; i < n; i++)
         for (int j = 0; j < n; j++)
            if (cost[i][j] < min) {
               min = cost[i][j];
               a = u = i;
               b = v = j;
            }
      while (par[u] != -1)
         u = par[u];
      while (par[v] != -1)
         v = par[v];
      if (u != v) {
 printf("From vertex %d to vertex %d and the cost = %d\n", a, b, min);
         mincost += min;
         par[v] = u;
         ne++;
      }
      cost[a][b] = cost[b][a] = 999;
   }
   printf("Cost of MST = %d\n", mincost);
}
int main() {
   printf("Enter the number of vertices: ");
   scanf("%d", &n);
   printf("Enter the cost matrix\n");
   for (int i = 0; i < n; i++)
      for (int j = 0; j < n; j++)
         scanf("%d", &cost[i][j]);
   kruskal();
   return 0;
}
```

2. Minimum Cost Spanning Tree Prim's algorithm

```
#include <stdio.h>
int cost[10][10], n;
void prim() {
   int vt[10] = {0};
   int a = 0, b = 0, min, mincost = 0, ne = 0;
   vt[0] = 1;
   while (ne < n - 1) {
      min = 999;
      for (int i = 0; i < n; i++) {
         if (vt[i] == 1) {
            for (int j = 0; j < n; j++) {
               if (cost[i][j] < min && vt[j] == 0) {
                  min = cost[i][j];
                  a = i;
                  b = j;
               }
            }
         }
      }
      printf("Edge from vertex %d to vertex %d and the cost %d\n", a, b, min);
      vt[b] = 1;
      ne++;
      mincost += min;
      cost[a][b] = cost[b][a] = 999;
   }
 printf("Minimum spanning tree cost is %d\n", mincost);
}
int main() {
   printf("Enter the number of vertices: ");
   scanf("%d", &n);
   printf("Enter the cost matrix\n");
   for (int i = 0; i < n; i++)
      for (int j = 0; j < n; j++)
         scanf("%d", &cost[i][j]);
   prim();
   return 0;
}
```

3a Floyd's algorithm

```
#include <stdio.h>
int min(int a, int b) {
   return (a < b ? a : b);
}
void floyd(int D[][10], int n) {
   for (int k = 1; k <= n; k++)
      for (int i = 1; i <= n; i++)
         for (int j = 1; j <= n; j++)
            D[i][j] = min(D[i][j], D[i][k] + D[k][j]);
}
int main() {
   int n, cost[10][10];
   printf("Enter the number of vertices: ");
   scanf("%d", &n);
   printf("Enter the cost matrix\n");
   for (int i = 1; i <= n; i++)
      for (int j = 1; j <= n; j++)
         scanf("%d", &cost[i][j]);
   floyd(cost, n);
   printf("All pair shortest path\n");
   for (int i = 1; i <= n; i++) {
      for (int j = 1; j <= n; j++)
         printf("%d  ", cost[i][j]);
      printf("\n");
   }
   return 0;
}
```

3b.

```
 #include<stdio.h>
void warshal(int A[][10],int n)
{
   for(int k=1;k<=n;k++)
      for(int i=1;i<=n;i++)
         for(int j=1;j<=n;j++)
            A[i][j]=A[i][j] || (A[i][k] && A[k][j]);
}
void main()
{
   int n, adj[10][10];
   printf("Enter no. of Vertices: ");
   scanf("%d",&n);
   printf("Enter the adjacency matrix\n");
   for(int i=1;i<=n;i++)
      for(int j=1;j<=n;j++)
         scanf("%d",&adj[i][j]);
   warshal(adj,n);
   printf("Transitive closure of the given graph is\n");
   for(int i=1;i<=n;i++)
   {
 for(int j=1;j<=n;j++)
         printf("%d  ",adj[i][j]);
      printf("\n");
   }
}
```

4. vertices using Dijkstra's algorithm

```
#include <stdio.h>
int cost[10][10], n, dist[10];
int minm(int m, int n) {
   return (m < n ? m : n);
}
void dijkstra(int source) {
   int s[10] = {0};
   int min, w = 0;
   for (int i = 0; i < n; i++)
      dist[i] = cost[source][i];
   dist[source] = 0;
   s[source] = 1;
   for (int i = 0; i < n - 1; i++) {
      min = 999;
      for (int j = 0; j < n; j++) {
         if (s[j] == 0 && min > dist[j]) {
            min = dist[j];
            w = j;
         }
      }
      s[w] = 1;
      for (int v = 0; v < n; v++) {
         if (s[v] == 0 && cost[w][v] != 999) {
            dist[v] = minm(dist[v], dist[w] +
cost[w][v]);
         }
      }
   }
}
int main() {
   int source;
  printf("Enter the number of vertices: ");
   scanf("%d", &n);
  printf("Enter the cost matrix\n");
   for (int i = 0; i < n; i++)
      for (int j = 0; j < n; j++)
         scanf("%d", &cost[i][j]);
 printf("Enter the source vertex: ");
   scanf("%d", &source);
   dijkstra(source);
 printf("The shortest distance is...\n");
   for (int i = 0; i < n; i++)
printf("Cost from %d to %d is %d\n", source, i,
dist[i]);

   return 0;
}
```

5. Topological sort

```
#include <stdio.h>
int cost[10][10], n, colsum[10];
void cal_colsum() {
   for (int j = 0; j < n; j++) {
      colsum[j] = 0;
      for (int i = 0; i < n; i++) {
         colsum[j] += cost[i][j];
      }
   }
}
void source_removal() {
   int i, j, k, select[10] = {0};
   printf("Topological ordering is:");
   for (i = 0; i < n; i++) {
      cal_colsum();
      for (j = 0; j < n; j++) {
     if (select[j] == 0 && colsum[j] == 0) {
            break;
         }
      }
      printf("%d ", j);
      select[j] = 1;
      for (k = 0; k < n; k++) {
         cost[j][k] = 0;
      }
   }
}
int main() {
   printf("Enter no. of Vertices: ");
   scanf("%d", &n);
   printf("Enter the cost matrix\n");
   for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
         scanf("%d", &cost[i][j]);
      }
   }
   source_removal();
   return 0;
}
```

6. Knapsack problem using Dynamic Programming method.

```
#include <stdio.h>
int n, m, p[10], w[10];
int max(int a, int b) {
   return (a > b ? a : b);
}
void knapsack_DP() {
   int V[10][10], i, j;
   for (i = 0; i <= n; i++) {
      for (j = 0; j <= m; j++) {
         if (i == 0 || j == 0) {
            V[i][j] = 0;
         } else if (j < w[i]) {
            V[i][j] = V[i - 1][j];
         } else {
       V[i][j] = max(V[i - 1][j], p[i] + V[i - 1][j - w[i]]);
         }
      }
   }
   for (i = 0; i <= n; i++) {
      for (j = 0; j <= m; j++) {
         printf("%d  ", V[i][j]);
      }
      printf("\n");
   }
   printf("Items included are:");
   while (n > 0) {
      if (V[n][m] != V[n - 1][m]) {
         printf("%d  ", n);
         m = m - w[n];
      }
      n--;
   }
}
int main() {
   int i;
   printf("Enter the no. of items: ");
   scanf("%d", &n);
   printf("Enter the weights of n items: ");
   for (i = 1; i <= n; i++) {
      scanf("%d", &w[i]);
   }
   printf("Enter the prices of n items: ");
   for (i = 1; i <= n; i++) {
      scanf("%d", &p[i]);
   }
   printf("Enter the capacity of Knapsack: ");
   scanf("%d", &m);
   knapsack_DP();
   return 0;}
```

7.Discrete and continuous knapsack problem greedy Approximation

```
#include <stdio.h>
int n, m, p[10], w[10];
void greedy_knapsack() {
   float max, profit = 0;
   int k = 0, i, j;
   printf("Item included is: ");
   for (i = 0; i < n; i++) {
      max = 0;
      for (j = 0; j < n; j++) {
         if (((float)p[j]) / w[j] > max) {
            k = j;
            max = ((float)p[j]) / w[j];
         }
      }
      if (w[k] <= m) {
         printf("%d ", k);
         m = m - w[k];
         profit = profit + p[k];
         p[k] = 0;
      } else {
         break;
      }
   }
   printf("\nDiscrete Knapsack profit = %f\n",
profit);
   printf("Continuous Knapsack also includes item
%d with portion: %f\n", k, (float)m / w[k]);
   profit = profit + ((float)m / w[k]) * p[k];
   printf("Continuous Knapsack profit = %f\n",
profit);
}
int main() {
   int i;
   printf("Enter the no. of items: ");
   scanf("%d", &n);
   printf("Enter the weights of n items: ");
   for (i = 0; i < n; i++) {
      scanf("%d", &w[i]);
   }
   printf("Enter the prices of n items: ");
   for (i = 0; i < n; i++) {
      scanf("%d", &p[i]);
   }
   printf("Enter the capacity of Knapsack: ");
   scanf("%d", &m);
   greedy_knapsack();
   return 0;
}
```

8. subset of a given set S = {sl , s2,. ... ,sn} of n positive integers

```
#include <stdio.h>
int x[10], w[10], count, d;
void sum_of_subsets(int s, int k, int rem) {
   x[k] = 1;
   if (s + w[k] == d) {
      printf("subset = %d\n", ++count);
      for (int i = 0; i <= k; i++) {
         if (x[i] == 1) {
            printf("%d  ", w[i]);
         }
      }
      printf("\n");
   } else if (s + w[k] + w[k + 1] <= d) {
      sum_of_subsets(s + w[k], k + 1, rem - w[k]);
   }
   if ((s + rem - w[k] >= d) && (s + w[k + 1] <= d)) {
      x[k] = 0;
      sum_of_subsets(s, k + 1, rem - w[k]);
   }
}
int main() {
   int sum = 0, n;
  printf("Enter no of elements: ");
   scanf("%d", &n);
printf("Enter the elements in increasing order: ");
   for (int i = 0; i < n; i++) {
      scanf("%d", &w[i]);
      sum = sum + w[i];
   }
   printf("Enter the sum: ");
   scanf("%d", &d);
   if ((sum < d) || (w[0] > d)) {
      printf("No subset possible\n");
   } else {
      sum_of_subsets(0, 0, sum);
   }
   return 0;
}
```

9. Random no generator for n number selection sort

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
void selectionSort(int arr[], int n) {
   int i, j, minIndex, temp;
   for (i = 0; i < n - 1; i++) {
      minIndex = i;
      for (j = i + 1; j < n; j++) {
         if (arr[j] < arr[minIndex]) {
            minIndex = j;
         }
      }
      temp = arr[minIndex];
      arr[minIndex] = arr[i];
      arr[i] = temp;
   }
}
int main() {
   int n, i;
   clock_t start, end;
   double cpu_time_used;
  printf("Enter the number of elements (n): ");
  scanf("%d", &n);
   if (n < 5000) {
printf("Please enter a value of n greater than
5000.\n");
      return 1;
   }
   int *arr = (int *)malloc(n * sizeof(int));
   if (arr == NULL) {
      printf("Memory allocation failed.\n");
      return 1;
   }
   srand(time(NULL));
   for (i = 0; i < n; i++) {
      arr[i] = rand() % 10000;
   }
   start = clock();
   selectionSort(arr, n);
   end = clock();
 cpu_time_used = ((double)(end - start)) /
CLOCKS_PER_SEC;
printf("Time taken for sorting: %lf seconds\n",
cpu_time_used);
   free(arr);
   return 0;
}
```

10. Random no generator for n number quick sort

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int hoarePartition(int arr[], int l, int r) {
   int pivot = arr[l];
   int i = l - 1, j = r + 1;
   while (1) {
      do {
         i++;
      } while (arr[i] < pivot);
      do {
         j--;
      } while (arr[j] > pivot);
      if (i >= j)
       return j;
      int temp = arr[i];
      arr[i] = arr[j];
      arr[j] = temp;
   }
}
void quickSort(int arr[], int low, int high) {
   if (low < high) {
      int pi = hoarePartition(arr, low, high);
      quickSort(arr, low, pi);
      quickSort(arr, pi + 1, high);
   }
}
int main() {
   int n, i;
   clock_t start, end;
   double cpu_time_used;
   printf("Enter the number of elements (n): ");
   scanf("%d", &n);
   if (n < 5000) {
 printf("Please enter a value of n greater than
5000.\n");
      return 1;
   }
   int *arr = (int *)malloc(n * sizeof(int));
   if (arr == NULL) {
      printf("Memory allocation failed.\n");
      return 1;
   }
   srand(time(NULL));
   for (i = 0; i < n; i++) {
      arr[i] = rand() % 10000;
   }
   start = clock();
   quickSort(arr, 0, n - 1);
   end = clock();
   cpu_time_used = ((double)(end - start)) /
CLOCKS_PER_SEC;
   printf("Time taken for sorting: %lf seconds\n",
cpu_time_used);
   free(arr);
   return 0; }
```

11. Random no generator for n number merge Sort

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
void merge(int arr[], int l, int m, int r) {
   int i, j, k;
   int n1 = m - l + 1;
   int n2 = r - m;
   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 r) {
   if (l < r) {
      int m = l + (r - l) / 2;
      mergeSort(arr, l, m);
      mergeSort(arr, m + 1, r);
      merge(arr, l, m, r);
   }
}
```

12. N Queen's problem using Backtracking

```
#include <stdio.h>
#include <math.h>
int place(int x[], int k) {
   for (int i = 1; i < k; i++) {
      if ((x[i] == x[k]) || (abs(x[i] - x[k]) == abs(i - k)))
         return 0;
   }
   return 1;
}
int nqueens(int n) {
   int x[10], k, count = 0;
   k = 1;
   x[k] = 0;
   while (k != 0) {
      x[k]++;
      while ((x[k] <= n) && (!place(x, k)))
         x[k]++;
      if (x[k] <= n) {
         if (k == n) {
            printf("\nSolution %d\n", ++count);
            for (int i = 1; i <= n; i++) {
               for (int j = 1; j <= n; j++)
                  printf("%c ", j == x[i] ? 'Q' : 'X');
               printf("\n");
            }
         } else {
            ++k;
            x[k] = 0;
         }
      } else {
         k--;
      }
   }
   return count;
}
int main() {
   int n;
 printf("Enter the size of chessboard: ");
   scanf("%d", &n);
printf("\nThe number of possibilities are %d", nqueens(n));
   return 0;
}
```