MANAKULA VINAYAGAR INSTITUTE OF TECHNOLOGY

KALITHEERTHALKUPPAM

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ACADEMIC YEAR 2014-2015[EVEN SEM]

DESIGN AND ANALYSIS OF ALGORITHMS LABORATORY MANUAL

CSP42

Subject Code	Subject Name	Lectures (Periods)	Tutorial (Periods)	Practical (Periods)
CS P42	DESIGN AND ANALYSIS OF ALGORITHMS LAB	-	-	3

LIST OF EXPERIMENTS

1. Implementation of binary search using Divide-and-Conquer technique.
2. Implementation of merge sort algorithms using Divide-and-Conquer technique.
3. Implementation of quick sort algorithms using Divide-and-Conquer technique.
4. Implementation of Knapsack using Greedy technique.
5. Implementation of Single-Source Shortest Paths algorithms using Greedy technique.
6. Implementation of Multi-Stage Graphs using Dynamic Programming technique.
7. Implementation of 0/1 Knapsack using Dynamic Programming technique.
8. Implementation of All Pairs Shortest Paths using Dynamic Programming technique.
9. Implementation of Traveling Salesman algorithms using Dynamic Programming technique.
10. Implementation of Pre-order, In-order, Post-order traversals using DFS traversal techniques.
11. Implementation of Pre-order, In-order, Post-order traversals using BFS traversal techniques.
12. Implementation of 8 Queens with the design of Backtracking.
13. Implementation of sum of subsets with the design of Backtracking.
14. Implementation of 0/1 Knapsack problems with Branch-and-Bound technique.
15. Implementation of Traveling Salesman problems with Branch-and-Bound technique.

EX.NO	LIST OF EXPERIMENTS
DIVIDE & CONQUER TECHNIQUE	
1	Binary Search
2	Merge sort
3	Quick sort
GREEDY TECHNIQUE	
4	Knapsack Problem
5	Single Source Shortest Path
DYNAMIC PROGRAMMING	
6	0/1 Knapsack problems
7	All Pairs Shortest Paths
8	Travelling Salesman algorithms
TRAVERSAL TECHNIQUE	
9	Pre-order, In-order, Post-order traversals using DFS traversal techniques.
10	Pre-order, In-order, Post-order traversals using BFS traversal techniques.
BACKTRACKING	
11	N-Queens Problem
12	Sum of Subset Problem
BRANCH AND BOUND TECHNIQUE	
13	0/1 Knapsack problems
14	Travelling Salesman problems

BINARY SEARCH

AIM:

To write a C++ program to implement Binary search.

ALGORITHM:

Step 1: Start the program.
Step 2: Declare the variables and an array of elements.
Step 3: Read the value of number of elements to be stored in the array.
Step 4: Read the values of the elements of the array.
Step 5: Divide the array of elements into two which gives a middle element.
Step 6: Get the element to be searched in the array.
Step 7: Compare the element with the middle element. If the element to be searched is the middle element stop searching and print the position.
Step 8: If the element is lesser than the middle element the searching proceeds with the first half of the array.
Step 9: If the element is greater than the middle element the searching proceeds with the second half of the array.
Step 10: If the element is not present in the array print" Unsuccessful search".
Step 11: If the element is present in the array, print the position of the searched element.
Step 12: Stop the program.

SOURCE CODE:

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
class binary
{
		int i,mid,low,high,n,x,a[20];
	public:
		void get();
		int search();
		void complex();
};

void main()
{
	int index;
	binary b;
	clrscr();
	b.get();
	index = b.search();
	if(index != -1)
	{
```

```
			cout<<"\nElement found in "<< index+1 <<" position";
		}
		else
		{
			cout<<"\nElement not found ";
		}
		b.complex();
		getch();
	}


	//GETS THE INPUT
	void binary::get()
	{
		cout<<"\nEnter the number of elements :";
		cin>>n;
		cout<<"\nEnter the array elements:\n";
		for(i=0;i<n;i++)
		{
			cin>>a[i];
		}
		cout<<"\nEnter search elements:\n";
		cin>>x;
	}

	//PERFORMS SEARCH
	int binary::search()
	{
		low=0;high=n-1;
		while (low<=high)
		{
			mid=(low+high)/2;
			if(x==a[mid])
			{
				return mid;
			}
			else if(x>a[mid])
			{
				low=mid+1;
			}
			else if(x<a[mid])
			{
				high=mid-1;
			}
		}
		return -1;
	}

	//DISPLAYS THE TIME & SPACE COMPLEXITY
	void binary::complex()
```

```
{
        cout<<"\n\nTime Complexity is O("<<log(n)<<")"<<"\n";
        cout<<"\nSpace Complexity is O("<<n<<")";
}
```

Sample input and output:

Enter the limit 6

Enter the numbers
23
45
34
36
40
67
Enter the number to be found 36
Element36 is found in the position 4

Result

Thus the program for Binary search has been executed successfully

MERGE SORT

AIM:

To create a C++ program to sort the unsorted array using Merge sort algorithm.

ALGORITHM:

Step 1: Start the program.
Step 2: Declare the variables and two arrays of elements.
Step 3: Read the value of number of elements to be stored in the two arrays.
Step 4: Read the values of the elements of two arrays.

Step 5: Compare the first element with the last element, if the first is lesser than the last element compute the middle value sort and split the array as first, middle, middle+1, and the last.

Step 6: Compare two arrays, If the 0th element of the first array (a[i]) is smaller than the 0th element of the second (b[i]) array then put the 0th element of the first array in the third array to be merged.

Step 7: Then compare the 1st element of a[i] with the 0th element of b[i] and repeat the steps.

Step 8: Print the sorted third array.
Step 9: Stop the program.

SOURCE CODE:

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
class mergesort
{
		int a[10];
	public:
		void getdata(int n)
		{
			int i;
			for(i=1;i<=n;i++)
			{
				cin>>a[i];
			}
		}
	void partition(int low,int high)
	{
		int mid;
		if(low<high)
		{
			mid=(low+high)/2;
			partition(low,mid);
			partition(mid+1,high);
			msort(low,mid,high);
```

```
        }
}
void msort(int low,int mid,int high)
{
        int b[10];
        int h,i,k,j;

        h=low;
        j=mid+1;

        i=low;
        // compare two partition
        while(h<=mid&&j<=high)
        {
                if(a[h]<a[j])
                {
                        b[i]=a[h];
                        h=h+1;
                }
                else
                {
                        b[i]=a[j];
                        j=j+1;
                }
                i=i+1;
        }
        //append remaining element
        if(h<=mid)
        {
                for(k=h;k<=mid;k++)
                {
                        b[i]=a[k];
                        i=i+1;
                }
        }
        else if(j<=high)
        {
                for(k=j;k<=high;k++)
                {
                        b[i]=a[k];
                        i=i+1;
                }
        }
        // copy to a[]
        for(k=low;k<=high;k++)
        {
                a[k]=b[k];
        }
}
void display(int n)
```

```
	{
		int i;
		cout<<"\nThe sorted array is:\n";
		for(i=1;i<=n;i++)
		{
			cout<<a[i]<<"\t";
		}
	}

	//DISPLAYS THE TIME & SPACE COMPLEXITY
	void complex(int n)
	{
		cout<<"\n\nTime Complexity is O("<<n*log10(n)<<")"<<endl;
		cout<<"\nSpace Complexity is O("<<n<<")"<<endl;
	}

};

void main()
{
	int n;
	clrscr();
	cout<<"\n\t\t MERGE SORT ";
	mergesort ms;
	cout<<"\nEnter the number of elements:\n";
	cin>>n;
	cout<<"\nEnter the values \n";
	ms.getdata(n);
	ms.partition(1,n);
	ms.display(n);
	ms.complex(n);
	getch();
}
```

Sample Input and Output:

MERGE SORT

Enter the number of elements:
5

Enter the values
23
32
12
54
15

The sorted array is:
12 15 23 32 54

Time Complexity is O(3.49485)

Space Complexity is O(5)

Result:
Thus the program for merge sort has been executed successfully.

QUICK SORT

Aim:

To create a C++ program to sort the unsorted array using Quick sort algorithm.

Algorithm:

Step 1: Start the program.
Step 2: Declare the variables and an array of elements.
Step 3: Read the value of number of elements to be stored in the array.
Step 4: Read the values of the elements of the array.
Step 5: Select the first element and the last element called pivot from the array and compare with the other elements from the both sides of the array.
Step 6: The elements which are lesser than the pivot element is placed before the pivot element and greater placed after the pivot element.
Sterp7: Then the array is divided into two. Step 5 and Step 6 is repeated on both the halves of the array until the sorting is done completely.
Step 8: Print the sorted array.
Step 9: Stop the program.

SOURCE CODE:

```
// QUICK SORT

#include<iostream.h>
#include<conio.h>

int a[10],i,j,n,temp,pivot;

class quicksort
{
public:
        void get();
        void sort(int,int);
        void interchange(int[],int,int);
        int partition(int[],int,int);
        void put();
        void complex();
};

void main()
{
        quicksort r;
        clrscr();
        r.get();
        r.sort(0,n-1);
        r.put();
        r.complex();
        getch();
}
```

```
void quicksort::get()
{
	cout<<"\nEnter the number of elements:";
	cin>>n;
	cout<<"Enter the numbers:\n";
	for(i=0;i<n;i++)
	{
		cin>>a[i];
	}
}

void quicksort::sort(int left,int right)
{
	if(left<right)
	{
		j=partition(a,left,right);
		sort(left,j-1);
		sort(j+1,right);
	}
}

int quicksort::partition(int a[],int left,int right)
{
 if(left<right)
 {
	pivot=a[left];
	i=left;
	j=right+1;
	do
	{
		do
		{
			i++;
		}while(a[i]<pivot);

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

		if(i<j)
		{
			interchange(a,i,j);
		}
	}while(i<j);
	interchange(a,left,j);

	}
	return j;
```

```
}

void quicksort::interchange(int a[],int i,int j)
{
	temp=a[i];
	a[i]=a[j];
	a[j]=temp;
}

void quicksort::put()
{
	cout<<"Sorted elements are:\n";
	for(i=0;i<n;i++)
	{
		cout<<a[i]<<"\t";
	}
}

//DISPLAYS THE TIME & SPACE COMPLEXITY
void quicksort::complex()
{
	cout<<"\n\nTime Complexity is O("<<n*n<<")"<<endl;
	cout<<"\nSpace Complexity is O("<<n<<")"<<endl;
}
```

SAMPLE INPUT AND OUTPUT:

```
Enter the number of elements:5
Enter the numbers:
10
45
15
13
8
Sorted elements are:
8	10	13	15	45

Time Complexity is O(25)

Space Complexity is O(5)
```

Result:

Thus the program for quick sort has been executed successfully.

KNAPSACK PROBLEM

Aim:

To write a C++ program to implement Knapsack Problem.

Algorithm:

1. Start the program.
2. Create a class knapsack and declare variables n, i, j with float variable of w[10], p[10], value[10], capacity and temp.
3. In public declare the functions getdata, sort, knaps, display.
4. Get the number of elements, their weights and profit and calculate the ratio of p and w.
5. Sort them in descending order.
6. Check the weight with capacity of bag and insert it to bag and obtain the remaining capacity.
7. Display the weight, load, and profit along with the total profit.
8. Stop.

SOURCE CODE:

```
#include<iostream.h>
#include<conio.h>
class knapsack
{
		int n,i,j;
		float w[10],p[10],value[10];
		float capacity,temp,c;
	public:
		void getdata();
		void sort();
		void knaps();
		void display();
};
float pro=0.0,s[10]={0,0,0,0,0,0,0,0,0,0};
void main()
{
	clrscr();
	knapsack ks;
	ks.getdata();
	ks.sort();
	ks.knaps();
	ks.display();
	getch();
}
void knapsack::getdata()
{
	cout<<"\n\t KNAPSACK PROBLEM ";
	cout<<"\n Enter the number of elements: ";
	cin>>n;
	// Read N values
```

```
	for(i=0;i<n;i++)
	{
		cout<<"\n Enter the weight: ";
		cin>>w[i];
		cout<<"\n Enter the profit: ";
		cin>>p[i];
		value[i]=p[i]/w[i];
	}
	cout<<"\n\nEnter Capacity of the bag:";
	cin>>capacity;
}
void knapsack::sort()
{
	// Sort decending
	for(i=0;i<n;i++)
	{
		for(j=i+1;j<n;j++)
		{
			if(value[i]<value[j])
			{
				temp=w[i];
				w[i]=w[j];
				w[j]=temp;
				temp=p[i];
				p[i]=p[j];
				p[j]=temp;
				temp=value[i];
				value[i]=value[j];
				value[j]=temp;
			}
		}
	}

	cout<<"\n\nProfit by weight in decreasing order:";
	cout<<"\n\n\tWeight\tProfit\tRatio";
	cout<<"\n\n\t------\t------\t-----\n\n";
	for(i=0;i<n;i++)
	{
		cout<<"\t"<<w[i]<<"\t"<<p[i]<<"\t"<<value[i]<<"\n";
	}

}

void knapsack::knaps()
{

	for(i=0;i<n;i++)
	{
		if(w[i]<=capacity)
		{
```

```
                s[i]=1;
                capacity=capacity-w[i];
            }
            else
            {
                break;
            }
        }
        if(i!=n)
        {
            s[i]=capacity/w[i];
        }

}
void knapsack::display()
{
        cout<<"\n\tWeight \t Load \t  profit\n";
        for(i=0;i<n;i++)
        {
            cout<<"\t"<<w[i]<<"\t"<<s[i]<<"\t"<<p[i]*s[i]<<"\n";
            pro=pro+(p[i]*s[i]);
        }
        cout<<"\n\nTotal Profit is : "<<pro;
}
```

SAMPLE INPUT AND OUTPUT:

```
      KNAPSACK PROBLEM
 Enter the number of elements: 4
 Enter the weight: 2
 Enter the profit: 10
 Enter the weight: 5
 Enter the profit: 15
 Enter the weight: 8
 Enter the profit: 25
 Enter the weight: 12
 Enter the profit: 30


Enter Capacity of the bag:20




Profit by weight in decreasing order:

     Weight    Profit   Ratio
     ------    ------   -----
```

2	10	5
8	25	3.125
5	15	3
12	30	2.5

Weight	Load	profit
2	1	10
8	1	25
5	1	15
12	0.416667	12.5

Total Profit is : 62.5

Result:

Thus the program for Knapsack problem has been executed successfully.

SINGLE SOURCE SHORTEST PATH ALGORITHM

Aim:

To write a C++ program to implement single source shortest path algorithm using greedy method.

Algorithm:

1. Start.
2. Initialize the variables i, j, min, n, b[10],cost[10][10], k, dist[10], u, v, w, num, source[10] inside the class path.
3. In getdata(), get the number of nodes along with the cost between the edges and print the cost of the vertices for i=1 to n and for j = 1 to n.
4. Get the source vertex and in spath(int v) for i=1 to n , source[i]=0; and dist[i]=cost[v][i].Also source[v]=1;b[k++]=v;dist[v]=0;
5. For num=2 to n min=9999; and for j= 1 to n , if((source[j]==0)&&(min>dist[j])) then min=dist[j] and u=j. Also source[u]=1;b[k++]=u.
6. For w= 1 to n , if((source[w]==0)&&(dist[w]>dist[u]+cost[u][w])) then, dist[w]=dist[u]+cost[u][v].
7. In display(), for i=1 to k display the shortest path for the given input graph.
8. Stop.

SOURCE CODE:

```
//          SHORTEST PATH ALGORITHM
#include<iostream.h>
#include<conio.h>
class path
{
        private:
                int i,j,min,n,cost[10][10],dist[10],v;
                int u,v,w,num,s[10];
        public:
                void getdata();
                void spath();
                void display();
                void distance();
};
void path::getdata()
{
        cout<<"\n**** SINGLE SOURCE SHORTEST PATH**** ";
        cout<<"\n\n\nEnter the number of nodes : ";
        cin>>n;
        cout<<"\nIf there is no edge then enter 9999\n";
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                {
                        if(i==j)
                        {
```

```
                    cost[i][j]=9999;
                }
                else
                {
                cout<<"\nEnter the cost between "<<i<<" and "<<j<<" : ";
                cin>>cost[i][j];
                }
            }
        }
}

void path::display()
{
        cout<<"The Adjacency Matrix \n";
        for(i=1;i<=n;i++)
        {
            for(j=1;j<=n;j++)
            {
                cout<<cost[i][j]<<"\t";
            }
            cout<<"\n";
        }
}

void path::spath()
{
        cout<<"\nEnter the source vertex : ";
        cin>>v;

        for(i=1;i<=n;i++)
        {
            s[i]=0;
            dist[i]=cost[v][i];
        }
        s[v]=1;
        dist[v]=0;
        for(num=2;num<=n;num++)
        {
            min=9999;
            for(j=1;j<=n;j++)
            {
                if((s[j]==0)&&(min>dist[j]))
                {
                    min=dist[j];
                    u=j;
                }
            }
            s[u]=1;
            for(w=1;w<=n;w++)
            {
```

```
                        if((s[w]==0)&&(dist[w]>dist[u]+cost[u][w]))
                        {
                                dist[w]=dist[u]+cost[u][w];
                        }
                }
                cout<<"\n";
        }
}

void path::distance()
{
        for(w=1;w<=n;w++)
        {
                cout<<"Distance between " << v << " to " << w << " is " <<dist[w];
                cout<< "\n";
        }
}



void main()
{
        clrscr();
        path p;
        p.getdata();
        p.display();
        p.spath();
        p.distance();
        getch();
}

//Source: 1
//
//Find nearest unvisited node from source
//Ex: 2
//
//Travel by alternate unvisited node and compare distance
//Ex: 1 to 3 and 3 to 2
//
//If alternate path is short, update dist[2]
```

SAMPLE INPUT AND OUTPUT:

```
**** SINGLE SOURCE SHORTEST PATH****


Enter the number of nodes :  4

If there is no edge then enter 9999
```

Enter the cost between 1 and 2 : 10

Enter the cost between 1 and 3 : 8

Enter the cost between 1 and 4 : 5

Enter the cost between 2 and 1 : 5

Enter the cost between 2 and 3 : 9999

Enter the cost between 2 and 4 : 2

Enter the cost between 3 and 1 : 9999

Enter the cost between 3 and 2 : 9999

Enter the cost between 3 and 4 : 9999

Enter the cost between 4 and 1 : 9999

Enter the cost between 4 and 2 : 2

Enter the cost between 4 and 3 : 1
The Adjacency Matrix

9999	10	8	5
5	9999	9999	2
9999	9999	9999	9999
9999	2	1	9999

Enter the source vertex : 1

Distance between 1 to 1 is 0
Distance between 1 to 2 is 7
Distance between 1 to 3 is 6
Distance between 1 to 4 is 5

Result:

Thus the C++ program for single source shortest path has been executed successfully.

ALL PAIR SHORTEST PATH ALGORITHM

Aim:

To write a C++ program to implement all pair shortest path algorithm using greedy method.

Algorithm:

1. Start the program.
2. Get the total number of vertices and get the cost of edges.
3. Display the adjacency matrix for the vertices.
4. Find the minimum cost of each vertex using floyd's algorithms.
5. Calculate the distance between each and every vertices.
6. If the cost of each vertices is minimum then the path assign the new cost to the edges.
7. Stop the program

SOURCE CODE:

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

int a[20][20],cost[20][20],i,j,n,k;

class path
{
	public:
		void get();				// TO GET THE COST OF EDGES
		void allpair(int a[20][20],int);
		void display();			// TO DISPLAY THE OUTPUT MATRIX
};

// GETS THE COST OF EACH EDGE
void path::get()
{
	cout<<"\n\t\tALL PAIRS SHORTEST PATH ALGORITHM";
	cout<<"\n\nEnter the total no of vertices:";
	cin>>n;
	cout<<"\nEnter the cost:";
	cout<<"\n\nIf there is no edge enter 9999\n";
	for(i=1;i<=n;i++)
	{
		for(j=1;j<=n;j++)
		{
			if(i==j)
			{
				cost[i][j]=0;
```

```
                        a[i][j]=0;
                    }
                    else
                    {
                        cout<<"\nEnter the cost between "<<i<<" and "<<j<<" : ";
                        cin>>cost[i][j];
                        a[i][j]=cost[i][j];
                    }
                }
            }


}
void path::allpair(int a[20][20], int n)
{
        for(k=1;k<=n;k++)
        {
                for(i=1;i<=n;i++)
                {
                        for(j=1;j<=n;j++)
                        {
                                if((a[i][k]+a[k][j])<a[i][j])
                                {
                                        a[i][j]=a[i][k]+a[k][j];
                                }
                        }
                }
        }
}
void path::display()
{
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                {
                        cout<<a[i][j];
                        cout<<"\t";
                }
                cout<<"\n\n";
        }
}
void main()
{
        path p;
        clrscr();
        p.get();
        cout<<"\n\nADJACENCY MATRIX\n\n";
        p.display();
        p.allpair(a,n);
        cout<<"\nALL PAIRS SHORTEST PATH\n\n";
```

```
        p.display();
        getch();
}
```

DYNAMIC ALL PAIR SHORTEST PATH

OUTPUT

ALL PAIRS SHORTEST PATH ALGORITHM

Enter the total no of vertices:3
Enter the cost:
If there is no edge enter 9999
Enter the cost between 1 and 2 : 1
Enter the cost between 1 and 3 : 11
Enter the cost between 2 and 1 : 6
Enter the cost between 2 and 3 : 4
Enter the cost between 3 and 1 : 7
Enter the cost between 3 and 2 : 8

ADJACENCY MATRIX

0	1	11
6	0	4
7	8	0

ALL PAIRS SHORTEST PATH

0	1	5
6	0	4
7	8	0

Result:

Thus the C++ program for single source shortest path has been executed successfully.

N QUEEN PROBLEM

Aim:

To write a C++ program to implement N Queen Problem.

Algorithm:

Algorithm place (k,I)
//return true if a queen can be placed in k th row and I th column. otherwise it returns //
//false .X[] is a global array whose first k-1 values have been set. Abs® returns the //absolute value of r.
{
For j=1 to k-1 do
If ((X [j]=I) //two in same column.
Or (abs (X [j]-I)=Abs (j-k)))
Then return false;
Return true;
}

Algorithm Nqueen (k,n)
//using backtracking it prints all possible positions of n queens in „n*n‟ chessboard. So
//that they are non-tracking.
{
For I=1 to n do
{
If place (k,I) then
{
X [k]=I;
If (k=n) then write (X [1:n]);
Else nquenns(k+1,n) ;
}
}
}

SOURCE CODE:

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
class queen
{
        public:
                int n,k,t,x[50];
                void get();
                void nqueens(int,int);
                int place(int,int);
};
void queen::get()
{
        cout<<"\n\t\tNQUEENS PROBLEM.";
        cout<<"\nEnter the value for n :";
```

```
	cin>>n;
	nqueens(1,n);
}
void queen::nqueens(int k,int n)
{
	for(int i=1;i<=n;i++)
	{
		if(place(k,i))
		{
			x[k]=i;
			if(k==n)
			{
				cout<<"\n\nThe solution is\n";
				for(int j=1;j<=n;j++)
				{
					cout<<"\n";
					cout<<"\n Queen "<<j<<"\t"<<x[j];
				}
			}
			else
			{
				nqueens(k+1,n);
			}
		}
	}
}
int queen::place(int k,int i)
{
	for(int j=1;j<k;j++)
	{
		if((x[j]==i)||(abs(x[j]-i)==abs(j-k)))
		{
			return 0;
		}
	}
	return 1;
}
void main()
{
	int n;
	clrscr();
	queen q;
	q.get();
	getch();
}
```

SAMPLE INPUT AND OUTPUT:

NQUEENS PROBLEM

Enter the value for n :4
The solution is

Queen 1	2
Queen 2	4
Queen 3	1
Queen 4	3

The solution is

Queen 1	3
Queen 2	1
Queen 3	4
Queen 4	2

Result:

Thus the program for N queen problem has been executed successfully.

SUM OF SUBSET

Aim:

To write a C++ program to implement sum of subset.

Algorithm:

Step 1: Start the program.
Step 2: Declare a class subset with function getdata, print, sumfosub.
Step 3: The function getdata is used to get the value of n capacity.
Step 4: Then function sumofsub is used to calculate the sum of the weights do not exceed the maximum capacity.
Step 5: The function print is used to display the output.
Step 6: The object is created to call the function in main.
Step 7: Stop the program.

SOURCE CODE:

```
//Sum Of Subset
// Sorted data
#include<iostream.h>
#include<conio.h>
int w[20],x[20],m;
float r;

class subset
{
        public:
                void getdata();
                void print(int);
                void sumofsub(float,int,float);
};

void subset::getdata()
{
        int i,n;
        cout<<"\nEnter the value of N:";
        cin>>n;
        cout<<"\nEnter the capacity:";
        cin>>m;
        r=0;
        for(i=1;i<=n;i++)
        {
                cout<<"\nWeight "<<i<<":";
                cin>>w[i];
                r=r+w[i];
        }
}

// s - loaded
// r - remaining
// w[k] - current load
```

```
// k - load number
// m - capacity
void subset::sumofsub(float s,int k,float r)
{
        x[k]=1;
        if(s+w[k]==m)
        {
                print(k);
        }
        else if (s+w[k]+w[k+1] <= m)
        {
                sumofsub(s+w[k],k+1,r-w[k]);
        }

        if((s+r-w[k] >= m) && (s+w[k+1] <= m))
        {
                x[k]=0;
                sumofsub(s,k+1,r-w[k]);
        }
}

void subset::print(int k)
{
        int j;
        for(j=1;j<=k;j++)
        {
                cout<<x[j]<<"\t";
        }
        cout<<"\n\n";
}

void main()
{
        clrscr();
        subset s1;
        cout<<"\n\n SUM OF SUBSET:\n\n";
        s1.getdata();
        s1.sumofsub(0,1,r);
        getch();
}
```

SAMPLE INPUT AND OUTPUT:

SUM OF SUBSET:
Enter the value of N:5

Enter the capacity:15

Weight 1:3

Weight 2:2

Weight 3:10

Weight 4:5

Weight 5:15
1 1 1

0 0 1 1

0 0 0 0 1

Result:

Thus the program for sum of subset has been executed successfully.