Department of Information Science and Engineering

Laboratory Manual

Design and Analysis of Algorithm LAB

18IS408

Course Outcomes:

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

Sl.No	Course Outcome	Bloom's Taxonomy Level (BTL)
C408.1	Understand the basics of algorithm design and analyse efficiency	2
C408.2	Develop and analyse algorithms using brute force and divide and conquer technique to solve problem	4
C408.3	Design and analyse algorithms using decrease and conquer, transform and conquer technique to solve problem	4
C408.4	Design and analyse algorithms using dynamic programming to solve problem	4
C408.5	Design and analyse algorithms using greedy technique, backtracking and branch and bound technique to solve problem	4

Software used

1. Code Blocks

Marks distribution

Internal Evaluation

Evaluation Criteria	Marks
CIE	20
Project*(if any)	10
MSE	20

SEE

Evaluation Criteria	Marks
Program write up	(5+5)
Program execution	(10+20)
Viva-Voce	10

Guidelines for mini project (if applicable)

1. Mini project must be carried out by a team of two members.

2. Mini project must be carried out to solve any relevant general purpose programming problem.

3. Implement the mini project using the C/C++ concepts in a meaningful way.

4. Evaluation is based on problem addressed, use of the programming concepts and the user experience of the mini project.

Prepared by:

Name of the faculty	**Designation**	**Unit**
Ms. Deepa	Assistant Professor	UNIT-I, UNIT-II, UNIT III, UNIT-IV and UNIT V

List of Experiments

*Programs are not restricted to only the below set.

Sl.No.	Title of the experiment	Page Number
	UNIT-I	
1.	Write a program to implement Euclids algorithm, middle school procedure and consecutive integer checking to compute GCD of two numbers.	5
2.	Write a program to find prime numbers using sieves method.	
3.	Write a program to find the uniqueness of an array.	6
4.	Write a program to find the factorial of a given number.	6
5.	Write a program to find Fibonacci series.	7
6.	Write a program to calculate maximum element.	7
	UNIT-II	
1.	Write a program to implement Sequential search and determine the time required to search an element.	9
2.	Write a program to sort a given set of elements using Selection sort and determine the time required to sort elements.	10
3.	Write a program to sort a given set of elements using the Bubble sort method and determine the time required to sort the elements.	10
4.	Write a program to implement Brute Force String Matching Technique and determine the time required.	11
5.	Write a program to sort a given set of elements using Merge sort method and determine the time required to sort the elements.	
6.	Write a program to sort a given set of elements using Quick sort method and determine the time required sort the elements.	
7.	Write a program to implement Recursive Binary search and determine the time required to search an element.	
8.	Write a program to implement Strassen's matrix multiplication and determine the time required.	
	UNIT-III	
1.	Write a program to sort a given set of elements using the Insertion sort method and determine the time required to sort the elements.	12
2.	Write a program to check whether a given graph is connected or not using DFS method and determine the time required.	12
3.	Write a program to print all the nodes reachable from a given starting node in a digraph using BFS method and determine the time required.	12
4.	Write a program to find the Topological sequence of vertices for the given graph and determine the time required	13

5.	Write a program to sort a given set of elements using the Heap sort method and determine the time required to sort the elements	13
	UNIT-IV	
1.	Write a program to sort a given set of elements using the Sorting by counting method and determine the time required to sort the elements.	17
2.	Write a program to sort a given set of elements using the Distribution counting method and determine the time required to sort the elements.	17
3.	Write a program to implement Horspool's algorithm for String Matching and determine the time required.	18
4.	Write a program to find the Binomial Coefficient using Dynamic Programming and determine the time required.	19
5.	Write a program to compute the transitive closure of a given directed graph using Warshall's algorithm and determine the time required.	21
6.	Write a program to implement Floyd's algorithm for the All-Pairs-Shortest-Paths problem and determine the time required.	22
7.	Write a program to implement 0/1 Knapsack problem using dynamic programming and determine the time required.	24
	UNIT-V	
1.	Write a program to find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm and determine the time required.	26
2.	Write a program to find Minimum Cost Spanning Tree of a given undirected graph using kruskals algorithm and determine the time required.	27
3.	From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm and determine the time required	28
4.	Write a program to implement N Queen's problem using Back Tracking method and determine the time required.	28
5.	Write a 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. For example, if S= {1, 2, 5, 6, 8} and d = 9 .If there are two solutions{1,2,6}and{1,8}.A suitable message is to be displayed if the given problem instance doesn't have a solution and determine the time required.	29

LESSON PLAN

Course Title: **DESIGN AND ANALYSIS OF ALGORITHMS LAB**	*Course Code:* **18IS408**

Hrs. /Week: **0+0+2+0**	*Credits:* **01**
Total Contact Hours: 26	*Duration of SEE:* **3 hrs**
SEE Marks: **50**	*CIE Marks:* **50**
Course Plan Author: Deepa	*Date:*
Checked By: **Dr. Karthik Pai B H**	*Date:*

Sl. No.	Programs	Tentative Dates (dd /mm/yy)					
		A1	A2	B1	B2	C1	C2
.1	Write a program to implement Euclids algorithm, middle school procedure and consecutive integer checking to compute GCD of two numbers.	31/12/19	2/1/20	30/12/19	1/1/20	1/1/20	30/12/19
.2	Write a program to find prime numbers using sieves method.	7/1/20	9/1/20	6/1/20	8/1/20	8/1/20	6/1/20
.3	Write a program to find the uniqueness of an array.	14/1/20	16/1/20	13/1/20	15/1/20	15/1/20	13/1/20
.4	Write a program to find the factorial of a given number.	14/1/20	16/1/20	13/1/20	15/1/20	15/1/20	13/1/20
.5	Write a program to find Fibonacci series.	14/1/20	16/1/20	13/1/20	15/1/20	15/1/20	13/1/20
.6	Write a program to calculate maximum element.	14/1/20	16/1/20	13/1/20	15/1/20	15/1/20	13/1/20
.7	Write a program to implement Sequential search and determine the time required to search an element.	21/1/20	23/1/20	20/1/20	22/1/20	22/1/20	20/1/20
.8	Write a program to sort a given set of elements using Selection sort and determine the time required to sort elements.	21/1/20	23/1/20	20/1/20	22/1/20	22/1/20	20/1/20
.9	Write a program to sort a given set of elements using the Bubble sort method and determine the time required to sort the elements.	21/1/20	23/1/20	20/1/20	22/1/20	22/1/20	20/1/20
.10	Write a program to implement Brute Force String Matching Technique and determine the time required.	28/1/20	30/1/20	27/1/20	29/1/20	29/1/20	27/1/20
.11	Write a program to sort a given set of elements using Merge sort method and determine the time required to sort the elements.27/1/20	28/1/20	30/1/20	27/1/20	29/1/20	29/1/20	27/1/20
.12	Write a program to sort a given set of elements using Quick sort method and determine the time required sort the elements.	28/1/20	30/1/20	27/1/20	29/1/20	29/1/20	27/1/20
.13	Write a program to implement Recursive Binary search and determine the time required to search an element.	11/2/20	13/2/20	10/2/20	12/2/20	12/2/20	10/2/20
.14	Write a program to implement Strassen's matrix multiplication and determine the time required.	11/2/20	13/2/20	10/2/20	12/2/20	12/2/20	10/2/20
.15	Write a program to sort a given set of elements using the Insertion sort method and determine the time required to sort the elements.	25/2/20	20/2/20	24/2/20	19/2/20	19/2/20	24/2/20
.16	Write a program to check whether a given graph is connected or not using DFS method and determine the time required.	25/2/20	20/2/20	24/2/20	19/2/20	19/2/20	24/2/20
.17	Write a program to print all the nodes reachable from a given starting node in a digraph using BFS method and determine the time required.	25/2/20	20/2/20	24/2/20	19/2/20	19/2/20	24/2/20

.18	Write a program to find the Topological sequence of vertices for the given graph and determine the time required	2/3/20	27/2/20	2/3/20	26/2/19	26/2/19	1/3/20
.19	Write a program to sort a given set of elements using the Heap sort method and determine the time required to sort the elements	2/3/20	5/3/20	2/3/20	4/3/20	4/3/20	1/3/20
.20	Write a program to sort a given set of elements using the Sorting by counting method and determine the time required to sort the elements.	2/3/20	5/3/20	2/3/20	4/3/20	4/3/20	1/3/20
.21	Write a program to sort a given set of elements using the Distribution counting method and determine the time required to sort the elements.	2/3/20	5/3/20	2/3/20	4/3/20	4/3/20	1/3/20
.22	Write a program to implement Horspool's algorithm for String Matching and determine the time required.	9/3/20	10/3/20	8/3/20	9/3/20	9/3/20	8/3/20
.23	Write a program to find the Binomial Coefficient using Dynamic Programming and determine the time required.	9/3/20	10/3/20	8/3/20	9/3/20	9/3/20	8/3/20
.24	Write a program to compute the transitive closure of a given directed graph using Warshall's algorithm and determine the time required.	9/3/20	10/3/20	8/3/20	9/3/20	9/3/20	8/3/20
.25	Write a program to implement Floyd's algorithm for the All-Pairs-Shortest-Paths problem and determine the time required.	9/3/20	10/3/20	8/3/20	9/3/20	9/3/20	8/3/20
.26	Write a program to implement 0/1 Knapsack problem using dynamic programming and determine the time required.	24/320	26/3/20	23/3/20	25/3/20	25/3/20	23/3/20
.27	Write a program to find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm and determine the time required.	24/3/20	26/3/20	23/3/20	25/3/20	25/3/20	23/3/20
.28	Write a program to find Minimum Cost Spanning Tree of a given undirected graph using kruskals algorithm and determine the time required.	31/3/20	2/4/20	30/3/20	1/4/20	1/4/20	30/3/20
.29	From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm and determine the time required	7/4/20	9/4/20	6/4/20	8/4/20	8/4/20	6/4/20
.30	Write a program to implement N Queen's problem using Back Tracking method and determine the time required.	14/4/20	16/4/20	13/4/20	15/4/20	15/4/20	13/4/20
.31	Write a 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. For example, if S= {1, 2, 5, 6, 8} and d = 9 .If there are two solutions{1,2,6}and{1,8}.A suitable message is to be displayed if the given problem instance doesn't have a solution and determine the time required.	14/4/20	16/4/20	13/4/20	15/4/20	15/4/20	13/4/20

Faculty Coordinator

Head-ISE

Unit - I

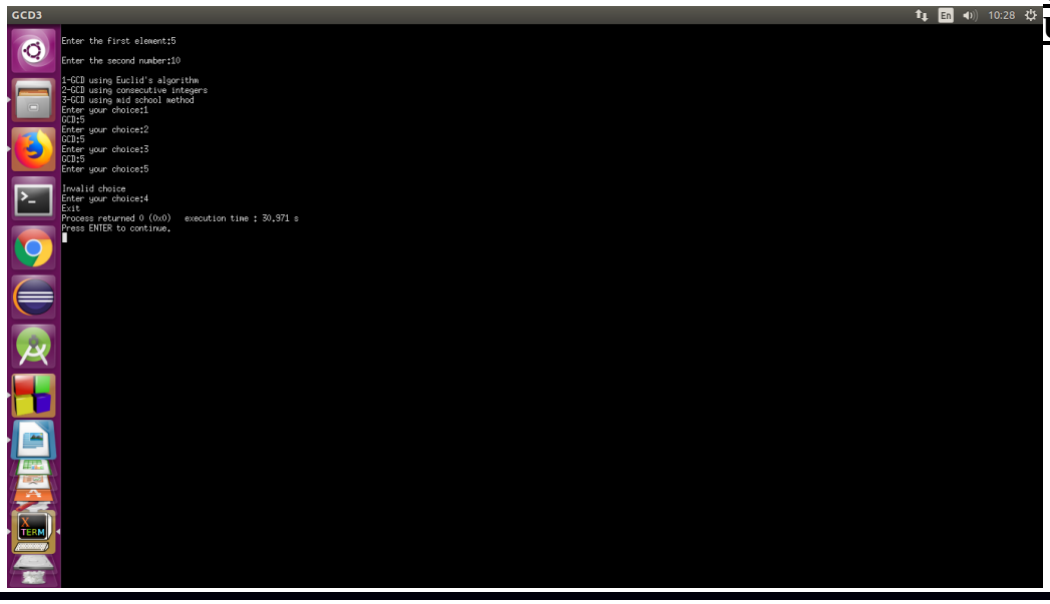
1. Write a program to implement Euclids algorithm, middle school procedure and consecutive integer checking to compute GCD of two numbers.

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

int euclid_alg(int m,int n)
{
   int r;
   while(n!=0)
   {
      r=m%n;
      m=n;
      n=r;
   }
   return(m);
}
int min(int a,int b)
{
   if(a>b)
      return b;
   else
      return a;
}
int consecutive_int(int m,int n)
{
   int t;
   if(m==0)
      return n;
   if(n==0)
      return m;
   t=min(m,n);
   while(t!=0)
   {
      if(m%t==0)
      {
         if(n%t==0)
         {
            return(t);
         }
```

```
        }
        t--;
    }
    return(-1);
}
int mid_school(int m,int n)
{
    int a=2,g=1;
    if(m==0)
        return n;
    if(n==0)
        return m;
    while((m>=a&&m!=0)&&(n>=a&&n!=0))
    {
        if(m%a==0)
        {
            if(n%a==0)
            {
                g=g*a;
                n=n/a;

            }
            m=m/a;
        }
        else
            a++;

    }
    return g;
}
int main()
{
    int m,n,ch,gcd;
    printf("\nEnter the first element:");
    scanf("%d",&m);
    printf("\nEnter the second number:");
    scanf("%d",&n);
    printf("\n1-GCD using Euclid's algorithm");
    printf("\n2-GCD using consecutive integers");
    printf("\n3-GCD using mid school method");

    while(1)
    {
        printf("\nEnter your choice:");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:gcd=euclid_alg(m,n);
                   printf("GCD:%d",gcd);
                   break;
```

```
        case 2:gcd=consecutive_int(m,n);
               printf("GCD:%d",gcd);
               break;
        case 3:gcd=mid_school(m,n);
               printf("GCD:%d",gcd);
               break;
        case 4:printf("Exit");
               exit(0);
               break;
        default:printf("\nInvalid choice");
      }
   }
   return 0;
}
```

OUTPUT:



2. Write a program to find prime numbers using sieves method.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
int main()
{
   int a[100],b[100],n,i,p,j,k;
   printf("\nEnter the number:");
   scanf("%d",&n);
   if(n>1)
   {
   for(i=2;i<=n;i++)
      a[i]=i;
```

```
   for(p=2;p<=sqrt(n);p++)
   {
      if(a[p]!=0)
      {
         j=p*p;
         while(j<=n)
         {
            a[j]=0;
            j=j+p;
         }
      }
   }
   }
   i=0;
   for(p=2;p<=n;p++)
   {
      if(a[p]!=0)
      {
         b[i]=a[p];
         i++;
      }
   }

   for(k=0;k<i;k++)
      printf("%d\t",b[k]);

   return(0);
}
```

Output:

```
Enter the number:
10
The prime numbers are
3 2 5 7
```

3. Write a program to check uniqueness of array.

```
int main()
{
int a[100],i,j,n,temp;
printf(“size of array”);
scanf(“%d”,&n);
printf(“elements are”);
for(i=0;i<n;i++)
scanf(“%d”,&a[i]);
for(i=0;i<n-2;i++)
{
for(j=i+1;j<n-1;j++)
```

```
{
if(a[i]==a[j])
{
printf(“not unique”);
exit(0);
}
else
{
printf(“unique”);
exit(0);
}
}
}return 0;
}
```

Output:

```
Array size:
4
elements
1 2 3 4
unique
```

4. Write a program to find factorial of a number

```
int factorial(int n)
{
if(n==0) return 0;
return(n*factorial(n-1));
}
int main()
{
int n fact;
printf(“enter any number”);
scanf(“%d”,&n);
fact=factorial(n)_;
printf(“%d”,fact);
}
```

Output:

```
Enter any number
5
Factorial of 5
120
```

5. Write a program to find Fibonacci number.

```
Int ifb(int n)
```

```
{
if(n<2) return n;
return (fib(n-1)+fib(n-2));
}
int main()
{
int n, fibnum;
printf("enter number");
scanf("%d",&n);
fibnum=fib(n);
printf(fib number:%d",fibnum);
}
```

Output:

```
Enter number
6
Fib Number
8
```

6. Write a program to calculate maximum element of an array.

```
int max(int a[100], i9nt n)
{
max =a[0];
for(i=1;i<=n-1;i++)
{
if(a[i]>max)
max=a[i];
}
printf("max element is %d", a[max]);
}
int main()
{
int n, a[100],i;
printf("enter the size of array");
scanf("%d",&n);
printf("enter elements of array");
for(i=0;i<n;i++)
scanf("%d",&a[i]);
max=(a,n);
}
```

Output:

```
Enter the size of array
5
enter array elements
10 35 12 150 90
max element
```

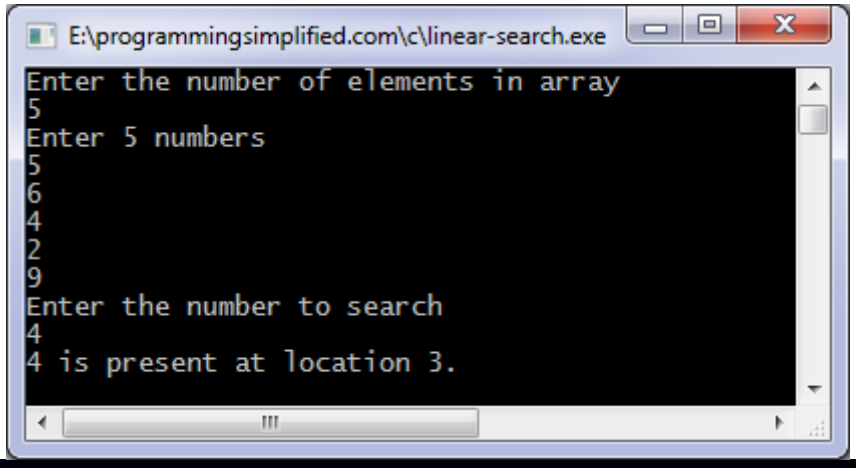
150

UNIT-II

1. Write a program to implement Sequential search and determine the time required to search an element.

```
#include <stdio.h>
int main()
{
   int array[100], search, c, n;
   printf("Enter the number of elements in array\n");
   scanf("%d",&n);

   printf("Enter %d integer(s)\n", n);
   for (c = 0; c < n; c++)
      scanf("%d", &array[c]);
   printf("Enter the       {number to search\n");
         printf("%d is present at location %d.\n", search, c+1);
         break;
      }
   }
   if (c == n)
      printf("%d is not present in array.\n", search);
   return 0;
}
```

Output:



Test Cases:

1.Check for number in first position
2. Check for number in last position
3.Check search unsuccessful

2. Sort a given set of elements using Selection sort and determine the time required to sort elements.

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int i,position,swap,a[100],d,n;
printf("Enter The No. Of Elements\n");
scanf("%d",&n);
printf("Enter %d Integers\n",n);
for(i=0;i<n;i++)
{
scanf("%d",&a[i]);
}
for(i=0;i<(n-1);i++)
{
position=i;
for(d=c+1;d<n;d++)
{
if(array[position]>array[d])
position=d;
}
if(position!=i)
{
swap=a[i];
a[i]=a[position];
a[position]=swap;
}
```

```
}
printf("Sorted List in Ascending Order is:\n");
for(i=0;i<=n;i++)
{
printf("%d\t",a[i]);
}
getch();
}
```

Output

Test Cases:

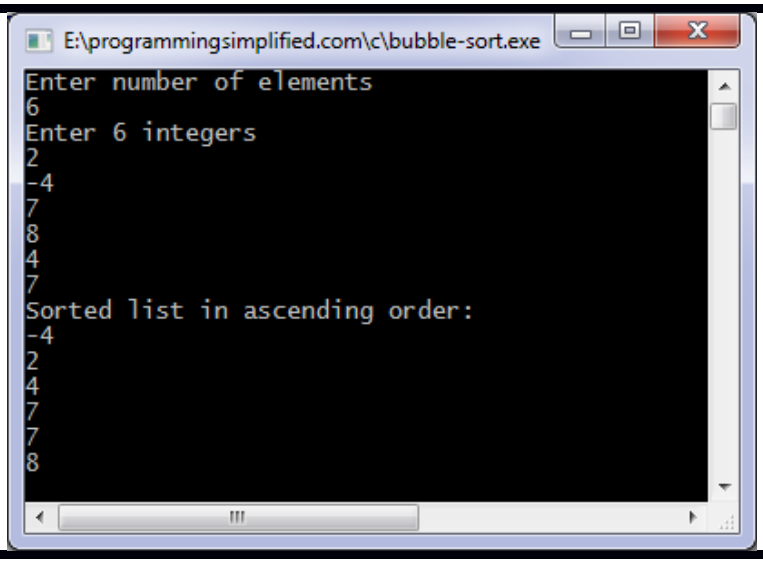
1. Check for array of size 5
2. Check for array of size 20
3. Check for array sorted in descending order

3. Sort a given set of elements using the Bubble sort method and determine the time required to sort the elements.

```
int main()
{
 int array[100], n, c, d, swap;
 printf("Enter number of elements\n");
 scanf("%d", &n);
 printf("Enter %d integers\n", n);
 for (c = 0; c < n; c++)
  scanf("%d", &array[c]);
 for (c = 0 ; c < ( n - 1 ); c++)
 {
  for (d = 0 ; d < n - c - 1; d++)
  {
   if (array[d] > array[d+1]) /* For decreasing order use < */
   {
    swap       = array[d];
    array[d]   = array[d+1];
    array[d+1] = swap;
```

```
    }
   }
  }
 printf("Sorted list in ascending order:\n");
 for ( c = 0 ; c < n ; c++ )
    printf("%d\n", array[c]);
 return 0;
}
```

Output:



Test Cases:

1. Check for array of size 5
2. Check for array of size 20
3. Check for array sorted in descending order

4. Implement Brute Force String Matching Technique and determine the time required.

```
#include <stdio.h>
 #include <string.h>  #define MAX 100

 /* try to find the given pattern in the search string */
 int bruteForce(char *search, char *pattern, int slen, int plen) {
 int i, j, k;

for (i = 0; i <= slen - plen; i++) {
 for (j = 0, k = i; (search[k] == pattern[j]) &&(j < plen); j++, k++);
if (j == plen)
 return j;}
 return -1;
 }
 int main() {
char searchStr[MAX], pattern[MAX];
 int res;
printf("Enter Search String:");
 fgets(searchStr, MAX, stdin);
```

```
printf("Enter Pattern String:");
 fgets(pattern, MAX, stdin);
 searchStr[strlen(searchStr) - 1] = '\0';
pattern[strlen(pattern) - 1] = '\0';
res = bruteForce(searchStr, pattern, strlen(searchStr), strlen(pattern));
if (res == -1) {
 printf("Search pattern is not available\n");
} else
{ printf("Search pattern available at the location %d\n", res); }
 return 0;
 }
```

Output:

```
Enter Search String:God is great
 Enter Pattern String:Great
 Search pattern available at the location 5
```

Test Cases:

1. Check for pattern in first position
2. Check for pattern in last position
3. Check search unsuccessful.

5. Sort a given set of elements using Merge sort method and determine the time required to sort the elements.

```
#include <stdio.h>
#include<conio.h>
void mergesort(int arr[], int l, int h);
void main(void)
{
int array[100],n,i = 0;
clrscr();
printf("\t\t\tMerge Sort\n\n\n\n");
printf("Enter the number of elements to be sorted: ");
scanf("%d",&n);
printf("\nEnter the elements to be sorted: \n");
for(i = 0 ; i < n ; i++ )
{
printf("\tArray[%d] = ",i);
scanf("%d",&array[i]);
}
printf("\nBefore Mergesort:");
for(i = 0; i < n; i++)
{
printf("%4d", array[i]);
}
printf("\n");
mergesort(array, 0, n - 1);
printf("\nAfter Mergesort:");
for(i = 0; i < n; i++)
```

```
{
printf("%4d", array[i]);
}
printf("\n");
getch();
}
void mergesort(int arr[], int l, int h)
{
int i = 0;
int length = h - l + 1;
int pivot  = 0;
int merge1 = 0;
int merge2 = 0;
int temp[100];
if(l == h)
return;
pivot  = (l + h) / 2;
mergesort(arr, l, pivot);
mergesort(arr, pivot + 1, h);
for(i = 0; i < length; i++)
{
temp[i] = arr[l + i];
}
merge1 = 0;
merge2 = pivot - l + 1;
for(i = 0; i < length; i++)
{
if(merge2 <= h - l)
{
if(merge1 <= pivot - l)
{
if(temp[merge1] > temp[merge2])
{
arr[i + l] = temp[merge2++];
}
else
{
arr[i + l] = temp[merge1++];
}
}
else
{
arr[i + l] = temp[merge2++];
}
}
else
{
arr[i + l] = temp[merge1++];
}
}
```

}

Output:

Test Cases:

1. Check for array of size 5
2. Check for array of size 20
3. Check for array sorted in descending order

6. Sort a given set of elements using Quick sort method and determine the time required sort

int array_to_sort[SIZE];

8. Sort a given set of elements using Quick sort method and determine the time required sort the elements.

```
#include<stdio.h>
#include<conio.h>
void quicksort(int [10],int,int);
void main()
{
clrscr();
int a[20],n,i;
printf("Enter size of the array:\n");
scanf("%d",&n);
printf("Enter %d elements:\n",n);
for(i=0;i<n;i++)
{
scanf("%d",&a[i]);
}
quicksort(a,0,n-1);
printf("Sorted elements:\n ");
for(i=0;i<n;i++)
{
printf("\t%d",a[i]);
}
getch();
}
void quicksort(int a[10],int first,int last)
{
```

```
int pivot,j,temp,i;
if(first<last)
{
pivot=first;
i=first;
j=last;
while(i<j)
{
while(a[i]<=a[pivot]&&i<last)
i++;
while(a[j]>a[pivot])
j--;
if(i<j)
{
temp=a[i];
a[i]=a[j];
a[j]=temp;
}
}
temp=a[pivot];
a[pivot]=a[j];
a[j]=temp;
quicksort(a,first,j-1);
quicksort(a,j+1,last);
}
```

Output:

```
Enter the elements
10 9 4 6
Unsorted array 10 9 4 6
Sorted array 4 6 9 10
```

Test Cases:

1. Check for array of size 5
2. Check for array of size 20
3. Check for array sorted in descending order

7. Implement Recursive Binary search and determine the time required to search an element.

```
#include<stdio.h>
int main(){
 int a[10],i,n,m,c,l,u;
 printf("Enter the size of an array: ");
 scanf("%d",&n);
printf("Enter the elements of the array: " );
for(i=0;i<n;i++){
 scanf("%d",&a[i]);}
 printf("Enter the number to be search: ");
scanf("%d",&m);
```

```
l=0,u=n-1;
c=binary(a,n,m,l,u);
 if(c==0)
printf("Number is not found.");
else
printf("Number is found.");
return 0;
}
int binary(int a[],int n,int m,int l,int u){
int mid,c=0;
if(mid=(l+u)/2;
if(m==a[mid]){
c=1;
}
else if(m<a[mid])
{return binary(a,n,m,l,mid-1);
}
else
return binary(a,n,m,mid+1,u);
}
else
return c;
}
```

Output:

Enter the size of an array: 5
Enter the elements of the array: 8 9 10 11 12
Enter the number to be search: 8
Number is found.

Test Cases:

1.Check for number in first position
2. Check for number in last position
3.Check search unsuccessful.

8. Implement Strassen's matrix multiplication and determine the time required.

```
int main(){
 int a[2][2],b[2][2],c[2][2],i,j;
 int m1,m2,m3,m4,m5,m6,m7;
 printf("Enter the 4 elements of first matrix: ");
 for(i=0;i<2;i++)
for(j=0;j<2;j++)
scanf("%d",&a[i][j]);
printf("Enter the 4 elements of second matrix: ");
 for(i=0;i<2;i++)
for(j=0;j<2;j++)
scanf("%d",&b[i][j]);
 printf("\nThe first matrix is\n");
for(i=0;i<2;i++){
 printf("\n");
```

```
for(j=0;j<2;j++)
 printf("%d\t",a[i][j]);
}
 printf("\nThe second matrix is\n");
 for(i=0;i<2;i++){
 printf("\n");
for(j=0;j<2;j++)
printf("%d\t",b[i][j]);
}
 m1= (a[0][0] + a[1][1])*(b[0][0]+b[1][1]);
 m2= (a[1][0]+a[1][1])*b[0][0];
 m3= a[0][0]*(b[0][1]-b[1][1]);
 m4= a[1][1]*(b[1][0]-b[0][0]);
m5= (a[0][0]+a[0][1])*b[1][1];
 m6= (a[1][0]-a[0][0])*(b[0][0]+b[0][1]);
 m7= (a[0][1]-a[1][1])*(b[1][0]+b[1][1]);
 c[0][0]=m1+m4-m5+m7;
c[0][1]=m3+m5;
c[1][0]=m2+m4;
c[1][1]=m1-m2+m3+m6;
 printf("\nAfter multiplication using \n");
for(i=0;i<2;i++){
 printf("\n");
 for(j=0;j<2;j++)
 printf("%d\t",c[i][j]);
 }
 return 0;
}
```

Output:

```
Enter the 4 elements of first matrix: 1
2
3
4
Enter the 4 elements of second matrix: 5
6
7
8
The first matrix is
1 2
3 4
The second matrix is
5 6
7 8
After multiplication using
19 22
43 50
```

Test Cases:

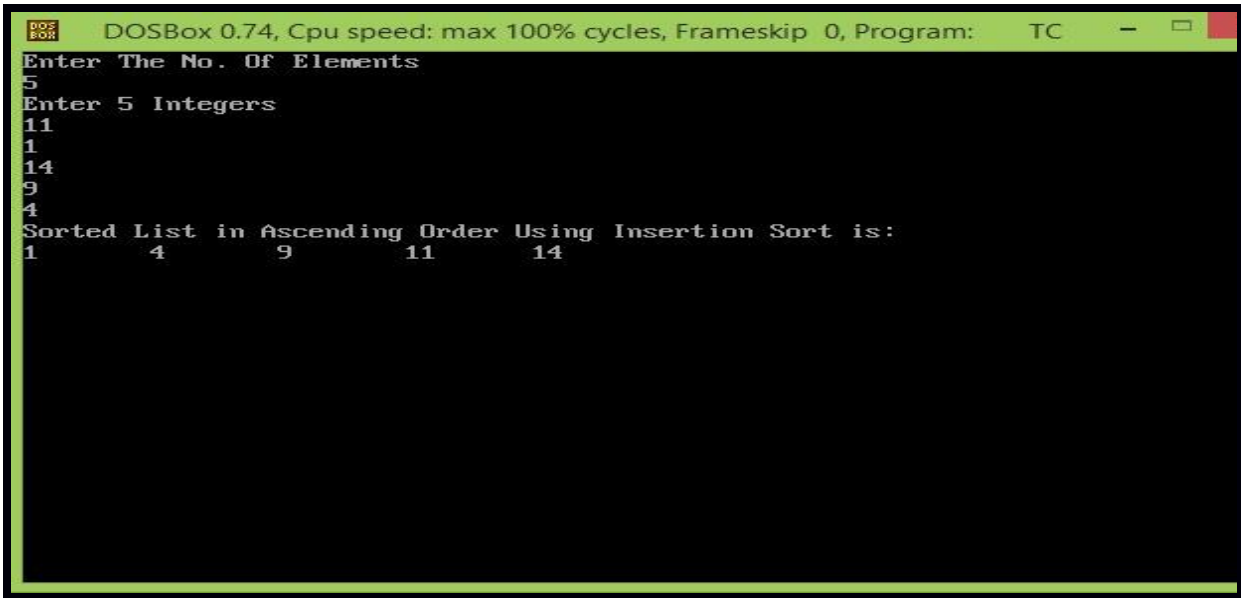
1. Check for valid matrix size
2.Ckeck for invalid matrix size

UNIT - III

1. Sort a given set of elements using the Insertion sort method and determine the time required to sort the elements.

```
void main()
{
clrscr();
int i,t,a[100],d,n;
printf("Enter The No. Of Elements\n");
scanf("%d",&n);
printf("Enter %d Integers\n",n);
for(i=0;i<n;i++)
{
scanf("%d",&a[i]);
}
for(i=1;i<=n-1;i++)
{
d=i;
while(d>0&&a[d]<a[d-1])
{
t=a[d];
a[d]=a[d-1];
a[d-1]=t;
d--;
}
}
printf("Sorted List in Ascending Order is:\n");
for(i=0;i<=n-1;i++)
{
printf("%d\t",a[i]);
}
getch();
}
```

Output:



Test Cases:

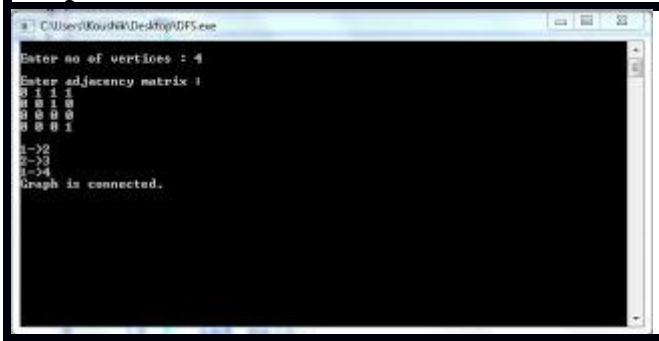
1. Check for array size 5
2. Check for array size 25

2. Check whether a given graph is connected or not using DFS method and determine the time required.

```
int a[20][20],reach[20],n;
void dfs(int v){
int i;
reach[v]=1;
for(i=1;i<=n;i++)
if(a[v][i]&&!reach[i]){
 printf("\n%d->%d",v,i);
 dfs(i);
 }
}
int main(){
 int i,j,count=0;
 printf("\nEnter no of vertices : ");
 scanf("%d",&n);
 for(i=1;i<=n;i++)
 for(j=1;j<=n;j++){
 reach[i]=0;
 a[i][j]=0;
 }
 printf("\nEnter adjacency matrix : \n");
 for(i=1;i<=n;i++)
 for(j=1;j<=n;j++)
 scanf("%d",&a[i][j]);
 dfs(1);
```

```
 for(i=1;i<=n;i++)
if(reach[i])
count++;
 if(count==n)
printf("\nGraph is connected.");
 else
printf("\nGraph is disconnected.");
 getch();
 return(0);
 }
```

Output:



Test Cases

1.Check for connected graph

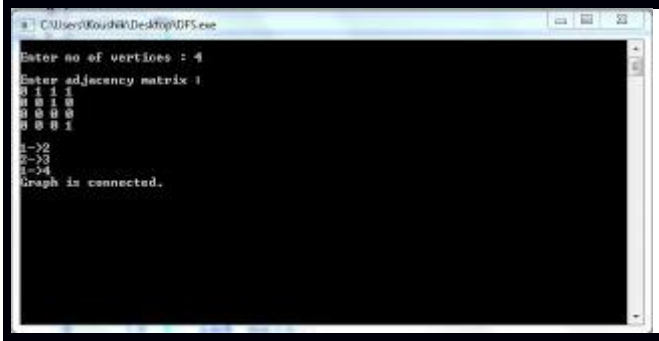
2.Check for disconnected graph

3. Print all the nodes reachable from a given starting node in a digraph using BFS method and determine the time required.

```
int a[20][20],q[20],visited[20],n,i,j,f=0,r=-1;
void bfs(int v)
{
 for(i=1;i<=n;i++)
  if(a[v][i] && !visited[i])
   q[++r]=i;
 if(f<=r)
 {
  visited[q[f]]=1;
  bfs(q[f++]);
 }
}
void main()
{
 int v;
 clrscr();
 printf("\n Enter the number of vertices:");
 scanf("%d",&n);
 for(i=1;i<=n;i++)
 {
  q[i]=0;
  visited[i]=0;
```

```
}
printf("\n Enter graph data in matrix form:\n");
for(i=1;i<=n;i++)
 for(j=1;j<=n;j++)
  scanf("%d",&a[i][j]);
printf("\n Enter the starting vertex:");
scanf("%d",&v);
bfs(v);
printf("\n The node which are reachable are:\n");
for(i=1;i<=n;i++)
 if(visited[i])
  printf("%d\t",i);
 else
  printf("\n Bfs is not possible");
getch();
}
```

Output:



Test Cases

1.Check for connected graph

2.Check for disconnected graph

4. Find the Topological sequence of vertices for the given graph and determine the time required

```
void main()
{
int n, count =0, am[10][10], indeg[10], flag[10], i, j, k;
clrscr();
printf("Enter number of vertices:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
indeg[i]=0;
flag[i]=0;
printf("\nEnter adjacency matrix:\n");
for(i=0;i<n;i++)
for(j=0;j<n;j++)
scanf("%d",&am[i][j]);
printf("\nMatrix is :\n");
for(i=0;i<n;i++)
```

```
{
for(j=0;j<n;j++)
printf("%d\t",am[i][j]);
printf("\n");
}
for(i=0;i<n;i++)
for(j=0;j<n;j++)
indeg[i] += am[j][i];
printf("\nThe topological ordering is:\n");
while(count<n)
{
for(k=0;k<n;k++)
if((indeg[k]==0) && (flag[k]==0))
{
printf("%d\n",k);
flag[k]=1;
count++;
for(i=0;i<n;i++)
if(am[k][i]==1)
indeg[i]--;
}
}
getch();
}
```

Output:

```
Enter number of vertices: 6
Enter adjacency matrix:
0 1 1 0 0 0
0 0 0 1 0 0
0 0 0 0 1 0
0 0 1 0 0 1
0 0 0 0 0 0
0 0 0 0 1 0
The topological ordering is 0 1 3 2 5 4
```

Test Cases:

1.Check for connected graph
2.Check for disconnected graph

5.Sort a given set of elements using the Heap sort method and determine the time required to sort the elements

```
void heapsort(int[], int);
void heapify(int[], int);
void adjust(int[], int);
int main()
{
int array[50],n,i;
clrscr();
printf("Enter the no. of elements to be sorted:\n ");
```

```
scanf("%d",&n);
printf("Enter %d elements: \n",n);
for(i=0 ; i<n ; i++)
{
scanf("%d",&array[i]);
}
heapsort(array,n);
printf("Sorted list in ascending order using heap sort is:\n");
for(i = 0; i < n; i++)
{
printf("%d\t", array[i]);
}
printf("\n");
getch();
return 0;
}
void heapsort(int array[], int n)
{
int i,t;
heapify(array,n);
for(i=n-1 ; i>0 ; i--)
{
t = array[0];
array[0] = array[i];
array[i] = t;
adjust(array,i);
}
}
void heapify(int array[], int n)
{
int item,i,j,k;
for(k=1 ; k<n ; k++)
{
item = array[k];
i = k;
j = (i-1)/2;
while( (i>0) && (item>array[j]) )
{
array[i] = array[j];
i = j;
j = (i-1)/2;
}
array[i] = item;
}
}
void adjust(int array[], int n)
{
int item,i,j;
j = 0;
item = array[j];
```

```
i = 2*j+1;
while(i<=n-1)
{
if(i+1 <= n-1)
if(array[i] < array[i+1])
i++;
if(item < array[i])
{
array[j] = array[i];
j = i;
i = 2*j+1;
}
else
break;
}
array[j] = item;
}
```

Output:

Test Cases:

1. Check for array of size 5
2. Check for array of size 20
3. Check for array sorted in descending order

UNIT - IV

1. Sort a given set of elements using the Sorting by counting method and determine the time required to sort the elements.

```
void Counting_sort(int A[], int k, int n)
{
int i, j;
int B[15], C[100];
for(i = 0; i <= k; i++)
C[i] = 0;
for(j =1; j <= n; j++)
C[A[j]] = C[A[j]] + 1;
```

```
for(i = 1; i <= k; i++)
C[i] = C[i] + C[i-1];
for(j = n; j >= 1; j--)
{
B[C[A[j]]] = A[j];
C[A[j]] = C[A[j]] - 1;
}
printf("\nThe Sorted array is :\n");
for(i = 1; i <= n; i++)
printf("\t%d",B[i]);
}
void main()
{
clrscr();
int n,i,k = 0, A[15];
printf("\t\tCOUNTING SORT ALGORITHM\n\n\n\n");
printf("Enter the number of input : ");
scanf("%d",&n);
printf("\n\nEnter the elements to be sorted :\n");
for ( i = 1; i <= n; i++)
{
scanf("%d",&A[i]);
if(A[i] > k)
{
k = A[i];
}
}
Counting_sort(A, k, n);
getch();
}
```

Output:

```
                COUNTING SORT ALGORITHM


Enter the number of input : 6

Enter the elements to be sorted :
11
1
14
9
4
18

The Sorted array is :
        1       4       9       11      14      18
```

Test Cases:

1. Check for array of size 5

2. Check for array of size 20

3. Check for array sorted in descending order

2. Sort a given set of elements using the Distribution counting method and determine the time required to sort the elements.

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

int i,j,n,m,a[20],s[20],count[20];
void dist(int a[],int n);

void main()
{
clrscr();
printf("Enter the number of elements:\n");
scanf("%d",&n);
printf("Enter the elements in an array:\n");
for(i=0;i<n;i++)
scanf("%d",&a[i]);
printf("Unsorted array is:\n");
for(i=0;i<n;i++)
printf("%d\t",a[i]);
dist(a,n);
getch();
}

void dist(int a[],int n)
{
m=n;

for(j=0;j<=m-1;j++)
count[j]=0;

for(i=1;i<=n;i++)
count[a[i]]= count[a[i]] + 1;

for(j=1;j<=m-1;j++)
count[j]= count[j-1] + count[j];

for(i=n;i>=0;i--)
{
s[count[a[i]]-1]= a[i];
count[a[i]]= count[a[i]] - 1;
}

printf("\nSorted array is:\n");
for(i=0;i<n;i++)
printf("%d\t",s[i]);
```

```
}
```

Output:

```
Enter the number of element
6
Enter the elements
11 13 23 23 11 11
Unsorted array is 11 13 23 23 11 11
Sorted array is 11 11 11 13 23 23
```

Test Cases:

1. Check for array of size 5
2. Check for array of size 20
3. Check for array sorted in descending order

3. Implement Horspool's algorithm for String Matching and determine the time required.

```
#define MAX 500
int t[MAX];
void shifttable(char p[])
{
 int i,j,m;
 m=strlen(p);
 for(i=0;i<MAX;i++)
  t[i]=m;
 for(j=0;j<m-1;j++)
  t[p[j]]=m-1-j;
}
int horspool(char src[],char p[])
{
 int i,j,k,m,n;
 n=strlen(src);
 m=strlen(p);
 printf("\nLength of text=%d",n);
 printf("\n Length of pattern=%d",m);
 i=m-1;
 while(i<n)
 {
  k=0;
  while((k<m)&&(p[m-1-k]==src[i-k]))
   k++;
  if(k==m)
   return(i-m+1);
else
 i+=t[src[i]];
 }
 return -1;
}
void main()
{
```

```
char src[100],p[100];
int pos;
clrscr();
printf("Enter the text in which pattern is to be searched:\n");
gets(src);
printf("Enter the pattern to be searched:\n");
gets(p);
shifttable(p);
pos=horspool(src,p);
if(pos>=0)
 printf("\n The desired pattern was found starting from position %d",pos+1);
else
 printf("\n The pattern was not found in the given text\n");
getch();
}
```

Output:

```
 Enter Search String:God is great
 Enter Pattern String:Great
 Search pattern available at the location 5
```

Test Cases:

1. Check for pattern in first position
2. Check for pattern in last position
3. Check search unsuccessful.

4. Find the Binomial Coefficient using Dynamic Programming and determine the time required.

```
int n,m;                        /* computer n choose m */
{
        int i,j;                /* counters */
        long bc[MAXN][MAXN];   /* table of binomial coefficient values */

        for (i=0; i<=n; i++) bc[i][0] = 1;
for (j=0; j<=n; j++) bc[j][j] = 1;


        for (i=1; i<=n; i++)
                for (j=1; j<i; j++)
                        bc[i][j] = bc[i-1][j-1] + bc[i-1][j];

        return( bc[n][m] );
}

main()
{
        int a, b;
```

```
        long binomial_coefficient();

        while (1) {
                scanf("%d %d",&a,&b);
                printf("%d\n",binomial_coefficient(a,b));
        }
}
```

Output:

Enter 4 2

Binomial coefficient is 6

Test Cases:

1.Check for 8,4

2. Check for 4, 6

5. Compute the transitive closure of a given directed graph using Warshall's algorithm and determine the time required.

```
int max(int,int);
 void warshal(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++)
            p[i][j]=max(p[i][j],p[i][k]&&p[k][j]);
 }
 int max(int a,int b) {
;
        if(a>b)
         return(a); else        return(b);
 }
 void main() {
        int n,e,u,v,i,j;

        clrscr();
        printf("\n Enter the number of vertices:");
        scanf("%d",&n);
        printf("\n Enter the number of edges:");
        scanf("%d",&e);
        for (i=1;i<=e;i++) {
                printf("\n Enter the end vertices of edge %d:",i);
                scanf("%d%d",&u,&v);
                p[u][v]=1;
        }
        printf("\n Matrix of input data: \n");
        for (i=1;i<=n;i++) {
                for (j=1;j<=n;j++)
                  printf("%d\t",p[i][j]);
                printf("\n");
        }
```

```
        warshal(p,n);
        printf("\n Transitive closure: \n");
        for (i=1;i<=n;i++) {
                for (j=1;j<=n;j++)
                  printf("%d\t",p[i][j]);
                printf("\n");
        }
```

Output:

```
Enter the number of vertices 3
Enter the edges 2
Enter the end vertices of edges
1 2
2 3
Transitive closure 0 1 1
                   0 0 1
                   0 0 0
```

Test Cases:

1.Check for 4 vertices
2. Check for 5 vertices.

6. Implement Floyd's algorithm for the All-Pairs-Shortest-Paths problem and determine the time required

```
int min(int,int);
void floyds(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(i==j)
     p[i][j]=0;
    else
     p[i][j]=min(p[i][j],p[i][k]+p[k][j]);
}
int min(int a,int b)
{
 if(a<b)
  return(a);
 else
  return(b);
}
void main()
{
 int p[10][10],w,n,e,u,v,i,j;;
 clrscr();
 printf("\n Enter the number of vertices:");
 scanf("%d",&n);
 printf("\n Enter the number of edges:\n");
```

```
scanf("%d",&e);
for(i=1;i<=n;i++)
{
 for(j=1;j<=n;j++)
  p[i][j]=999;
}
for(i=1;i<=e;i++)
{
 printf("\n Enter the end vertices of edge%d with its weight \n",i);
 scanf("%d%d%d",&u,&v,&w);
 p[u][v]=w;
}
printf("\n Matrix of input data:\n");
for(i=1;i<=n;i++)
{
 for(j=1;j<=n;j++)
  printf("%d \t",p[i][j]);
 printf("\n");
}
floyds(p,n);
printf("\n Transitive closure:\n");
for(i=1;i<=n;i++)
{
 for(j=1;j<=n;j++)
  printf("%d \t",p[i][j]);
 printf("\n");
}
printf("\n The shortest paths are:\n");
for(i=1;i<=n;i++)
 for(j=1;j<=n;j++)
 {
  if(i!=j)
   printf("\n <%d,%d>=%d",i,j,p[i][j]);
 }
getch();
}
```

Output:

Enter the number of vertices 3
Enter the number of edges 3
Enter the end vertices with weight
1 2 3
1 3 6
2 3 1
Shortest path
999 3 4
999 999 1
999 999 999

Test Cases:

1.Check for 4 vertices
2. Check for 5 vertices

7. Implement 0/1 Knapsack problem using dynamic programming and determine the time required

```
void knapsack(int n, float weight[], float profit[], float capacity)
{int i, j, u;
u = capacity;
for (i = 0; i < n; i++)
x[i] = 0.0;
for (i = 0; i < n; i++)
{
if (weight[i] > u)
break;
else
{
x[i] = 1.0;
tp = tp + profit[i];
u = u - weight[i];
}
}
if (i < n)
x[i] = u / weight[i];
tp = tp + (x[i] * profit[i]);
printf("\nThe result vector is:- ");
for (i = 0; i < n; i++)
printf("%f\t", x[i]);
printf("\nMaximum profit is:- %f", tp);
}
void main()
{
clrscr();
float weight[20], profit[20], capacity;
int num, i, j;
float ratio[20], temp;
printf("\nEnter the no. of objects:- ");
scanf("%d", &num);
printf("\nEnter the weights and profits of each object:- ");
for (i = 0; i < num; i++)
{
scanf("%f %f", &weight[i], &profit[i]);
}
printf("\nEnter the capacity of knapsack:- ");
scanf("%f", &capacity);
for (i = 0; i < num; i++)
{
ratio[i] = profit[i] / weight[i];
}
for (i = 0; i < num; i++)
{
for (j = i + 1; j < num; j++)
```

```
{
if (ratio[i] < ratio[j])
{
temp = ratio[j];
ratio[j] = ratio[i];
ratio[i] = temp;
temp=weight[j];
weight[j] = weight[i];
weight[i] = temp;
temp = profit[j];
profit[j] = profit[i];
profit[i] = temp;
}
}
}
knapsack(num, weight, profit, capacity);
getch();
}
```

Output:

```
Enter the no. of objects:-
7

Enter the weights and profits of each object:- 2
10
3
5
5
15
7
7
1
6
4
18
1
3

Enter the capacity of knapsack:- 15

The result vector is:- 1.000000 1.000000        1.000000        1.000000
1.000000        0.666667        0.000000
Maximum profit is:- 55.333332_
```

Test Cases:

Check for 4 objects

Check for 5 objects.

UNIT - V

1. Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm and determine the time required.

```
#define infinity 9999
#define MAX 20

int G[MAX][MAX],spanning[MAX][MAX],n;

int prims();

int main()
{
   int i,j,total_cost;
   printf("Enter no. of vertices:");
   scanf("%d",&n);

   printf("\nEnter the adjacency matrix:\n");

   for(i=0;i<n;i++)
      for(j=0;j<n;j++)
         scanf("%d",&G[i][j]);

   total_cost=prims();
   printf("\nspanning tree matrix:\n");

   for(i=0;i<n;i++)
   {
      printf("\n");
      for(j=0;j<n;j++)
         printf("%d\t",spanning[i][j]);
   }

   printf("\n\nTotal cost of spanning tree=%d",total_cost);
   return 0;
}

int prims()
{
   int cost[MAX][MAX];
   int u,v,min_distance,distance[MAX],from[MAX];
   int visited[MAX],no_of_edges,i,min_cost,j;

   //create cost[][] matrix,spanning[][]
   for(i=0;i<n;i++)
      for(j=0;j<n;j++)
      {
         if(G[i][j]==0)
            cost[i][j]=infinity;
         else
            cost[i][j]=G[i][j];
            spanning[i][j]=0;
      }
```

```
  //initialise visited[],distance[] and from[]
  distance[0]=0;
  visited[0]=1;

  for(i=1;i<n;i++)
  {
     distance[i]=cost[0][i];
     from[i]=0;
     visited[i]=0;
  }

  min_cost=0;       //cost of spanning tree
  no_of_edges=n-1;       //no. of edges to be added

  while(no_of_edges>0)
  {
     //find the vertex at minimum distance from the tree
     min_distance=infinity;
     for(i=1;i<n;i++)
        if(visited[i]==0&&distance[i]<min_distance)
        {
           v=i;
           min_distance=distance[i];
        }

     u=from[v];

     //insert the edge in spanning tree
     spanning[u][v]=distance[v];
     spanning[v][u]=distance[v];
     no_of_edges--;
     visited[v]=1;

     //updated the distance[] array
     for(i=1;i<n;i++)
        if(visited[i]==0&&cost[i][v]<distance[i])
        {
           distance[i]=cost[i][v];
           from[i]=v;
        }
min_cost=min_cost+cost[u][v];
  }

  return(min_cost);
}
```

Output

Enter no. of vertices:6

*E*nter the adjacency matrix:
0 3 1 6 0 0
3 0 5 0 3 0
1 5 0 5 6 4
6 0 5 0 0 2
0 3 6 0 0 6
0 0 4 2 6 0

spanning tree matrix:

0 3 1 0 0 0
3 0 0 0 3 0
1 0 0 0 0 4
0 0 0 0 0 2
0 3 0 0 0 0
0 0 4 2 0 0

Total cost of spanning tree=13

Test Cases:

32. Check for 4 vertices.
33. Check for 5 vertices.

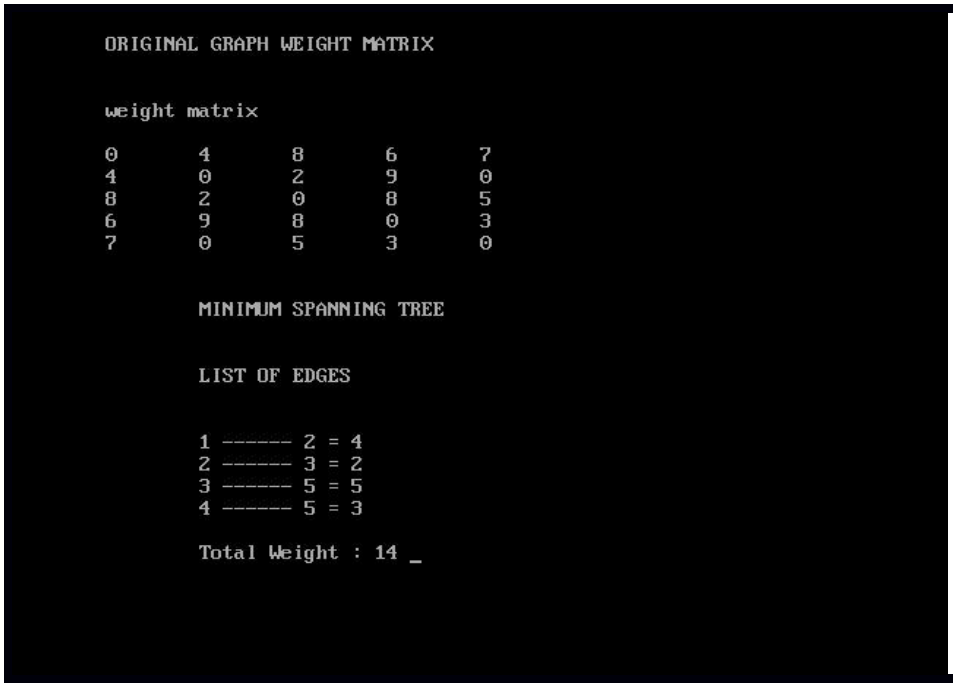
2. Find Minimum Cost Spanning Tree of a given undirected graph using kruskals algorithm and determine the time required.

```
#define INF 0
char vertex[10];
int wght[10][10];
int span_wght[10][10];
int source;
struct Sort
{
int v1,v2;
int weight;
}que[20];
int n,ed,f,r;
int cycle(int s,int d)
{
int j,k;
if(source==d)
return 1;
for(j=0;j<n;j++)
if(span_wght[d][j]!=INF && s!=j)
{
if(cycle(d,j))
return 1;
}
return 0;
}
```

```
void build_tree()
{
int i,j,w,k,count=0;
for(count=0;count<n;f++)
{
i=que[f].v1;
j=que[f].v2;
w=que[f].weight;
span_wght[i][j]=span_wght[j][i]=w;
source=i;
k=cycle(i,j);
if(k)
span_wght[i][j]=span_wght[j][i]=INF;
else
count++;
}
}
void swap(int *i,int *j)
{
int t;
t=*i;
*i=*j;
*j=t;
}
void main()
{
int i,j,k=0,temp;
int sum=0;
clrscr();
printf("\n\tEnter the No. of Nodes : ");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("\n\tEnter %d value : ",i+1);
fflush(stdin);
scanf("%c",&vertex[i]);
for(j=0;j<n;j++)
{
wght[i][j]=INF;
span_wght[i][j]=INF;
}
}
printf("\n\nGetting Weight\n");
for(i=0;i<n;i++)
for(j=i+1;j<n;j++)
{
printf("\nEnter 0 if path Doesn't exist between %c to %c : ",vertex[i],vertex[j]);
scanf("%d",&ed);
if(ed>=1)
{
```

```
wght[i][j]=wght[j][i]=ed;
que[r].v1=i;
que[r].v2=j;
que[r].weight=wght[i][j];
if(r)
{
for(k=0;k<r;k++)
if(que[k].weight>que[r].weight)
{
swap(&que[k].weight,&que[r].weight);
swap(&que[k].v1,&que[r].v1);
swap(&que[k].v2,&que[r].v2);
}
}
r++;
}
}
clrscr();
printf("\n\tORIGINAL GRAPH WEIGHT MATRIX\n\n");
printf("\n\tweight matrix\n\n\t");
for(i=0;i<n;i++,printf("\n\t"))
for(j=0;j<n;j++,printf("\t"))
printf("%d",wght[i][j]);
build_tree();
printf("\n\n\t\tMINIMUM SPANNING TREE\n\n");
printf("\n\t\tLIST OF EDGES\n\n");
for(i=0;i<n;i++)
for(j=i+1;j<n;j++)
if(span_wght[i][j]!=INF)
{
printf("\n\t\t%c ------ %c = %d ",vertex[i],vertex[j],span_wght[i][j]);
sum+=span_wght[i][j];
}
printf("\n\n\t\tTotal Weight : %d ",sum);
getch();
}
```

Output:



Test Cases:

1. Check for 6 vertices.
2. Check for 5 vertices.

3.From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm and determine the time required

```
void dijkstra(int n, int v, int cost[10][10],int dist[10])
{
int count, u, i, w, visited[10], min;
for(i=0;i<n;i++)
{
visited[i]=0;
dist[i]=cost[v][i];
}
visited[v]=1;
dist[v]=1;
count=2;
while(count<=n)
{
min=999;
for(w=0;w<n;w++)
if((dist[w]<min) && (visited[w]!=1))
{
min=dist[w];
u=w;
}
visited[u]=1;
count++;
for(w=0;w<n;w++)
```

```
if((dist[u]+cost[u][w]<dist[w]) && (visited[w]!=1))
dist[w]=dist[u]+cost[u][w];
}
}
void main()
{
int n, v, cost[10][10], dist[10], i, j;
clrscr();
printf("Enter number of vertices:");
scanf("%d",&n);
printf("\nEnter cost matrix (for infinity, enter 999):\n");
for(i=0;i<n;i++)
for(j=0;j<n;j++)
scanf("%d",&cost[i][j]);
printf("\nEnter source vertex:");
scanf("%d",&v);
dijkstra(n,v,cost,dist);
printf("\nShortest path from \n");
for(i=0;i<n;i++)
if(i!=v)
printf("\n%d -> %d = %d", v, i, dist[i]);
getch();
}
```

OUTPUT:

```
Enter number
of vertices:3
Enter cost matrix (for infinity, enter 999):
0  2  999
999 0  2
1 4  0
Enter source vertex:0
0 -> 1 = 2
0 -> 2 = 4
```

4. Implement n queen's problem using back tracking method and determine the time required.

```
char a[10][10];
int n;
void printmatrix()
{
int i, j;
printf("\n");
for (i = 0; i < n; i++)
{
for (j = 0; j < n; j++)
printf("%c\t", a[i][j]);
```

```
printf("\n\n");
}
}
int getmarkedcol(int row)
{
int i;
for (i = 0; i < n; i++)
if (a[row][i] == 'Q')
{
return (i);
break;
}
}
int feasible(int row, int col)
{
int i, tcol;
for (i = 0; i < n; i++)
{
tcol = getmarkedcol(i);
if (col == tcol || abs(row - i) == abs(col - tcol))
return 0;
}
return 1;
}
void nqueen(int row)
{
int i, j;
if (row < n)
{
for (i = 0; i < n; i++)
{
if (feasible(row, i))
{
a[row][i] = 'Q';
nqueen(row + 1);
a[row][i] = '.';
}
}
}
else
{
printf("\nThe solution is:- ");
printmatrix();
}
}
void main()
{
clrscr();
int i, j;
printf("\nEnter the no. of queens- ");
```

```
scanf("%d", &n);
for (i = 0; i < n; i++)
for (j = 0; j < n; j++)
a[i][j] = '.';
nqueen(0);
getch();
}
```

Output:

Test cases:

1.check for 2 queen

2.check for 5 queen

5. 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. For example, if S= {1, 2, 5, 6, 8} and d = 9 .If there are two solutions{1,2,6}and{1,8}.A suitable message is to be displayed if the given problem instance doesn't have a solution and determine the time required.

```
void main()
{
   int s[20],d,sum=0,n,x[20],top=0,i,tot=0;
   clrscr();
   printf("\nEnter the number of values ");

scanf("%d",&n);
   printf("\nEnter the values in ascending order ");
   for(i=0;i<n;i++)
   {
      scanf("%d",&s[i]);
   }
   printf("\nEnter the sum ");
   scanf("%d",&d);
   x[top]=-1;
   printf("\nThe solution to the subset problem is ");
   while(top>=0)
   {
      x[top]=x[top]+1;
```

```
        sum=sum+s[x[top]];
        if(sum==d)
        {
            printf("\n");
            tot=tot+1;
            for(i=0;i<=top;i++)
            {
                printf("%d ",s[x[i]]);
            }
            sum=sum-s[x[top]];
        }
        else if(sum>d||top>=n)
        {
            sum=sum-s[x[top]];
            if(top>=1)
            {
                sum=sum-s[x[top-1]];
            }
            top=top-1;
        }
        else
        {
            top=top+1;
            x[top]=x[top-1];
        }
    }
    if(tot==0)
    {
        printf("not possible ");
    }
    getch();
}
```