SAMBHRAM INSTITUTE OF TECHNOLOGY

# DEPARTMENT OF ARTICIAL INTELLIGENCE AND MACHINE LEARNING

## ANALYSIS AND DESIGN OF ALGORITHMS LABORATORY

As per VTU - Choice Based Credit System - 22 Scheme

(Effective from the academic year of 2023 -2024)

IV Semester AIML

## LABORATORY MANUAL

**Prepared By**

Prof. Saritha Kishore

# Syllabus

| ANALYSIS AND DESIGN OF ALGORITHMS LABORATORY<br>(Effective from the academic year 2023 -2024)<br>SEMESTER – IV | | | |
|---|---|---|---|
| Course Code | BCSL404 | CIE Marks | 50 |
| Number of Contact Hours/Week (L:T:P: S) | 0:0:2:0 | SEE Marks | 50 |
| Credits | 01 | Exam Hours | 2 |
| Examination type (SEE) | Practical | | |

**Course Learning Objectives:** This course (BCSL404) will enable students to:

**Course objectives:**

- To design and implement various algorithms in C/C++ programming using suitable development tools to address different computational challenges.
- To apply diverse design strategies for effective problem-solving.

  To Measure and compare the performance of different algorithms to determine their efficiency and suitability for specific tasks.

**Programs List:**

| | |
|---|---|
| 1. | Design and implement C/C++ Program to find Minimum Cost Spanning Tree of a given connected undirected graph using Kruskal's algorithm. |
| 2. | Design and implement C/C++ Program to find Minimum Cost Spanning Tree of a given connected undirected graph using Prim's algorithm. |
| 3. | a. Design and implement C/C++ Program to solve All-Pairs Shortest Paths problem using Floyd's algorithm.<br>b. Design and implement C/C++ Program to find the transitive closure using Warshal's algorithm. |
| 4. | Design and implement C/C++ Program to find shortest paths from a given vertex in a weighted connected graph to other vertices using Dijkstra's algorithm. |
| 5. | Design and implement C/C++ Program to obtain the Topological ordering of vertices in a given digraph. |
| 6. | Design and implement C/C++ Program to solve 0/1 Knapsack problem using Dynamic Programming method. |
| 7 | Design and implement C/C++ Program to solve discrete Knapsack and continuous Knapsack problems using greedy approximation method. |
| 8. | Design and implement C/C++ Program to find a subset of a given set S = {sl , s2,.....,sn} of n positive integers whose sum is equal to a given positive integer d. |
| 9. | Design and implement C/C++ Program to sort a given set of n integer elements using Selection Sort method and compute its time complexity. Run the program for varied values of n> 5000 and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator. |

| | |
|---|---|
| 10. | Design and implement C/C++ Program to sort a given set of n integer elements using Quick Sort method and compute its time complexity. Run the program for varied values of n> 5000 and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator. |
| 11. | Design and implement C/C++ Program to sort a given set of n integer elements using Merge Sort method and compute its time complexity. Run the program for varied values of n> 5000, and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator. |
| 12. | Design and implement C/C++ Program for N Queen's problem using Backtracking. |

**Course outcomes (Course Skill Set):**

At the end of the course the student will be able to:

1. Develop programs to solve computational problems using suitable algorithm design strategy.

2. Compare algorithm design strategies by developing equivalent programs and observing running times for analysis (Empirical).

3. Make use of suitable integrated development tools to develop programs

4. Choose appropriate algorithm design techniques to develop solution to the computational and complex problems.

5. Demonstrate and present the development of program, its execution and running time(s) and record the results/inferences.

**1. Design and implement C/C++ Program to find Minimum Cost Spanning Tree of a given connected undirected graph using Kruskal's algorithm.**

```
#define INF 999
#define MAX 100
int p[MAX], c[MAX][MAX], t[MAX][2];
int find(int v)
{
   while (p[v])
      v = p[v];
   return v;
}
void union1(int i, int j)
{
   p[j] = i;
}
void kruskal(int n)
{
   int i, j, k, u, v, min, res1, res2, sum = 0;
   for (k = 1; k < n; k++)
   {
      min = INF;
      for (i = 1; i < n - 1; i++)
      {
         for (j = 1; j <= n; j++)
         {
            if (i == j) continue;
            if (c[i][j] < min)
            {
               u = find(i);
               v = find(j);
               if (u != v)
               {
                  res1 = i;
                  res2 = j;
                  min = c[i][j];
               }
            }
         }
      }
      union1(res1, find(res2));
      t[k][1] = res1;
      t[k][2] = res2;
      sum = sum + min;
   }
```

```
    printf("\nCost of spanning tree is=%d", sum);
    printf("\nEdgesof spanning tree are:\n");
    for (i = 1; i < n; i++)
        printf("%d -> %d\n", t[i][1], t[i][2]);
}

int main()
{
    int i, j, n;
    printf("\nEnter the n value:");
    scanf("%d", & n);
    for (i = 1; i <= n; i++)
        p[i] = 0;
    printf("\nEnter the graph data:\n");
    for (i = 1; i <= n; i++)
        for (j = 1; j <= n; j++)
            scanf("%d", & c[i][j]);
    kruskal(n);
    return 0;
}
```

```
Enter the n value:5

Enter the graph data:
1 3 4 6 2
1 7 6 9 3
5 2 8 99 45
1 44 66 33 6
12 4 3 2 0

Cost of spanning tree is=11
Edgesof spanning tree are:
2 -> 1
1 -> 5
3 -> 2
1 -> 4
```

2. **Design and implement C/C++ Program to find Minimum Cost Spanning Tree of a given connected undirected graph using Prim's algorithm**

```
#include<stdio.h>
#define INF 999
int prim(int c[10][10],int n,int s)
{
   int v[10],i,j,sum=0,ver[10],d[10],min,u;
   for(i=1; i<=n; i++)
   {
      ver[i]=s;
      d[i]=c[s][i];
      v[i]=0;
   }
   v[s]=1;
   for(i=1; i<=n-1; i++)
   {
      min=INF;
      for(j=1; j<=n; j++)
         if(v[j]==0 && d[j]<min)
         {
            min=d[j];
            u=j;
         }
      v[u]=1;
      sum=sum+d[u];
      printf("\n%d -> %d sum=%d",ver[u],u,sum);
      for(j=1; j<=n; j++)
         if(v[j]==0 && c[u][j]<d[j])
         {
            d[j]=c[u][j];
            ver[j]=u;
         }
   }
   return sum;
}
```

```
void main()
{
   int c[10][10],i,j,res,s,n;
   printf("\nEnter n value:");
   scanf("%d",&n);
   printf("\nEnter the graph data:\n");
   for(i=1; i<=n; i++)
      for(j=1; j<=n; j++)
         scanf("%d",&c[i][j]);
   printf("\nEnter the souce node:");
   scanf("%d",&s);
   res=prim(c,n,s);
   printf("\nCost=%d",res);
   getch();
}
```

```
Enter n value:4

Enter the graph data:
4 5 2 1
7 5 9 2
1 7 6 9
0 2 8 5

Enter the souce node:4

4 -> 1 sum=0
4 -> 2 sum=2
1 -> 3 sum=4
Cost=4
```

**3. a. Design and implement C/C++ Program to solve All-Pairs Shortest Paths problem using Floyd's algorithm.**

**b. Design and implement C/C++ Program to find the transitive closure using Warshal's algorithm.**

```
#include<stdio.h>

 intmin(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 no. 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");
      }
}
```

OUTPUT:

```
Enter no. of Vertices: 4
Enter the cost matrix
0 999 3 999
2 0 999 999
999 7 0 1
6 999 999 0
All pair shortest path
0 10 3 4
2 0 5 6
7 7 0 1
6 16 9 0
```

**3b. Design and implement C/C++ Program to find the transitive closure using Warshal's algorithm.**

**Program:**

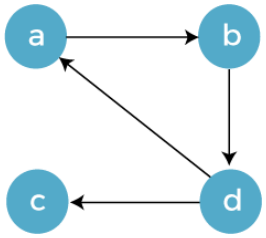
```
#include<stdio.h>

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");
        }
}
```

Sample Input and Output:




```
Enter no. of Vertices: 4 Enter
the adjacency matrix0 1 0 0
0 0 0 1
0 0 0 0
1 0 1 0
Transitive closure of the given graph is
      1  1  1  1
      1  1  1  1
      0  0  0  0
      1  1  1  1
```

**4. Design and implement C/C++ Program to find shortest paths from a given vertex in a weighted connected graph to other vertices using Dijkstra's algorithm.**

**Single Source Shortest Paths Problem:**

For a given vertex called the source in a weighted connected graph, find the shortest paths to all its other vertices. Dijkstra's algorithm is the best known algorithm for the single source shortest paths problem. This algorithm is applicable to graphs with nonnegative weights only and finds the shortest paths to a graph's vertices in order of their distance from a given source. It finds the shortest path from the source to a vertex nearest to it, then to a second nearest, and so on. It is applicable to both undirected and directed graphs.

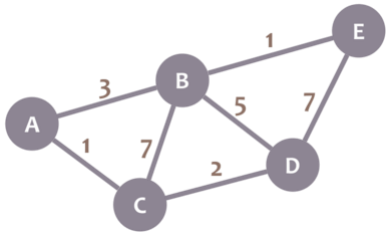
```
Algorithm : Dijkstra(G,s)
//Dijkstra's algorithm for single-source shortest paths
//Input :A weighted connected graph G=(V,E) with nonnegative weights and its vertex s
//Output : The length dv of a shortest path from s to v and its penultimate vertex pv for
//every v in V.
{
        Initialise(Q)      // Initialise vertex priority queue to empty
        for every vertex v in V do
        {
                dv←œ; pv←null
                Insert(Q,v,dv)   //Initialise vertex priority queue in the priority queue
        }
        ds←0;   Decrease(Q,s ds)          //Update priority of s with ds
        Vt←Ø
        for i←0 to |v|-1 do
        {
                u* ← DeleteMin(Q)       //delete the minimum priority element
                Vt ←Vt U {u*}
                for every vertex u in V-Vt that is adjacent to u* do
                {
                        if du* + w(u*,u)<du
                        {
                                du←du* + w(u*, u):  pu←u*
                                Decrease(Q,u,du)
                        }
                }
        }
}
```

Complexity: The Time efficiency for graphs represented by their weight matrix and the priority queue implemented as an unordered array and for graphs represented by their adjacency lists and the priority queue implemented as a min-heap, it is O(|E| log |V|).

**Program:**

```
#include<stdio.h>
#define INF 999
void dijkstra(int c[10][10],int n,int s,int d[10])
{
   int v[10],min,u,i,j;
   for(i=1; i<=n; i++)
   {
      d[i]=c[s][i];
      v[i]=0;
   }
   v[s]=1;
   for(i=1; i<=n; i++)
   {
      min=INF;
      for(j=1; j<=n; j++)
         if(v[j]==0 && d[j]<min)
         {
            min=d[j];
            u=j;
         }
      v[u]=1;
      for(j=1; j<=n; j++)
         if(v[j]==0 && (d[u]+c[u][j])<d[j])
            d[j]=d[u]+c[u][j];
   }
}
int main()
{
   int c[10][10],d[10],i,j,s,sum,n;
   printf("\nEnter n value:");
   scanf("%d",&n);
   printf("\nEnter the graph data:\n");
   for(i=1; i<=n; i++)
      for(j=1; j<=n; j++)
         scanf("%d",&c[i][j]);
   printf("\nEnter the souce node:");
   scanf("%d",&s);
   dijkstra(c,n,s,d);
   for(i=1; i<=n; i++)
      printf("\nShortest distance from %d to %d is %d",s,i,d[i]);
   return 0;
}
```

Sample Input and Output:




Enter the no.of vertices:5
Enter the cost matrix
0 3 1 999 999
3 0 7 5 1
1 7 0 2 999
999 5 2 0 7
999 1 999 7 0
Enter the source vertex:0
the shortest distance is...Cost from 0 to 0 is 0
Cost from 0 to 1 is 3
Cost from 0 to 2 is 1
Cost from 0 to 3 is 3
Cost from 0 to 4 is 4

**5. Design and implement C/C++ Program to obtain the Topological ordering of vertices ina given digraph.**

Program:

```
#include<stdio.h>
#include<conio.h>
int temp[10],k=0;
void sort(int a[][10],int id[],int n)
{
   int i,j;
   for(i=1; i<=n; i++)
   {
      if(id[i]==0)
      {
         id[i]=-1;
         temp[++k]=i;
         for(j=1; j<=n; j++)
         {
            if(a[i][j]==1 && id[j]!=-1)
               id[j]--;
         }
         i=0;
      }
   }
}
void main()
{
   int a[10][10],id[10],n,i,j;
   printf("\nEnter the n value:");
   scanf("%d",&n);
   for(i=1; i<=n; i++)
      id[i]=0;
   printf("\nEnter the graph data:\n");
   for(i=1; i<=n; i++)
      for(j=1; j<=n; j++)
      {
         scanf("%d",&a[i][j]);
         if(a[i][j]==1)
            id[j]++;
      }
```

```
sort(a,id,n);
if(k!=n)
   printf("\nTopological ordering not possible");
else
{
   printf("\nTopological ordering is:");
   for(i=1; i<=k; i++)
      printf("%d ",temp[i]);
}
getch();
```

### Sample Input and Output:

Enter no. of Vertices: 6

Enter the cost matrix

0 0 1 1 0 0

0 0 0 1 1 0

0 0 0 1 0 1

0 0 0 0 0 1

0 0 0 0 0 1

0 0 0 0 0 0

Topological ordering is:1 2 3 45 6

**6. Design and implement C/C++ Program to solve 0/1 Knapsack problem using Dynamic Programming method.**

```
#include<stdio.h>
int w[10],p[10],n;
int max(int a,int b)
{
   return a>b?a:b;
}
int knap(int i,int m)
{
   if(i==n) return w[i]>m?0:p[i];
   if(w[i]>m) return knap(i+1,m);
   return max(knap(i+1,m),knap(i+1,m-w[i])+p[i]);
}
int main()
{
   int m,i,max_profit;
   printf("\nEnter the no. of objects:");
   scanf("%d",&n);
   printf("\nEnter the knapsack capacity:");
   scanf("%d",&m);
   printf("\nEnter profit followed by weight:\n");
   for(i=1; i<=n; i++)
      scanf("%d %d",&p[i],&w[i]);
   max_profit=knap(1,m);
   printf("\nMax profit=%d",max_profit);
   return 0;
}
```

Enter the no. of objects:4

Enter the knapsack capacity:5

Enter profit followed by weight:
12 3
43 5
45 2
55 3

Max profit=100

**7. Design and implement C/C++ Program to solve discrete Knapsack and continuous Knapsack problems using greedy approximation method.**

```
#include <stdio.h>
#define MAX 50
int p[MAX], w[MAX], x[MAX];
double maxprofit;
int n, m, i;
void greedyKnapsack(int n, int w[], int p[], int m)
{
   double ratio[MAX];

// Calculate the ratio of profit to weight for each item
   for (i = 0; i < n; i++)
   {
      ratio[i] = (double)p[i] / w[i];
   }
// Sort items based on the ratio in non-increasing order
   for (i = 0; i < n - 1; i++)
   {
      for (int j = i + 1; j < n; j++)
      {
         if (ratio[i] < ratio[j])
         {
            double temp = ratio[i];
            ratio[i] = ratio[j];
            ratio[j] = temp;

            int temp2 = w[i];
            w[i] = w[j];
            w[j] = temp2;

            temp2 = p[i];
            p[i] = p[j];
            p[j] = temp2;
         }
      }
   }
   int currentWeight = 0;
   maxprofit = 0.0;
// Fill the knapsack with items
   for (i = 0; i < n; i++)
   {
      if (currentWeight + w[i] <= m)
      {
         x[i] = 1; // Item i is selected
         currentWeight += w[i];
         maxprofit += p[i];
      }
```

```
        else
        {
// Fractional part of item i is selected
            x[i] = (m - currentWeight) / (double)w[i];
            maxprofit += x[i] * p[i];
            break;
        }
    }
    printf("Optimal solution for greedy method: %.1f\n", maxprofit);
    printf("Solution vector for greedy method: ");
    for (i = 0; i < n; i++)
        printf("%d\t", x[i]);
}



int main()
{
    printf("Enter the number of objects: ");
    scanf("%d", &n);
    printf("Enter the objects' weights: ");
    for (i = 0; i < n; i++)
        scanf("%d", &w[i]);
    printf("Enter the objects' profits: ");
    for (i = 0; i < n; i++)
        scanf("%d", &p[i]);
    printf("Enter the maximum capacity: ");
    scanf("%d", &m);
    greedyKnapsack(n, w, p, m);
    return 0;
}
```

```
Enter the number of objects: 4
Enter the objects' weights: 56 78 98 78
Enter the objects' profits: 23 45 76 78
Enter the maximum capacity: 100
Optimal solution for greedy method: 78.0
Solution vector for greedy method: 1  0   0   0
```

**8. Design and implement C/C++ Program to find a subset of a given set S = {sl , s2, .....,sn} of n positive integers whose sum is equal to a given positive integer d.**

**Sum of Subsets**

Subset-Sum Problem is to find a subset of a given set S= {s1, s2… $s_n$} of n positive integers whose sum is equal to a given positive integer d. It is assumed that the set's elements are sorted in increasing order. The state-space tree can then be constructed as a binary tree and applying backtracking algorithm, the solutions could be obtained. Some instances of the problem may have no solutions

```
Algorithm SumOfSub(s, k, r)
//Find all subsets of w[1…n] that sum to m. The values of x[j], 1<= j < k, have already
 //been determined. s=∑(j=1 to k-1) w[j]*x[j]  and r =∑(j=k to n) w[j]. The w[j]'s are in ascending order.

{
    x[k] ← 1  //generate left child
    if (s+w[k] = m)
            write (x[1...n])  //subset found
    else if ( s + w[k]+w[k+1] <= m)
            SumOfSub( s + w[k], k+1, r-w[k])
    //Generate right child
    if( (s + r - w[k] >= m) and (s + w[k+1] <= m) )
    {
            x[k] ← 0
            SumOfSub( s, k+1, r-w[k] )
    }
}
```

Complexity: Subset sum problem solved using backtracking generates at each step maximal two new subtrees, and the running time of the bounding functions is linear, so the running time is O(2n ).

**Program:**

```
#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)
        {
           //if subset found
           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 )//left tree evaluation
           sum_of_subsets(s+w[k], k+1, rem-w[k]);

        if( ( s+rem-w[k] >= d) && ( s + w[k+1]) <= d)//right tree evaluation
        {
           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("eneter the sum:");
        scanf("%d",&d);

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

Sample Input and Output:

```
enter no of elements:5
enter the elements in increasing order:1 2 3 4 5
eneter the sum:10
subset = 1
1 2  3  4
subset =
 2 1 4  5
subset = 3
     2    3 5
```

9. **Design and implement C/C++ Program to sort a given set of n integer elements using Selection Sort method and compute its time complexity. Run the program for varied valuesof n> 5000 and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.**

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

int a[10000],n,count;void

selection_sort()
{
        for(int i=0;i<n-1;i++)
        {
                int min = i;
                for(int j=i+1;j<n;j++)
                {
                        count++;
                        if(a[j]<a[min])
                                min=j;
                }
                int temp=a[i];
                a[i]=a[min];
                a[min]=temp;
        }
}

int main()
{
        printf("Enter the number of elements in an array:");
        scanf("%d",&n);
        printf("All the elements:");
        srand(time(0));
        for(int i=0;i<n;i++)
        {
                a[i]=rand();
                printf("%d ",a[i]);
        }
        selection_sort();
        printf("\nAfter sorting\n");
        for(int i=0;i<n;i++)
                printf("%d  ", a[i]);
        printf("\nNumber of basic operations = %d\n",count);
```

}

Sample Input and Output:

```
Enter the number of elements in an array:5All
the elements:
24152 32742 28304 4804 22274
After sorting
4804 22274  24152 28304  32742
Number of basic operations = 10

Enter the number of elements in an array:10
Allthe elements:
24243 6017 4212 23217 16170 24802 1085 24280 9847 6392
After sorting
1085 4212  6017 6392  9847  16170  23217  24243  24280  24802
Number of basic operations = 45
```

**10. Design and implement C/C++ Program to sort a given set of n integer elements using Quick Sort method and compute its time complexity. Run the program for varied values of n> 5000 and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.**

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

int count=0;
int partition(int a[], int low,int high)
{
        int pivot=a[low],temp,i=low+1,j=high;
        while(1)
        {
           //Traverse i from left to right, segregating element of left group
           while(i<=high && a[i]<=pivot)//a[i]<=pivot used for avoiding multiple duplicates
           {
              i++; count++;
           }
           //Traverse j from right to left, segregating element of right group
           while(j>0 && a[j]>pivot)
           {
              j--; count++;
           }
           count+=2;
           //If grouping is incomplete
           if(i<j)
           {
              temp = a[i];
              a[i] = a[j];
              a[j] =temp;
           }
           else if(i>j)//If grouping is completed
           {
              temp = a[low];
              a[low] = a[j];
              a[j] = temp;
              return j;
           }
           else //Duplicate of Pivot found
              return j;
        }
}

void quicksort(int a[],int low, int high)
{
        int s;
        if(low<high)
        {
           //partition to place pivot element in between left and right group
           s = partition(a,low,high);
```

```
            quicksort(a,low,s-1);
            quicksort(a,s+1,high);
        }
}
int main()
{
            int a[10000],n;
            printf("Enter the number of elements in an array:");
            scanf("%d",&n);
            printf("All the elements:");
            srand(time(0));
            for(int i=0;i<n;i++)
            {
                    a[i]=rand();
                    printf("%d ",a[i]);
            }
            quicksort(a,0,n-1);
            printf("\nAfter sorting\n");
            for(int i=0;i<n;i++)
                    printf("%d  ", a[i]);
            printf("\nNumber of basic operations = %d\n",count);
}
```

Sample Input and Output:

```
Enter the number of elements in an array:5All
the elements:
24442 6310 12583 16519 22767
After sorting
6310 12583  16519 22767  24442
Number of basic operations = 18

Enter the number of elements in an array:10All
the elements:
24530 1605 3396 10868 6349 9906 12836 28823 21075 22418
After sorting
1605 3396  6349 9906  10868  12836  21075  22418  24530  28823
Number of basic operations = 44
```

**11. Design and implement C/C++ Program to sort a given set of n integer elements using Merge Sort method and compute its time complexity. Run the program for varied valuesof n> 5000, and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.**

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

int count=0;
void merge(int a[], int low,int mid,int high)
{
        int i,j,k,c[10000];

        i=low, j=mid+1, k=0;
        while((i<=mid) && (j<=high))
        {
                count++;
                //choose the least element and store in Temporary array 'C'
                if(a[i]<a[j])
                        c[k++]=a[i++];
                else
                        c[k++]=a[j++];
        }


        //Copy the remaining array elements from any one of sub-array
        while(i<=mid)
                c[k++]=a[i++];
        while(j<=high)
                c[k++]=a[j++];
        for(i=low,j=0;j<k;i++, j++)
                a[i]=c[j];
}

void merge_sort(int a[], int low, int high)
{
        int mid;
        if(low < high)
        {
                //Divide the given array into 2 parts
                mid=(low+high)/2;
                merge_sort(a,low,mid);
                merge_sort(a,mid+1,high);
                merge(a,low,mid,high);
        }
}

int main()
{
        int a[10000],n,i;
        printf("Enter the number of elements in an array:");
        scanf("%d",&n);
        printf("All the elements:");
```

```
        srand(time(0));
        for(i=0;i<n;i++)
        {
                a[i]=rand();
                printf("%d ",a[i]);
        }
        merge_sort(a,0,n-1);
        printf("\nAfter sorting\n");
        for(i=0;i<n;i++)
                printf("%d  ", a[i]);
        printf("\nNumber of basic operations = %d\n",count);
}
```

Sample Input and Output:

Enter the number of elements in an array:5All
the elements:
24759 329 8704 24132 7473
After sorting
329  7473  8704  24132  24759
Number of basic operations = 8

Enter the number of elements in an array:10All
the elements:
24854 17121 2477 1072 11684 5437 26057 1167 17322 3583
After sorting
1072 1167  2477 3583  5437  11684  17121  17322  24854  26057
Number of basic operations = 22

**12. Design and implement C/C++ Program for N Queen's problem using Backtracking.**

.

**Program:**

```
#include<stdio.h>
#include<math.h>            //for abs() function

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; //feasible
}
int nqueens(int n)
{
        int x[10], k, count=0;

        k=1;// select the first queen
        x[k]=0; //no positions allocated
        while(k != 0)  // until all queens are present
        {
                x[k]++;      // place the kth queen in next column
                while((x[k] <= n)  && (!place(x,k)))
                        x[k]++;     // check for the next column to place queen

                if(x[k] <= n)
                {
                        if(k == n)  // all queens are placed
                        {
                                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;                //select the next queen
                        x[k]=0;    // start from the next column
                        }
                }
                else
                        k--;            // backtrack
        }
```

```
        return count;
}

void main()
{
        int n;
        printf("Enter the size of chessboard: ");
        scanf("%d",&n);
        printf("\nThe number of possibilities are %d",nqueens(n));
}
```

### Sample Input and Output:

1. Enter the size of chessboard: 4

Solution 1
X Q X X
X X X Q
Q X X X
X X Q X

Solution 2
X X Q X
Q X X X
X X X Q
X Q X X

The number of possibilities are 2

2. Enter the size of chessboard: 3

The number of possibilities are 0

3. Enter the size of chessboard: 1

Solution 1
Q

The number of possibilities are 1