1)selection sort

```
#include <stdio.h>
void main() {
   int i, j, temp, n, a[20];
   printf("Enter no. of elements:\n");
   scanf("%d", &n);
   printf("Enter the numbers:\n");
   for(i = 0; i < n; i++)
   scanf("%d", &a[i]);
   for(i = 0; i < n; i++) {
   for(j = i + 1; j < n; j++) {
   if(a[i] > a[j]) {
            temp= a[i];
            a[i] = a[j];
            a[j] = temp;
         }
      }
   }
   printf("The numbers arranged in ascending order are:\n");
   for(i = 0; i < n; i++)
      printf("%d\t", a[i]);
}
```

10) kruskals algorithm minimum spanning tree

```
#include <stdio.h>
#include <stdlib.h>
int i, j, k, a, b, u, v, n, ne = 1;
int min, mincost = 0, cost[9][9], parent[9];
int find(int i);
```

```
int uni(int i, int j);
void main() {
   printf("\n\tImplementation of Kruskal's Algorithm\n");
   printf("\nEnter the number of vertices: ");
   scanf("%d", &n);
   printf("\nEnter the cost adjacency matrix:\n");
   for (i = 1; i <= n; i++) {
      for (j = 1; j <= n; j++) {
         scanf("%d", &cost[i][j]);
         if (cost[i][j] == 0) {
            cost[i][j] = 999;
         }
      }
   }
   printf("The edges of minimum cost spanning tree are:\n");
   while (ne < n) {
      for (i = 1, min = 999; i <= n; i++) {
         for (j = 1; j <= n; j++) {
            if (cost[i][j] < min) {
               min = cost[i][j];
               a = u = i;
               b = v = j;
            }
         }
      }
      u = find(u);
      v = find(v);
      if (uni(u, v)) {
         printf("%d edge (%d, %d) = %d\n", ne++, a, b, min);
         mincost += min;
      }
```

```
        cost[a][b] = cost[b][a] = 999;
    }
    printf("\nMinimum cost = %d\n", mincost);
    return 0;
}
int find(int i) {
    while (parent[i]) {
        i = parent[i];
    }
    return i;
}
int uni(int i, int j) {
    if (i != j) {
        parent[j] = i;
        return 1;
    }
}
```

2)TSP

```
#include <stdio.h>
int ary[10][10], completed[10], n, cost = 0;
void takeInput() {
    int i, j;
    printf("Enter the number of villages: ");
    scanf("%d", &n);

    printf("\nEnter the Cost Matrix\n");
    for (i = 0; i < n; i++) {
        printf("\nEnter Elements of Row %d:\n", i + 1);
        for (j = 0; j < n; j++) {
```

```
            scanf("%d", &ary[i][j]);
        }
        completed[i] = 0;
    }
    printf("\n\nThe cost list is:\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            printf("\t%d", ary[i][j]);
        }
        printf("\n");
    }
}
int least(int c) {
    int i, nc = 999;
    int min = 999, kmin;

    for (i = 0; i < n; i++) {
        if ((ary[c][i] != 0) && (completed[i] == 0)) {
            if (ary[c][i] + ary[i][c] < min) {
                min = ary[c][i] + ary[i][c];
                kmin = ary[c][i];
                nc = i;
            }
        }
    }
    if (min != 999) {
        cost += kmin;
    }
    return nc;
}
void mincost(int city) {
```

```
    int ncity;
    completed[city] = 1;
    printf("%d--->", city + 1);

    ncity = least(city);
    if (ncity == 999) {
        ncity = 0;
        printf("%d", ncity + 1);
        cost += ary[city][ncity];
        return;
    }
    mincost(ncity);
}
void main() {
    takeInput();
    printf("\n\nThe Path is:\n");
    mincost(0);  // starting from the first city
    printf("\n\nMinimum cost is %d\n", cost);
}
```

7) in degree out degree

```
#include <stdio.h>
#define MAX 10
void accept_graph(int G[][MAX], int n) {
    int i, j;
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            printf("Edge (V%d, V%d) exists? (yes=1, no=0): ", i, j);
            scanf("%d", &G[i][j]);
        }
```

```
    }
}
void disp_adj_mat(int G[][MAX], int n) {
    int i, j;
    printf("Adjacency Matrix:\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            printf("%4d", G[i][j]);
        }
        printf("\n");
    }
}
void calc_out_degree(int G[][MAX], int n) {
    int i, j, sum;
    printf("Out degree:\n");
    for (i = 0; i < n; i++) {
        sum = 0;
        for (j = 0; j < n; j++) {
            sum += G[i][j];
        }
        printf("out-deg(V%d) = %d\n", i, sum);
    }
}
void calc_in_degree(int G[][MAX], int n) {
    int i, j, sum;
    printf("In degree:\n");
    for (i = 0; i < n; i++) {
        sum = 0;
        for (j = 0; j < n; j++) {
            sum += G[j][i];
        }
```

```
        printf("in-deg(V%d) = %d\n", i, sum);
    }
}
void main() {
    int G[MAX][MAX], n;
    printf("Enter the number of vertices: ");
    scanf("%d", &n);
    accept_graph(G, n);
    disp_adj_mat(G, n);
    calc_out_degree(G, n);
    calc_in_degree(G, n);
}
```

3)minimum and maximum

```
#include <stdio.h>
int max, min;
int a[100];
void maxmin(int i, int j) {
    int max1, min1, mid;
    if (i == j) {
        max = min = a[i];
    } else if (i == j - 1) {
        if (a[i] < a[j]) {
            max = a[j];
            min = a[i];
        } else {
            max = a[i];
            min = a[j];
        }
    } else {
```

```
        mid = (i + j) / 2;
        maxmin(i, mid);
        max1 = max;
        min1 = min;
        maxmin(mid + 1, j);
        if (max < max1) max = max1;
        if (min > min1) min = min1;
    }
}
void main() {
    int i, num;
    printf("Enter the total number of elements: ");
    scanf("%d", &num);

    printf("Enter the numbers:\n");
    for (i = 0; i < num; i++) {
        scanf("%d", &a[i]);
    }
    max = a[0];
    min = a[0];
    maxmin(0, num - 1);
    printf("Minimum element in the array: %d\n", min);
    printf("Maximum element in the array: %d\n", max);
}
```

9) optimal binary search

```
#include <stdio.h>
#define MAX 10
int w[MAX][MAX], c[MAX][MAX], r[MAX][MAX], p[MAX], q[MAX];
```

```
void main() {
    int temp = 0, root, min, min1, n;
    int i, j, k, b;

    printf("Enter the number of elements: ");
    scanf("%d", &n);

    for (i = 1; i <= n; i++) {
        printf("Enter the frequency of element %d: ", i);
        scanf("%d", &p[i]);
    }
    printf("\n");
    for (i = 0; i <= n; i++) {
        printf("Enter the probability of %d: ", i);
        scanf("%d", &q[i]);
    }
    for (i = 0; i <= n; i++) {
        for (j = 0; j <= n; j++) {
            if (i == j) {
                w[i][j] = q[i];
                c[i][j] = 0;
                r[i][j] = 0;
            } else {
                w[i][j] = c[i][j] = r[i][j] = 0;
            }
        }
    }
    for (b = 0; b < n; b++) {
        for (i = 0, j = b + 1; j <= n; i++, j++) {
            w[i][j] = w[i][j - 1] + p[j] + q[j];
            min = 30000;
```

```
            for (k = i + 1; k <= j; k++) {
                min1 = c[i][k - 1] + c[k][j] + w[i][j];
                if (min > min1) {
                    min = min1;
                    temp = k;
                }
            }
            c[i][j] = min;
            r[i][j] = temp;

            printf("W[%d][%d] = %d\tC[%d][%d] = %d\tr[%d][%d] = %d\n", i, j, w[i][j], i, j, c[i][j], i, j, r[i][j]);
        }
        printf("\n");
    }
    printf("Minimum cost = %d\n", c[0][n]);
    root = r[0][n];
    printf("Root = %d\n", root);
}
```

4) div and conq Quick sort

```
#include <stdio.h>
void qsort(int a[], int first, int last);
int partition(int a[], int first, int last);
void qsort(int a[], int first, int last) {
    int j;
    if (first < last) {
        j = partition(a, first, last);
        qsort(a, first, j - 1);
```

```
        qsort(a, j + 1, last);
    }
}
int partition(int a[], int first, int last) {
    int v = a[first];
    int i = first;
    int j = last;
    int temp;

    do {
        do {
            i++;
        } while (i <= last && a[i] < v);

        do {
            j--;
        } while (a[j] > v);

        if (i < j) {
            temp = a[i];
            a[i] = a[j];
            a[j] = temp;
        }
    } while (i < j);

    a[first] = a[j];
    a[j] = v;

    return j;
}
int main() {
```

```
    int n, i;
    int a[100];

    printf("Enter the number of elements: ");
    scanf("%d", &n);

    printf("Enter the elements:\n");
    for (i = 0; i < n; i++) {
        scanf("%d", &a[i]);
    }
    qsort(a, 0, n - 1);

    printf("Sorted array:\n");
    for (i = 0; i < n; i++) {
        printf("%d ", a[i]);
    }
    printf("\n");

    return 0;
}
```

8) back tracking

```
#include <stdio.h>
#include <math.h>
int board[20], count = 0;
void queen(int row, int n);
void print(int n);
int place(int row, int column);
int main() {
```

```
    int n;
    printf("N-Queens Problem Using Backtracking");
    printf("\nEnter number of Queens: ");
    scanf("%d", &n);
    queen(1, n);
    return 0;
}
void print(int n) {
    int i, j;
    printf("\n\nSolution %d\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 (board[i] == j)
                printf("\tQ");
            else
                printf("\t-");
        }
    }
    printf("\n");
}
int place(int row, int column) {
    for (int i = 1; i < row; i++) {
        if (board[i] == column || abs(board[i] - column) == abs(i - row))
            return 0;
    }
    return 1;
}
```

```
void queen(int row, int n) {
   for (int column = 1; column <= n; column++) {
      if (place(row, column)) {
         board[row] = column;
         if (row == n)
            print(n);
         else
            queen(row + 1, n);
      }
   }
}
```

5) merge sort

```
#include <stdio.h>
#include <conio.h>
void merge(int[], int, int, int);
void mergesort(int[], int, int);
void merge(int a[25], int low, int mid, int high) {
   int b[25], h, i, j, k;
   h = low;
   i = low;
   j = mid + 1;
   while ((h <= mid) && (j <= high)) {
      if (a[h] < a[j]) {
         b[i] = a[h];
         h++;
      } else {
         b[i] = a[j];
         j++;
      }
```

```
        i++;
    }
    if (h > mid) {
        for (k = j; k <= high; k++) {
            b[i] = a[k];
            i++;
        }
    } else {
        for (k = h; k <= mid; k++) {
            b[i] = a[k];
            i++;
        }
    }
    for (k = low; k <= high; k++) {
        a[k] = b[k];
    }
}
void mergesort(int a[25], 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 a[25], n, i;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
```

```
    printf("Enter the elements:\n");
    for (i = 0; i < n; i++) {
        scanf("%d", &a[i]);
    }
    mergesort(a, 0, n - 1);
    printf("Sorted array:\n");
    for (i = 0; i < n; i++) {
        printf("%d ", a[i]);
    }
    getch();
    return 0;
}
```

6) vertices and edges

```
#include <stdio.h>
#define MAX_VERTICES 100
void main() {
    int adjMatrix[MAX_VERTICES][MAX_VERTICES] = {0};
    int i, j, u, v,numVertices, numEdges;
    printf("Enter the number of vertices in the graph: ");
    scanf("%d", &numVertices);
    printf("Enter the number of edges in the graph: ");
    scanf("%d", &numEdges);

    printf("Enter the edges (u, v):\n");
    for (i = 0; i < numEdges; i++) {
        scanf("%d %d", &u, &v);
        adjMatrix[u][v] = 1;
        adjMatrix[v][u] = 1;
```

```
    }
    printf("\nAdjacency Matrix:\n");
    for (i = 0; i < numVertices; i++) {
        for (j = 0; j < numVertices; j++) {
            printf("%d ", adjMatrix[i][j]);
        }
        printf("\n");
    }
}
```