1)Merge Sort

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAX 1000
int count;
void merge(int a[MAX], int low, int mid, int high)
{
   int i, j, k, b[MAX];
   i = low;
   j = mid + 1;
   k = low;
   while (i <= mid && j <= high)
   {
      if (a[i] < a[j])
      {
         b[k++] = a[i++];
      }
      else
      {
         b[k++] = a[j++];
      }
      count++;
   }
   while (i <= mid)
   {
      b[k++] = a[i++];
      count++;
   }
   while (j <= high)
   {
      b[k++] = a[j++];
      count++;
   }
   for (i = low; i <= high; i++)
   {
      a[i] = b[i];
```

```
    }
}
void mergesort(int a[MAX], int low, int high)
{
    int mid;
    if (low < high)
    {
        mid = (low + high) / 2;
        mergesort(a, low, mid);
        mergesort(a, mid + 1, high);
        merge(a, low, mid, high);
    }
}
int main()
{
    int i, j, n, a[MAX], b[MAX], c[MAX];
    int c1, c2, c3;
    printf("\nEnter n: ");
    scanf("%d", &n);
    printf("\nEnter elements: ");
    for (i = 0; i < n; i++)
    {
        scanf("%d", &a[i]);
    }
    count = 0;
    mergesort(a, 0, n - 1);
    printf("\nSorted elements: \n");
    for (i = 0; i < n; i++)
    {
        printf("%d\n", a[i]);
    }
    printf("\nNumber of counts: %d\n", count);
    printf("\nSIZE\tASC\tDESC\tRAND\n");
    srand(time(NULL));
    for (i = 16; i < 550; i = i * 2)
    {
        for (j = 0; j < i; j++)
```

```
        {
            a[j] = j;        // Ascending order
            b[j] = i - j;    // Descending order
            c[j] = rand() % i; // Random order
        }
        count = 0;
        mergesort(a, 0, i - 1);
        c1 = count;
        count = 0;
        mergesort(b, 0, i - 1);
        c2 = count;
        count = 0;
        mergesort(c, 0, i - 1);
        c3 = count;
        printf("\n%d\t%d\t%d\t%d", i, c1, c2, c3);
    }
    return 0;
}
```

2)Topological Order

```
#include <stdio.h>
#include <stdlib.h>
int j=0,pop[10],v[10];
void dfs(int source,int n,int a[10][10])
{
   int i,k,top=-1,stack[10];
   v[source]=1;
   stack[++top]=source+1;
   while(top!=-1)
   {
      for(k=0;k<n;k++)
      {
         if( a[source][k]==1 && v[k]==1)
         {
            for(i=top;i>=0;i--)
            if(stack[i] == k+1)
            {
               printf("\n Topological order not possible");
               exit(0);
            }
         }
         else
         {
            if( a[source][k] == 1 && v[k] == 0)
            {
               v[k]=1;
               stack[++top]=k+1;
               source=k;
               k=0;
            }
         }
      }
      pop[j++]=source+1;
      top--;
      source=stack[top]-1;
   }
```

```
}
void topo(int n, int a[10][10])
{
int i,k;
for(i=0;i<n;i++)
v[i]=0;
for(k=0;k<n;k++)
if(v[k]==0)
dfs(k,n,a);
}
int main()
{
   int n,i,j,a[10][10];
   printf("\n Enter the no. of vertices:");
   scanf("%d", &n);
   printf("\n Enter the adjacency matrix\n");
   for(i=0;i<n;i++)
   for(j=0;j<n;j++)
   scanf("%d",&a[i][j]);
   topo(n,a);
   printf("\n The topological ordering is\n");
   for(i=n-1;i>=0;i--)
   printf("%d\t",pop[i]);
}
```

3)Presort

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAX 1000
int count;
void merge(int a[MAX], int low, int mid, int high)
{
   int i, j, k, b[MAX];
   i = low;
   j = mid+1;
   k = low;
   while (i <= mid && j <= high)
   {
      if (a[i] < a[j])
      {
         b[k++] = a[i++];
      }
      else
      {
         b[k++] = a[j++];
      }
      count++;
   }
   while (i <= mid)
   {
      b[k++] = a[i++];
      count++;
   }
   while (j <= high)
   {
      b[k++] = a[j++];
      count++;
   }
   for (i = low; i <= high; i++)
   {
      a[i] = b[i];
```

```
    }
}
void mergesort(int a[MAX], int low, int high)
{
    int mid;
    if (low < high)
    {
        mid = (low + high) / 2;
        mergesort(a, low, mid);
        mergesort(a, mid + 1, high);
        merge(a, low, mid, high);
    }
}
int presort(int n, int a[MAX])
{
    mergesort(a, 0, n - 1);
    int i;
    for (i = 0; i <= n - 2; i++)
    {
        if (a[i] == a[i + 1])
            return 1;
    }
    return 0;
}
int main()
{
    int i, n, a[MAX];
    printf("\n Enter n: ");
    scanf("%d", &n);
    printf("\nEnter elements: ");
    for (i = 0; i < n; i++)
    {
        scanf("%d", &a[i]);
    }
    int x;
    x = presort(n, a);
    if(x==0)
```

```
    {
        printf("Unique!\n");
    }
    if(x==1)
    {
        printf("Not Unique!\n");
    }
    count = 0;
    clock_t start, end;
    double cpu;
    start = clock();
    mergesort(a,0,n-1);
    end = clock();
    cpu = ((double) (end - start)) / CLOCKS_PER_SEC;
    printf("Time taken: %f sec", cpu);
    printf("\nSorted elements: \n");
    for (i = 0; i < n; i++)
    {
        printf("%d\n", a[i]);
    }
    printf("\nNumber of counts: %d\n", count);
}
```

4)Horspool Algorithm

```
#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("%s",src);
   printf("\nEnter the pattern to be searched\n");
   scanf("%s",pat);
   shifttable(pat);
   pos=horspool(src,pat);
   if(pos>=0)
   {
      printf("\n Found at %d position ",pos+1);
      printf("\n Number of shifts are %d",count);
   }
   else
   printf("\n String match failed");
   return 0;
}
```

5)Knapsack Problem

```
#include <stdio.h>
#define MAX 150
int knap(int n,int m);
int big(int a,int b);
int w[MAX];
int p[MAX];
int v[MAX][MAX];
int big(int a,int b)
{
   if (a > b) return a;
   else return b;
}
int knap(int n,int m)
{
   int i,j;
   for(i=1;i<=n;i++)
   for(j=1;j<=m;j++)
   {
      if((j-w[i])<0)
      v[i][j]=v[i-1][j];
      else
      v[i][j]=big(v[i-1][j],p[i]+v[i-1][j-w[i]]);
   }
   return v[n][m];
}
int main()
{
   int i,j,profit,n,m;
   printf("\nEnter n (no. of items): ");
   scanf("%d",&n);
   printf("\nEnter the knapsack capacity:");
   scanf("%d",&m);
   printf("\nEnter the weights and profits:\n");
   for(i=1;i<=n;i++)
   {
      printf("w[%d] = ",i);
```

```
        scanf("%d",&w[i]);
        printf("p[%d] = ",i);
        scanf("%d",&p[i]);
    }
    for(i=0;i<=n;i++)
    v[i][0]=0;
    for(j=0;j<=m;j++)
    v[0][j]=0;
    profit=knap(n,m);
    printf("\nGoal=%d\n\n",profit);
    return 0;
}
```

6)Dijkstra's Algorithm

```
#include <stdio.h>
#define INFINITY 999
void dijk(int cost[10][10],int n,int source,int v[10],int d[10]);
int main()
{
   int n;
   int cost[10][10];
   int source;
   int v[10];
   int d[10];
   int i,j;
   printf("Enter n: ");
   scanf("%d",&n);
   printf("Enter Cost matrix: \n");
   for(i=1;i<=n;i++)
   for(j=1;j<=n;j++)
   scanf("%d",&cost[i][j]);
   printf("\nEnter source:");
   scanf("%d",&source);
   for(i=1;i<=n;i++)
   {
      d[i]=cost[source][i];
      v[i]=0;
   }
   dijk(cost,n,source,v,d);
   printf("Shortest distance from source %d\n\n",source);
   for(i=1;i<=n;i++)
   printf("%d-->%d=%d\n\n",source,i,d[i]);
   return 0;
}
void dijk(int cost[10][10],int n,int source,int v[10],int d[10])
{
   int least,i,j,u;
   v[source] =1;
   for(i=1;i<=n;i++)
   {
```

```
        least = INFINITY;
        for(j=1;j<=n;j++)
        {
           if(v[j]==0&&d[j]<least)
           {
              least = d[j];
              u=j;
           }
        }
        v[u]=1;
        for(j=1;j<=n;j++)
        {
           if(v[j]==0 && (d[j] > (d[u]+ cost[u][j])))
           d[j]=d[u]+cost[u][j];
        }
     }
}
```

7)Sum of Subsets

```
#include <stdio.h>
#define MAX 10
int s[MAX],x[MAX];
int d;
void sumofsub(int p,int k,int r)
{
   int i;
   x[k]=1;
   if ((p+s[k])==d)
   {
      for(i=1;i<=k;i++)
      if (x[i]==1)
      printf("%d ",s[i]);
      printf("\n");
   }
   else
   if (p+s[k]+s[k+1]<=d)
   sumofsub(p+s[k],k+1,r-s[k]);
   if((p+r-s[k]>=d) && (p+s[k+1]<=d))
   {
      x[k]=0;
      sumofsub(p,k+1,r-s[k]);
   }
}
void main()
{
int i,n,sum=0;
printf("\nEnter max. number:");
scanf("%d",&n);
printf("\nEnter the set in increasing order:\n");
for(i=1;i<=n;i++)
scanf("%d",&s[i]);
printf("\n Enter the max. subset value: ");
scanf("%d",&d);
for(i=1;i<=n;i++)
sum=sum+s[i];
```

```
if(sum<d||s[1]>d)
printf("\n No subset possible");
else
sumofsub(0,1,sum);
}
```

8)N-queens problem

```
#include <stdio.h>
#include <stdlib.h>
void nqueens(int n);
int can_place(int c[10],int r);
void display(int c[10],int r);
int count = 0;
int main()
{
 int n;
 printf("Enter n(no. of queens):");
 scanf("%d",&n);
 if(n==2||n==3)
 printf("Solution does not exist.");
 else
 {
    nqueens(n);
    printf("Total no. of solutions: %d\n",count);
 }
 return 0;
}
void nqueens(int n)
{
   int r;
   int c[10];
   int i;
   r=0;
   c[r]=-1;
   while(r>=0)
   {
      c[r]++;
      while(c[r]<n && !can_place(c,r))
      c[r]++;
      if(c[r]<n)
      {
         if(r==n-1)
         {
```

```
            printf("Solution %d:",++count);
            for(i=0;i<n;i++)
            printf("%4d",c[i]+1);
            display(c,n);
          }
          else
          {
            r++;
            c[r]=-1;
          }
        }
        else
        r--;
    }
}
int can_place(int c[10],int r)
{
    int i;
    for(i=0;i<r;i++)
    {
        if((c[i]==c[r])||(abs(i-r)==abs(c[i]-c[r])))
        return 0;
    }
    return 1;
}
void display(int c[10],int n)
{
    char cb[10][10];
    int i,j;
    for(i=0;i<n;i++)
    for(j=0;j<n;j++)
    cb[i][j]='-';
    for(i=0;i<n;i++)
    cb[i][c[i]]='Q';
    printf("\n\nChessboard: \n");
    for(i=0;i<n;i++)
    {
```

```
        for(j=0;j<n;j++)
        printf("%4c",cb[i][j]);
        printf("\n\n");
    }
}
```