VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“Jnana Sangama”, Belgaum-590018

“ANALYSIS AND DESIGN OF ALGORITHMS LAB MANUAL”

[BCSL404]

(IV SEM - CBCS SCHEME)

Prepared by:

Mr. Vinaykumar V N & Divya C

Assistant Professor
Department of AI & ML

Department of Artificial Intelligence
&
Machine Learning

Tiptur-572201

Analysis & Design of Algorithms Lab		Semester	4
Course Code	**BCSL404**	CIE Marks	50
Teaching Hours/Week (L:T:P: S)	0:0:2:0	SEE Marks	50
Credits	01	Exam Hours	2
Examination type (SEE)	Practical		

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.

Sl.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.
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.

Assessment Details (both CIE and SEE)

The weightage of Continuous Internal Evaluation (CIE) is 50% and for Semester End Exam (SEE) is 50%. The minimum passing mark for the CIE is 40% of the maximum marks (20 marks out of 50) and for the SEE minimum passing mark is 35% of the maximum marks (18 out of 50 marks). A student shall be deemed to have satisfied the academic requirements and earned the credits allotted to each subject/ course if the student secures a minimum of 40% (40 marks out of 100) in the sum total of the CIE (Continuous Internal Evaluation) and SEE (Semester End Examination) taken together

Continuous Internal Evaluation (CIE):

CIE marks for the practical course are **50 Marks**.

The split-up of CIE marks for record/ journal and test are in the ratio **60:40**.

- Each experiment is to be evaluated for conduction with an observation sheet and record write-up. Rubrics for the evaluation of the journal/write-up for hardware/software experiments are designed by the faculty who is handling the laboratory session and are made known to students at the beginning of the practical session.
- Record should contain all the specified experiments in the syllabus and each experiment write-up will be evaluated for 10 marks.
- Total marks scored by the students are scaled down to **30 marks** (60% of maximum marks).
- Weightage to be given for neatness and submission of record/write-up on time.
- Department shall conduct a test of 100 marks after the completion of all the experiments listed in the syllabus.
- In a test, test write-up, conduction of experiment, acceptable result, and procedural knowledge will carry a weightage of 60% and the rest 40% for viva-voce.
- The suitable rubrics can be designed to evaluate each student's performance and learning ability.
- The marks scored shall be scaled down to **20 marks** (40% of the maximum marks).

The Sum of scaled-down marks scored in the report write-up/journal and marks of a test is the total CIE marks scored by the student.

Semester End Evaluation (SEE):

- SEE marks for the practical course are 50 Marks.
- SEE shall be conducted jointly by the two examiners of the same institute, examiners are appointed by the Head of the Institute.
- The examination schedule and names of examiners are informed to the university before the conduction of the examination. These practical examinations are to be conducted between the schedule mentioned in the academic calendar of the University.
- All laboratory experiments are to be included for practical examination.
- (Rubrics) Breakup of marks and the instructions printed on the cover page of the answer script to be strictly adhered to by the examiners. **OR** based on the course requirement evaluation rubrics shall be decided jointly by examiners.
- Students can pick one question (experiment) from the questions lot prepared by the examiners jointly.
- Evaluation of test write-up/ conduction procedure and result/viva will be conducted jointly by examiners.

General rubrics suggested for SEE are mentioned here, writeup-20%, Conduction procedure and result in -60%, Viva-voce 20% of maximum marks. SEE for practical shall be evaluated for 100 marks and scored marks shall be scaled down to 50 marks (however, based on course type, rubrics shall be decided by the examiners)

Change of experiment is allowed only once and 15% of Marks allotted to the procedure part are to be made zero.

The minimum duration of SEE is 02 hours

Suggested Learning Resources:

- Virtual Labs (CSE): http://cse01-iiith.vlabs.ac.in/

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>
#define INFINITY 999
#define MAX 100
int parent[MAX],cost[MAX][MAX],t[MAX][2];

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

void union1(int i,int j)
{
      parent[j]=i;
}

void kruskal(int n)
{
      int i,j,k,u,v,mincost,res1,res2,sum=0;
      for(k=1;k<n;k++)
      {
            mincost=INFINITY;
            for(i=1;i<n;i++)
            {
                  for(j=1;j<=n;j++)
                  {
                        if(i==j) continue;

                        if(cost[i][j]<mincost)
                        {
                              u=find(i);
                              v=find(j);

                              if(u!=v)
                              {
                                    res1=i;
                                    res2=j;
                                    mincost=cost[i][j];
                              }
                        }
                  }
            }
            union1(res1,find(res2));
            t[k][1]=res1;
            t[k][2]=res2;
            sum=sum+mincost;
      }
      printf("\nCost of spanning tree is %d\n",sum);
      printf("\nEdges of spanning tree are\n");
```

```
	for(i=1;i<n;i++)
	printf("%d->%d\n",t[i][1],t[i][2]);
}

void main()
{
	int i,j,n;
	printf("\nEnter the number of vertices : ");
		scanf("%d",&n);

	for(i=1;i<=n;i++)
		parent[i]=0;
	printf("\nEnter the cost adjacency matrix 0-for self edge and 999-if no edge\n");

	for(i=1;i<=n;i++)
		for(j=1;j<=n;j++)
			scanf("%d",&cost[i][j]);
	kruskal(n);
}
```

Output:

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 INFINITY 999

int prim(int cost[10][10],int source,int n)
{
      int i,j,sum=0,visited[10];
      int distance[10],vertex[10];
      int min,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;
      printf("\n enter the number of vertices:\n");
            scanf("%d",&n);
      printf("\nenter 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);
}
```

Output:

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

```
#include<stdio.h>
#define INFINITY 999
int min(int i,int j)
{
	if(i<j)
		return i;
	else
		return j;
}

void floyd(int n,int p[10][10])
{
	int i,j,k;
	for(k=1;k<=n;k++)
		for(i=1;i<=n;i++)
			for(j=1;j<=n;j++)
				p[i][j]=min(p[i][j],p[i][k]+p[k][j]);
}

int main()
{
	int i,j,n,a[10][10],d[10][10],source;
	double starttime,endtime;
	printf("Enter the no of nodes: ");
		scanf("%d",&n);
	printf("\nEnter the adjacency matrix\n");
	for(i=1;i<=n;i++)
		for(j=1;j<=n;j++)
			scanf("%d",&a[i][j]);

	floyd(n,a);

	printf("\n\nThe distance matrix is \n");
	for(i=1;i<=n;i++)
	{
		for(j=1;j<=n;j++)
			printf("%d\t",a[i][j]);
		printf("\n");
	}
	return 0;
}
```

Output:

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

```
#include<stdio.h>
void warshall(int p[10][10],int n)
{
      int i,j,k;
      for(k=1;k<=n;k++)
      {
            for(i=1;i<=n;i++)
            {
                  for(j=1;j<=n;j++)
                  {
                        if(p[i][k]==1 && p[k][j]==1)
                        {
                              p[i][j]=1;
                        }
                  }
            }
      }
}

void main()
{
      int a[10][10],i,j,n;
      printf(" enter the no of vertices:\n");
            scanf("%d",&n);
      printf(" enter the adjacency matrix:\n");
      for(i=1;i<=n;i++)
      {
            for(j=1;j<=n;j++)
            {
                  scanf("%d",&a[i][j]);
            }
      }

      warshall(a,n);

      printf(" the resultant path matrix:\n");
      for(i=1;i<=n;i++)
      {
            for(j=1;j<=n;j++)
            {
                  printf("%d\t",a[i][j]);
            }
            printf("\n");
      }
}
```

Output:

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>
#define INFINITY 999
void dijkstra(int cost[10][10],int n,int source,int distance[10])
{
      int visited[10],min,u;
      int i,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()
{
      int n,cost[10][10],distance[10];
      int i,j,source,sum;
      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]);
}
```

Output:

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

```
#include<stdio.h>
int temp[10],k=0;

void topo(int n,int indegree[10],int a[10][10])
{
	int i,j;
	for(i=1;i<=n;i++)
	{
		if(indegree[i]==0)
		{
			indegree[i]=-1;
			temp[++k]=i;
			for(j=1;j<=n;j++)
			{
				if(a[i][j]==1 && indegree[j]!=-1)
					indegree[j]--;
			}
			i=0;
		}
	}
}

void main()
{
	int i,j,n,indegree[10],a[10][10];
	printf("Enter the number of vertices: ");
		scanf("%d",&n);
	for(i=1;i<=n;i++)
		indegree[i]=0;
	printf("Enter the adjacency matrix\n");
	for(i=1;i<=n;i++)
		for(j=1;j<=n;j++)
		{
			scanf("%d",&a[i][j]);
			if(a[i][j]==1)
				indegree[j]++;
		}
		topo(n,indegree,a);
		if(k!=n)
			printf("\nTopological ordering is not possible\n");
		else
		{
			printf("The topological ordering is \n");
			for(i=1;i<=k;i++)
				printf("%d\t",temp[i]);
		}
}
```

Output:

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]);
}

void main()
{
      int m,i,max_profit;
      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);
}
```

Output:

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
float 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];
    for (i = 0; i < n; i++)
    {
        ratio[i] = (double)p[i] / w[i];
    }
    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;
    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
        {
            x[i] = (m - currentWeight) / (double)w[i];
            maxprofit += x[i] * p[i];
            break;
        }
    }
    printf("Optimal solution for greedy method: %.1f\n", maxprofit);

}
```

```
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

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>
#define MAX 10

int s[MAX], x[MAX];
int d;  // Target sum

void sumofsub(int p, int k, int r)
{
    int i;

    // Include the current element
    x[k] = 1;

    // Check if the sum matches the target value
    if (p + s[k] == d)
    {
        printf("{");
        for (i = 1; i <= k; i++)
        {
            if (x[i] == 1)
                printf("%d,", s[i]);
        }
        printf("\b}\n");  // Remove the last comma and close the brace
    }
    else if (p + s[k] + s[k + 1] <= d)
    {
        // Recurse by including the current element
        sumofsub(p + s[k], k + 1, r - s[k]);
    }

    // Exclude the current element
    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("\nEnter the max subset value: ");
      scanf("%d", &d);

    // Calculate the total sum of the set
    for (i = 1; i <= n; i++)
        sum += s[i];

    if (sum < d || s[1] > d)
    {
        printf("\nNo subset possible\n");
    }
    else
    {
        sumofsub(0, 1, sum);  // Start the recursion
    }
}
```

Output:

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 <time.h>

void selsort(int a[], int n) {
      int i, j, small, pos, temp;
      for (j = 0; j < n - 1; j++) {
            small = a[j];
            pos = j;
            for (i = j + 1; i < n; i++) {
                  if (a[i] < small) {
                        small = a[i];
                        pos = i;
                  }
            }
            temp = a[j];
            a[j] = a[pos];
            a[pos] = temp;
      }
}

int main() {
      int a[10], i, n;
      struct timespec start, end;
      double dura;
      printf("\nEnter the n value: ");
            scanf("%d", &n);
      printf("\nEnter the array: ");
      for (i = 0; i < n; i++)
            scanf("%d", &a[i]);
      clock_gettime(CLOCK_MONOTONIC, &start);

      selsort(a, n);

      clock_gettime(CLOCK_MONOTONIC, &end);
      dura = (end.tv_sec - start.tv_sec) + (end.tv_nsec - start.tv_nsec) / 1e9;
      printf("\nTime taken is: %lf seconds", dura);
      printf("\nSorted array is: ");
      for (i = 0; i < n; i++)
            printf("%d ", a[i]);
      printf("\n");
      return 0;
}
```

Output:

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 partition(int a[], int low, int high) {
    int i, j, temp, key;
    key = a[low];
    i = low;
    j = high + 1;
    while (i <= j) {
        do i++; while (i <= high && key >= a[i]);
        do j--; while (key < a[j]);
        if (i < j) {
            temp = a[i];
            a[i] = a[j];
            a[j] = temp;
        }
    }
    temp = a[low];
    a[low] = a[j];
    a[j] = temp;
    return j;
}

void quicksort(int a[], int low, int high) {
    int mid;
    if (low < high) {
        mid = partition(a, low, high);
        quicksort(a, low, mid - 1);
        quicksort(a, mid + 1, high);
    }
}

int main() {
    int a[1000];
    int n, i;
    clock_t start, end;

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

    printf("Enter the array elements:\n");
    for (i = 0; i < n; i++)
        a[i] = rand() % 100;
    printf("Unsorted array:\n");
    for (i = 0; i < n; i++)
        printf("%d\t", a[i]);
    printf("\n");
    start = clock();
    quicksort(a, 0, n - 1);
    end = clock();

    printf("The sorted elements are:\n");
    for (i = 0; i < n; i++)
        printf("%d\t", a[i]);
    printf("\n");

    // Calculate time taken
   printf("The time taken is %f seconds\n", (double)(end - start) / CLOCKS_PER_SEC);

    return 0;
}
```

Output:

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>
#include<omp.h>

void merge(int a[],int low,int mid,int high)
{
      int b[1000],i,j,k;
      i=low;
      j=mid+1;
      k=low;

      while(i<=mid && j<=high)
      {
            if(a[i]<a[j])
            {
                  b[k]=a[i];
                  k++;
                  i++;
            }
            else
            {
                  b[k]=a[j];
                  k++;
                  j++;
            }
      }

      while(i<=mid)
      {
            b[k]=a[i];
            k++;
            i++;
      }

      while(j<=high)
      {
            b[k]=a[j];
            k++;
            j++;
      }

      for(i=low;i<=high;i++)
      {
            a[i]=b[i];
      }
}
```

```
void mergesort(int a[],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);
      }
}

void main()
{
      int a[1000],n,i;
      clock_t start,end;
      printf("Enter the number of elements:\n");
            scanf("%d",&n);
      printf("Enter the array elements:\n");
      for(i=0;i<n;i++)
      {
            a[i]=rand()%100;
      }
      printf("The elements before sorting are:\n");
      for(i=0;i<n;i++)
      {
            printf("%d\t",a[i]);
      }
      start=clock();
      mergesort(a,0,n-1);
      end=clock();
      printf("\nThe sorted elements are:\n");
      for(i=0;i<n;i++)
      {
            printf("%d\t",a[i]);
      }
      printf("the time taken is %f",(double)(end-start)/(CLOCKS_PER_SEC));
}
```

Output:

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

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 50

int can_place(int c[],int r)
{
	int i;
	for(i=0;i<r;i++)
		if(c[i]==c[r] || (abs(c[i]-c[r])==abs(i-r)))
			return 0;
	return 1;
}

void display(int c[],int n)
{
	int i,j;
	char cb[10][10];
	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';
	for(i=0;i<n;i++)
	{
		for(j=0;j<n;j++)
			printf("%c",cb[i][j]);
		printf("\n");
	}
}

void n_queens(int n)
{
	int r;
	int c[MAX];
	c[0]= -1;
	r=0;
	while(r>=0)
	{
		c[r]++;
		while(c[r]<n && !can_place(c,r))
			c[r]++;
		if(c[r]<n)
		{
			if(r==n-1)
			{
				display(c,n);
				printf("\n");
			}
			else
			{
				r++;
				c[r]=-1;
			}
		}
```

```
            else
                  r--;
      }
}

void main()
{
      int n;
      printf("\nEnter the number of queens : ");
            scanf("%d",&n);

      n_queens(n);
}
```

Output: