1) MERGE SORT:

```
#include<stdio.h>
#include<stdlib.h>
#define size 1000
int count;

void merge(int A[size], int l, int m, int r){
   int i=l, j=m+1, k=l, B[size];
   while(i<=m && j<=r){
      if(A[i] < A[j]){
         B[k++] = A[i++];
         count++;
      }else{
         B[k++] = A[j++];
         count++;
      }
   }
   while(i<=m){
      B[k++] = A[i++];
   }
   while(j<=r){
      B[k++] = A[j++];
   }
   for(i=l; i<=r; i++){
      A[i] = B[i];
   }
}

void mergesort(int A[size], int l, int r){
   if(l>=r){
      return;
   }
   int mid = l + (r-l)/2;
   mergesort(A,l,mid);
```

```
    mergesort(A,mid+1,r);
    merge(A,l,mid,r);
}

int main(){
    int A[size], X[size], Y[size], Z[size];
    int i, j, n, c1, c2, c3;
    printf("Enter the length: ");
    scanf("%d", &n);
    printf("Enter the elements: ");
    for(i=0; i<n; i++){
        scanf("%d", &A[i]);
    }
    mergesort(A,0,n-1);
    printf("The sorted array is: ");
    for(i=0; i<n; i++){
        printf("%d ", A[i]);
    }
    printf("The no. of basic operation is: %d\n", count);
    printf("\nSIZE\tASC\tDSC\tRANDOM\n");
    for(i=16; i < size; i=i*2){
        for(j=0; j<i; j++){
            X[j] = j;
            Y[j] = i-j-1;
            Z[j] = rand()%i;
        }
        count = 0;
        mergesort(X,0,i-1);
        c1 = count;
        count = 0;
        mergesort(Y,0,i-1);
        c2 = count;
        count = 0;
        mergesort(Z,0,i-1);
```

```
        c3 = count;
        printf("%d\t%d\t%d\t%d\t\n", i, c1, c2, c3);
    }
    return 0;
}
```

2) PRESORT:

```
#include<stdio.h>
#include<stdlib.h>
#define size 100

void merge(int A[size],int l, int m, int r ){
    int B[size];
    int i = l, j = m+1, k = l;
    while( i <= m &&  j <= r ){
        if(A[i] > A[j])
            B[k++] = A[j++];
        else
            B[k++] = A[i++];
    }
    while(i <= m)
        B[k++] = A[i++];
    while(j <= r)
        B[k++] = A[j++];
    for(i = l; i <= r; i++)
        A[i] = B[i];
    return;
}
void mergesort(int arr[size], int l, int r){
    if(l >= r)
        return;
    int m = l + (r-l)/2;
    mergesort(arr, l , m);
    mergesort(arr, m+1, r);
    merge(arr, l , m, r);
```

```
}

int main(){
   int arr[size],n;
   printf("Enter the number of elements\n");
   scanf("%d",&n);
   printf("Enter the elements\n");
   for(int i = 0; i < n; i++)
      scanf("%d",&arr[i]);
   mergesort(arr, 0, n-1);
   printf("\n");
   for(int i = 0; i < n ; i++)
      printf("%d ",arr[i]);
   for(int i = 1; i < n; i++)
      if(arr[i-1] == arr[i])
      {
         printf("\nThe elements are not unique\n");
         exit(0);
      }
   printf("\nThe elements are unique\n");
   return 0;
}
```

3) TOPOLOGICAL ORDER:

```
#include<stdio.h>
#include<stdlib.h>
int stack[10];
int output[10];
int top = -1;
int wow = 0;

void dfs(int a[10][10], int n, int visited[10], int current){
   int j, m;
   stack[++top] = current;
   visited[current] = 1;
```

```
    for(j=0; j<n; j++){
        if(a[current][j] == 1 && visited[j] == 0){
            dfs(a,n,visited,j);
        }
    }
    m = stack[top--];
    printf("%d ", m);
    output[wow++] = current;
}

void DFS(int a[10][10], int n){
    int visited[10], comp = 0, i;
    for(i=0; i<n; i++){
        visited[i] = 0;
    }
    printf("Pop order: ");
    for(i=0; i<n; i++){
        if(visited[i] == 0){
            dfs(a,n,visited,i);
            comp++;
        }
    }
    if(comp>1){
        printf("The graph is disconnected.\n");
        printf("The no. of components: %d.\n", comp);
    }else{
        printf("The graph is connected.\n");
    }
}

int main(){
    int a[10][10], n;
    printf("Enter the no. of nodes: ");
    scanf("%d", &n);
```

```
    printf("Enter the adjacency matrix: \n");
    for(int i=0; i<n; i++){
        for(int j=0; j<n; j++){
            scanf("%d", &a[i][j]);
        }
    }
    DFS(a,n);
    printf("Topological order: ");
    while(wow > 0){
        printf("%d ", output[--wow]);
    }
    return 0;
}
```

4) KNAPSACK PROBLEM:

```
#include<stdio.h>

int max(int a, int b){
    return(a > b? a : b);
}

void knapsack(int n, int m, int w[n+1], int v[n+1][m+1], int p[n+1]){
    int 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]]);
            }
        }
    }
}
```

```
void ItemOfOptimal(int n, int m, int w[n+1], int v[+1][m+1]){
   int i, j;
   if(v[i][j] == 0){
      printf("Not possible\n");
      return;
   }
   printf("Optimal Solution: %d", v[n][m]);
   i=n;
   j=m;
   printf("Objects selected: ");
   while( i!= 0 && j!= 0){
      if(v[i][j] != v[i-1][j]){
         printf("\n%d ", i);
         j = j - w[i];
      }
      i = i -1;
   }
   printf("\n");
}

int main()
{
   int m, n, i, j, p[20], w[20], v[20][20];
   printf("Enter no. of objects:");
   scanf("%d", &n);
   printf("Enter weight of %d objects:", n);
   for (i = 1; i <= n; i++)
      scanf("%d", &w[i]);
   printf("Enter Profits/values:");
   for (i = 1; i <= n; i++)
      scanf("%d", &p[i]);
   printf("Enter capacity:");
   scanf("%d", &m);
```

```
    knapsack(n, m, w, v, p);
    ItemOfOptimal(n, m, w, v);
}
```

5) SUM OF SUBNET:

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

int w[10], x[10], d;

void subnet(int s, int k, int r){
    int i;
    static int b = 1;
    x[k] = 1;
    if(w[k] + s == d){
        printf("\nSubnet %d): ", b++);
        for(i=1; i<=k; i++){
            if(x[i] == 1){
                printf("%d\t", w[i]);
            }
        }
    }else if(s + w[k] + w[k+1] <= d){
        subnet(s+w[k], k+1, r-w[k]);
    }
    if((s+r-w[k] >= d) && (s+w[k+1] <=d)){
        x[k] = 0;
        subnet(s,k+1,r-w[k]);
    }
}

int main(){
    int n, i, sum = 0;
    printf("SUBNET PROBLEM:\n");
    printf("Enter the no. of elements: ");
```

```
    scanf("%d", &n);
    printf("Enter the elemenst: ");
    for(i=1; i<=n; i++){
        scanf("%d", &w[i]);
        sum+=w[i];
    }
    printf("Enter the subnet max value required: ");
    scanf("%d", &d);
    if(sum < d || w[1] > d){
        printf("No subnet possible\n");
        exit(0);
    }
    subnet(0,1,sum);
    return 0;
}
```

6) N queen:

```
#include <stdio.h>
#include <stdlib.h>
int x[20];
int place(int k, int i)
{
    int j;
    for (j = 1; j <= k - 1; j++)
    {
        if ((x[j] == i) || (abs(x[j] - i) == abs(j - k)))
            return 0;
    }
    return 1;
}
void printQueen(int n)
{
    int i, j;
    static int count = 1;
    printf("\n\nSolution %d is- \n\n", count++);
```

```
   for (i = 1; i <= n; ++i)
      printf("\t%d", i);
   for (i = 1; i <= n; ++i)
   {
      printf("\n\n%d", i);
      for (j = 1; j <= n; ++j)
      {
         if (x[i] == j)
            printf("\tQ");
         else
            printf("\t-");
      }
   }
}
void NQueen(int k, int n)
{
   int i;
   for (i = 1; i <= n; i++)
   {
      if (place(k, i))
      {
         x[k] = i;
         if (k == n)
         {
            printQueen(n);
         }
         else
            NQueen(k + 1, n);
      }
   }
}
int main()
{
   int n;
```

```
    printf("\nN-QUEEN PROBLEM\n");
    printf("\nEnter the number of queens to be placed - ");
    scanf("%d", &n);
    if (n == 2 || n == 3 || n == 0)
        printf("\nNo solutions possible!!");
    else
        NQueen(1, n);
    return (0);
}
```

7) Dijkstras:

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

int minDistance(int V,int dist[], bool sptSet[])
{
    int min = INT_MAX, min_index;
    for (int v = 0; v < V; v++)
        if (sptSet[v] == false && dist[v] <= min)
            min = dist[v], min_index = v;
    return min_index;
}

void printPredecessor(int val,int pred[])
{
    if(pred[val] == -1){
        printf("%d",val);
        return;
    }
    printPredecessor(pred[val],pred);
    printf("=>%d",val);
}

void printSolution(int V,int dist[],int pred[])
```

```
{
   printf("Vertex \t\t Distance from Source\t\tTracking\n");
   for (int i = 0; i < V; i++){
      printf("%d \t\t\t\t %d\t\t", i, dist[i]);
      printPredecessor(i,pred);
      printf("\n");
   }
}

void dijkstra(int V, int graph[V][V], int src)
{
   int dist[V];
   int pred[V];
   bool sptSet[V];
   for (int i = 0; i < V; i++)
      dist[i] = INT_MAX, sptSet[i] = false;
   dist[src] = 0;
   pred[src] = -1;
   for (int count = 0; count < V - 1; count++) {
      int u = minDistance(V,dist, sptSet);
      sptSet[u] = true;
      for (int v = 0; v < V; v++)
         if (!sptSet[v] && graph[u][v]
            && dist[u] != INT_MAX
            && (dist[u] + graph[u][v])< dist[v]){
            dist[v] = dist[u] + graph[u][v];
            pred[v] = u;
         }
   }
   printSolution(V,dist,pred);
}

int main()
{
```

```
    int V;
    printf("Enter number of vertices:");
    scanf("%d",&V);
    int graph[V][V];
    printf("Enter the values of the graph:\n");
    for(int i = 0; i < V; i++)
        for(int j=0; j < V; j++)
            scanf("%d",&graph[i][j]);
    printf("\n");
    dijkstra(V,graph, 0);
    return 0;
}
```

8) Horspool:

```
#include <stdio.h>
#include <string.h>
#define MAX 256
int t[MAX];
int count = 1;
void shifttable(char pat[])
{
    int i, j, m;
    m = strlen(pat);
    for (i = 0; i < MAX; i++)
        t[i] = m;
    for (j = 0; j < m - 1; j++)
        t[pat[j]] = m - 1 - j;
}
int horspool(char src[], char pat[])
{
    int i, j, k, m, n;
    n = strlen(src);
    m = strlen(pat);
    i = m - 1;
    while (i < n)
```

```
    {
        k = 0;
        while ((k < m) && (pat[m - 1 - k] == src[i - k]))
            k++;
        if (k == m)
            return (i - m + 1);
        else
        {
            i = i + t[src[i]];
            count = count + 1;
        }
    }
    return -1;
}
int main()
{
    char src[100], pat[10];
    int pos;
    printf("\nEnter the main source string\n");
    scanf(" %[^\n]s",src);
    printf("\nEnter the pattern to be searched\n");
    scanf(" %[^\n]s",pat);
    shifttable(pat);
    pos = horspool(src, pat);

    if (pos >= 0)
    {
        printf("\nFound at % d position", pos + 1);
        printf("\nNumber of shifts are %d\n", count);
    }
    else
        printf("\nString match failed\n");
    return 0;
}
```