BRINDAVAN GROUP OF INSTITUTIONS

Brindavan College of Engineering

Dwarakanagar, Bagalur Main Road, Yelahanka, Bengaluru–560063
Affiliated to VTU Belagavi, Approved by AICTE, New Delhi, India
Accredited 'B++'level by NAAC
www.brindavancollege.com

IV Semester

ANALYSIS AND DESIGN OF ALGORITHMS LAB BCSL404

ACADEMIC YEAR
2023 – 2024

LABORATORYMANUAL

PREPARED BY

Prof. PADMAVATHI .R

BRINDAVAN GROUP OF INSTITUTIONS

Brindavan College of Engineering

Department of Information Science and Engineering

IV Semester

ANALYSIS AND DESIGN OF ALGORITHMS LAB
BCSL404

ACADEMIC YEAR
2023 – 2024

NAME OF THE STUDENT :.....

UNIVERSITY SEAT NO. :.....

BATCH :.....

PREPARED BY

Prof. PADMAVATHI .R

BRINDAVAN GROUP OF INSTITUTIONS

Brindavan College of Engineering

Department of Information Science and Engineering

LABORATORYCERTIFICATE

*This is to certify that Mr. /Ms.*_____
*Bearing USN*_____*of branch and semester has satisfactorily completed the course of experiments* ***in Analysis and Design of Algorithms Lab****, code* ***BCSL404*** *prescribed by the Visvesvaraya Technological University, Belagavi of this Institute for the academic year20–20.*

MARKS	
Maximum Marks	**Marks Obtained**

Signature of Faculty-In-Charge *Head of the Department*

Date:

Brindavan College of Engineering
Dwarakanagar, Bagalur Main Road, Yelahanka, Bengaluru–560063
Affiliated to VTU Belagavi, Approved by AICTE, New Delhi, India
Accredited 'B++'level by NAAC

Department of Information Science and Engineering

DEPARTMENT VISION

To create conductive environment for quality education and expertize the students in globalized technologies

DEPARTMENT MISSION

- To provide conducive academic environment with intellectual capabilities.
- To instill skill development in emerging.
- To impact professional responsibilities and commitments ethical standards and social concerns.

Brindavan College of Engineering
Dwarakanagar, Bagalur Main Road, Yelahanka, Bengaluru–560063
Affiliated to VTU Belagavi, Approved by AICTE, New Delhi, India
Accredited 'B++'level by NAAC

Department of Information Science and Engineering

Dos & DON'Ts IN LABORATORY

DOs

- Be on time and students should carry observation and completed records in all aspects.
- Dress code & wearing ID card is compulsory.
- Electronic gadgets are not allowed inside the lab.
- Students should be at their concerned desktop.
- After execution the students should get it verified by the concerned faculty.
- The executed results should be noted in their observations and get it verified by the concerned faculty.
- Observe good housekeeping practices. Keep the equipment's in proper place after the conduction.
- Students must ensure that all the switches are in the OFF position; desktop is shutdown properly after completion of the assignments.

DON'Ts

- Do not come late to lab.
- Do not touch server computer.
- Do not leave the lab without the permission of the faculty in-charge.
- Do not wear foot wear and enter the lab.
- Do not insert pen drive/memory card to any computer in the lab.
- Do not upload, delete or alter any software flies.

ANALYSIS & DESIGN OF ALGORITHMS LAB

Course Code: BCSL404 **CIE Marks: 50**

Teaching Hours/Week(L:T:P:S): 02 Hours **SEE Marks: 50**

Credits:01 **ExamHours:03**

Course objectives:

1. To design and implement various algorithms in C/C++ programming using suitable development tools to address different computational challenges.
2. To apply diverse design strategies for effective problem-solving.
3. To Measure and compare the performance of different algorithms to determine their efficiency and suitability for specific tasks.

Laboratory Experiments

SI.NO.	Experiments
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. 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.

DemonstrationExperiments(ForCIE)

SI.NO.	Experiments
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:

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

CONTENTS

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

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

int i, j, k, a, b, u, v, n, ne=1;
int min, mincost=0, adj[20][20], parent[20];

int find(int i)
{
        while(partne[i])
        i=parent[i];
        return i;
}

intuni(inti,int j)
{
        if(i<=j)
        {
                parent([j]=1);
                return 1;
        }
        return 0;
}

void main()
{
        printf("\n Enter no. of vertices:");
        scanf("%d", &n);

        printf("\n Enter the cost adjacency matrix:\n");

        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                {
                        scanf("%d", &adj[i][j]);
                        if(adj[i][j]==0)
                        adj[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(adj[i][j]<min)
                    {
                        min=adj[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;
            }
            adj[a][b]=adj[b][a]=999;
        }
        printf("\n\n Minimum Cost=%d\n", mincost);
}
```

OUTPUT:

Enter the no. of vertices:

5

Enter the cost adjacency matrix:

```
0    10  0   30  100
10   0   50   0    0
0    50   0   20   10
30   0   20   0    60
100  0  10  60    0
```

The edges of Minimum Cost Spanning tree are

1) (1,2) = 10

2) (3,5) = 10

3) (3,4) = 20

4) (1,4) = 30

Minimum Cost = 70

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>
#include<conio.h>
#define INFINITY 999
int prim(int cost[10][10],intsource,int n)
{
        inti,j,sum=0,visited[10];
        int distance[10],vertex[10];
        intmin,u,v;
for(i=1;i<=n;i++)
    {
        vertex[i]=source;
        visited[i]=0;
        distance[i]=cost[source][i];
    }
visited[source]=1;
for(i=1;i<n;i++)
    {
                min=INFINITY;
                for(j=1;j<=n;j++)
                {
                        if(!visited[j]&&distance[j]<min)
                        {
                                min=distance[j];
                                 u=j;
                         }
                }
        visited[u]=1;
        sum=sum+distance[u];
        printf("\n%d->%d",vertex[u],u);
        for(v=1;v<=n;v++)
            {
                        if(!visited[v]&&cost[u][v]<distance[v])
                        {
                                distance[v]=cost[u][v];
                                vertex[v]=u;
                        }
             }
    }
return sum;
}
void main()
{
int a[10][10],n,i,j,m,source;
clrscr();
```

```
printf("\n enter the  number of vertices:\n");
scanf("%d",&n);
printf("\n enter the cost matrix:\n 0-self loop and 999-no edge:\n");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
scanf("%d",&a[i][j]);
printf("\n enter the source:\n");
scanf("%d",&source);
  m=prim(a,source,n);
printf("\n the cost of spanning tree=%d",m);
getch();
}
```

OUTPUT:

Enter the number of vertices: 5

Enter the cost matrix 0-for self edge and 999-if no edge

```
0       3       4       999     5
3       0       999     6       1
4       999     0       9       7
999     6       9       0       2
5       1       7       2       0
```

Enter the source

```
2
2->5
5->4
2->1
1->3
```

Cost = 10

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

```
#include<stdio.h>
#include<conio.h>
void main()
{
   inti,j,k,a[20][20],n;
   clrscr();
   printf("\nEnter the value of n ");
   scanf("%d",&n);
   printf("\nEnter the adjacency matrix ");
   for(i=0;i<n;i++)
   {
      for(j=0;j<n;j++)
      {
         scanf("%d",&a[i][j]);
      }
   }
   for(k=0;k<n;k++)
   {
      for(j=0;j<n;j++)
      {
         for(i=0;i<n;i++)
         {
            a[i][j]=a[i][j]<(a[i][k]+a[k][j])?a[i][j]:(a[i][k]+a[k][j]);
         }
      }
   }
   printf("\nFloyd's shortest path is ");
   for(i=0;i<n;i++)
   {
      for(j=0;j<n;j++)
      {
         printf("%c%d",j==0?'\n':' ',a[i][j]);
      }
   }
   getch();
}
```

OUTPUT:

```
Enter the value of n: 5

Enter the adjacency matrix:
0 999 3 999 999
4 0 999 1 2
3 8 0 2 6
999 1 999 0 4
999 1 6 4 0

Floyd's shortest path is:
 0 6 3 5 8
 4 0 7 1 2
 3 3 0 2 5
 5 1 8 0 3
 5 1 6 2 0
```

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

```
#include<stdio.h>
#include<conio.h>
void main()
{
   intarr[20][20],i,j,k,n;
   clrscr();
   printf("\nEnter the number of nodes ");
   scanf("%d",&n);
   printf("\nEnter the adjacency matrix ");
   for(i=0;i<n;i++)
   {
      for(j=0;j<n;j++)
      {
         scanf("%d",&arr[i][j]);
      }
   }
   for(k=0;k<n;k++)
   {
      for(j=0;j<n;j++)
      {
         for(i=0;i<n;i++)
         {
            arr[i][j]=arr[i][j]||(arr[i][k]&&arr[k][j]);
```

```
        }
      }
    }
    printf("\nThe transitive closure formed by Warshalls algorithm is ");
    for(i=0;i<n;i++)
    {
      printf("\n");
      for(j=0;j<n;j++)
      {
        printf("%d ",arr[i][j]);
      }
    }
    getch();
}
```

OUTPUT:

Enter the value of n: 4
Enter the adjacency matrix:
0 1 0 0
0 0 0 1
0 0 0 0
1 0 1 0

The transitive closure formed by Warshall'salogorithm 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.

```
#include<stdio.h>
#include<conio.h>
#define INFINITY 999
voiddijkstra(int cost[10][10],intn,intsource,int distance[10])
{
        int visited[10],min,u;
        inti,j;
for(i=1;i<=n;i++)
{
        distance[i]=cost[source][i];
        visited[i]=0;
        }
visited[source]=1;
for(i=1;i<=n;i++)
{
        min=INFINITY;
        for(j=1;j<=n;j++)
        if(visited[j]==0 && distance[j]<min)
        {
                min=distance[j];
                u=j;
        }
        visited[u]=1;
        for(j=1;j<=n;j++)
if(visited[j]==0 && (distance[u]+cost[u][j])<distance[j])
        {
                distance[j]=distance[u]+cost[u][j];
```

```
                }
        }
}
void main()
{
        intn,cost[10][10],distance[10];
        inti,j,source,sum;
        clrscr();
        printf("\nEnter how many nodes :  ");
        scanf("%d",&n);
        printf("\nCost Matrix\nEnter 999 for no edge\n");
        for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
        scanf("%d",&cost[i][j]);
        printf("Enter the source node\n");
        scanf("%d",&source);
        dijkstra(cost,n,source,distance);
        for(i=1;i<=n;i++)
        printf("\n\nShortest Distance from  %d to %d is %d",source,i,distance[i]);
getch();
}
```

OUTPUT:

Enter how many nodes:4

Cost Matrix

Enter 999 for no edge

999	999	3	999
999	999	4	7
999	4	999	15
999	7	15	999

Enter the source node

1

Shortest Distance for 1 to 1 is 999

Shortest Distance for 1 to 2 is 7

Shortest Distance for 1 to 3 is 3

Shortest Distance for 1 to 4 is 14

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

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int a[20][20],rem[20],ind,n,i,j,flag=0,t=0;
   clrscr();
   printf("\nEnter the value of n ");
   scanf("%d",&n);
   printf("\nEnter the adjacency matrix ");
   for(i=0;i<n;i++)
   {
      rem[i]=0;
      for(j=0;j<n;j++)
      {
         scanf("%d",&a[i][j]);
      }
   }
   while(flag==0)
   {
   flag=1;
      for(i=0;i<n;i++)
      {
         if(rem[i]==0)
         {
            ind=0;
            for(j=0;j<n;j++)
            {
               if(!(rem[j]==1||a[j][i]==0))
               {
                  ind=1;
                  break;
               }
            }
            if(ind==0)
            {
               printf("%s",t==0?"\nTopological ordering is ":"");
               rem[i]=1;
               printf("%d ",i+1);
               flag=0;
               t++;
               break;
            }
         }
      }
```

```
    }
    if(t!=n)
    {
        printf("\nTopological ordering is not possible(it can only be partially ordered)!!");
    }
    getch();
}
```

OUTPUT:

Enter the value of n: 5

Enter the adjacency matrix:
0 1 0 0 0
0 0 0 1 1
0 1 0 0 0
0 0 0 0 1
0 0 0 0 0

Topological Ordering is 1 3 2 4 5

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

```
#include<stdio.h>
#include<conio.h>
int w[10],p[10],n;
int max(inta,int b)
{
        return a>b?a:b;
}
int knap(inti,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]);
}
void main()
{
        intm,i,max_profit;
        clrscr();
        printf("\nEnter the number of objects: ");
        scanf("%d",&n);
        printf("\nEnter the knapsack capacity: ");
        scanf("%d",&m);
        printf("\nEnter profit followed by weight: ");
        for(i=1;i<=n;i++)
        scanf("%d%d",&p[i],&w[i]);
        max_profit=knap(1,m);
        printf("\nMax profit = %d",max_profit);
        getch();
}
```

OUTPUT:

Enter the number of objects: 3

Enter the knapsack capacity:116

Enter the profit followed by weight:

100 12

12 15

20 30

Max profit=132

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];
doublemaxprofit;
int n, m, i;
voidgreedyKnapsack(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;
}
}
```

```
}
intcurrentWeight = 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;
}
```

OUTPUT:

```
Enter the no. of objects:4
Enter the object's weights: 2 3 4 5
Enter the object's profits: 1 2 5 6
Enter the maximum capacity: 8

Optimal solution for greedy method: 5
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.

```
#include<stdio.h>
#include<conio.h>#defi
neMAX10
ints[MAX],x[MAX],d;
voidsumofsub(intp,intk,intr)
{
        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
```

```
        printf("\nEnterthemaxsubsetvalue:");scanf("%d",&d);
        for(i=1;i<=n;i++)sum=sum+s[
        i];
        if(sum<d||s[1]>d)
            printf("\nNosubsetpossible");
        else


        sumofsub(0,1,sum);retur
        n0;
}
```

OUTPUT:

```
Enterthenvalue:9
Entertheset inincreasing order:123456789
Enterthemaxsubset value:9
126
135
18
234
27
36
45
9
```

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.

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

// Function to perform selection sort
voidselectionSort(intarr[], int n) {
int i, j, min_idx;

   // One by one move boundary of unsorted subarray
for (i = 0; i < n - 1; i++) {
      // Find the minimum element in unsorted array
min_idx = i;
for (j = i + 1; j < n; j++) {
if (arr[j] <arr[min_idx]) {
min_idx = j;
         }
      }
      // Swap the found minimum element with the first element
int temp = arr[min_idx];
arr[min_idx] = arr[i];
arr[i] = temp;
   }
}
```

```
int main() {
int n;
printf("Enter the number of elements: ");
scanf("%d", &n);
int *arr = (int *)malloc(n * sizeof(int));
printf("Enter %d elements: ", n);
for (int i = 0; i < n; i++) {
scanf("%d", &arr[i]);
    }
clock_tstart_time, end_time;
doubletotal_time;

start_time = clock(); // Recording the start time
selectionSort(arr, n);
end_time = clock(); // Recording the end time
total_time = ((double)(end_time - start_time)) / CLOCKS_PER_SEC;
printf("Sorted array: ");
for (int i = 0; i < n; i++) {
printf("%d ", arr[i]);
    }
printf("\n");
printf("Time taken for sorting: %f seconds\n", total_time);
    // Free the allocated memory
free(arr);
return 0;
}
```

OUTPUT:

Enter the value of n: 5

Enter the 5 elements: 34 25 90 12 45

Sorted array: 12 25 34 45 90

Time taken for sorting: 0.000001 seconds

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>

// Function to swap two elements
void swap(int *a, int *b) {
int temp = *a;
   *a = *b;
   *b = temp;
}

// Function to partition the array
int partition(intarr[], int low, int high) {
int pivot = arr[high];
int i = (low - 1);
for (int j = low; j <= high - 1; j++) {
if (arr[j] < pivot) {
i++;
swap(&arr[i], &arr[j]);
      }
   }
swap(&arr[i + 1], &arr[high]);
return (i + 1);
}

// Function to perform QuickSort
voidquickSort(intarr[], int low, int high) {
if (low < high) {
int pi = partition(arr, low, high);
quickSort(arr, low, pi - 1);
quickSort(arr, pi + 1, high);
   }
}
int main() {
int n;
printf("Enter the number of elements: ");
scanf("%d", &n);
```

```
intarr[n];
printf("Enter %d elements: ", n);
for (int i = 0; i < n; i++) {
scanf("%d", &arr[i]);
    }

clock_tstart_time, end_time;
doubletotal_time;

start_time = clock(); // Recording the start time

quickSort(arr, 0, n - 1);

end_time = clock(); // Recording the end time
total_time = ((double)(end_time - start_time)) / CLOCKS_PER_SEC;

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

printf("Time taken for sorting: %f seconds\n", total_time);

return 0;
}
```

OUTPUT:
Enter the number of elements: 6
Enter 6 elements: 5 23 7 1 56
Sorted array: 1 5 7 23 56
Time taken for sorting is: 0.000002 seconds

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.

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

// Function to merge two subarrays of arr[].
// First subarray is arr[l..m]
// Second subarray is arr[m+1..r]
void merge(intarr[], int l, int m, int r) {
int i, j, k;
int n1 = m - l + 1;
int n2 = r - m;

   // Create temporary arrays
int L[n1], R[n2];

   // Copy data to temporary arrays L[] and R[]
for (i = 0; i < n1; i++)
      L[i] = arr[l + i];
for (j = 0; j < n2; j++)
      R[j] = arr[m + 1 + j];

   // Merge the temporary arrays back into arr[l..r]
   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++;
   }

   // Copy the remaining elements of L[], if any
```

```
while (i < n1) {
arr[k] = L[i];
i++;
k++;
    }

    // Copy the remaining elements of R[], if any
while (j < n2) {
arr[k] = R[j];
j++;
k++;
    }
}

// Main function that sorts arr[l..r] using merge()
voidmergeSort(intarr[], int l, int r) {
if (l < r) {
        // Same as (l+r)/2, but avoids overflow for large l and r
int m = l + (r - l) / 2;

        // Sort first and second halves
mergeSort(arr, l, m);
mergeSort(arr, m + 1, r);

        // Merge the sorted halves
merge(arr, l, m, r);
    }
}

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

intarr[n];
printf("Enter %d integers: ", n);
for (int i = 0; i < n; i++) {

scanf("%d", &arr[i]);
    }

clock_tstart_time, end_time;
```

```
    doubletotal_time;

    start_time = clock(); // Recording the start time

    mergeSort(arr, 0, n - 1);

    end_time = clock(); // Recording the end time
    total_time = ((double)(end_time - start_time)) / CLOCKS_PER_SEC;

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

    printf("Time taken for sorting: %f seconds\n", total_time);

    return 0;
}
```

OUTPUT:

Enter the number of elements: 10

Enter 6 elements:456 342 2934 2386 5000 45 954 10 564 28

Sorted array: 10 28 45 342 456 564 954 2386 2934 5000

Time taken for sorting is: 0.000004 seconds

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

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
voidprintSolution(int x[], int n)
{
for (int i = 0; i < n; i++)
 {
printf("\n");
for (int j = 0; j < n; j++)
{
if (x[i] == j)
{
printf("Q ");
          }
else
{
printf("X ");
          }
       }
   }
printf("\n");
}

intisValid(int x[], int row, int col)
{
for (int i = 0; i < row; i++)
{
if (x[i] == col || abs(row - i) == abs(col - x[i]))
{
return 0;
       }
   }
return 1;
}

voidsolveNQueens(int n, int x[], int row, int *count)
{
if (row == n)
{
       *count += 1;
```

```
printSolution(x, n);
return;
    }
for (int col = 0; col < n; col++)
{
if (isValid(x, row, col))
{
x[row] = col;
solveNQueens(n, x, row + 1, count);
        }
    }
}

int main()
{
int n;
printf("Enter the total number of queens: ");
scanf("%d", &n);
if (n <= 0)
{
printf("Invalid input!\n");
return 1;
}

int x[n];
int count = 0;

solveNQueens(n, x, 0, &count);

if (count == 0) {
printf("No solution found!\n");
    } else {
printf("\nThe total number of solutions is %d.\n", count);
    }
return 0;
}
```

OUTPUT:

Enter the total number of Queen's: 4
X Q X X
X XX Q
Q X XX
X X Q X

X X Q X
Q X XX
X XX Q
X Q X X

The total number of solutions is 2.