LABORATORY MANUAL

DESIGN AND ANALYSIS OF ALGORITHMS

21CS42

2022-2023

ATRIA INSTITUTE OF TECHNOLOGY
Adjacent to Bangalore Baptist Hospital,
ANAND NAGAR, BANGALORE

SYLLABUS

DESIGN AND ANALYSIS OF ALGORITHMS LABORATORY

Sub. Code: 21CSL42 **IA Marks :20**

Number of Lecture Hors/Week: 2 H

Module-1

1. Sort a given set of n integer elements using the 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. Demonstrate using C++/Java how the brute force method works along with its time complexity analysis: worst case, average case and best case

Module-2

1. 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. Demonstrate using C++/Java how the divide-and-conquer method works along with its time complexity analysis: worst case, average case and best case.
2. 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. Demonstrate using C++/Java how the divide-and-conquer method works along with its time complexity analysis: worst case, average case and best case

Module-3

1. Write & Execute C++/Java Program To solve Knapsack problem using Greedy method.
2. Write & Execute C++/Java Program To find shortest paths to other vertices from a given vertex in a weighted connected graph, using Dijkstra's algorithm.
3. Write & Execute C++/Java Program To find Minimum Cost Spanning Tree of a given connected undirected graph using Kruskal's algorithm. Use Union-Find algorithms in your program.
4. Write & Execute C++/Java Program To find Minimum Cost Spanning Tree of a given connected undirected graph using Prim's algorithm

Module-4

1. Write C++/ Java programs to Solve All-Pairs Shortest Paths problem using Floyd's algorithm.
2. Write C++/ Java programs to Solve Travelling Sales Person problem using Dynamic programming.
3. Write C++/ Java programs to Solve 0/1 Knapsack problem using Dynamic Programming method

Module-5

1. Design and implement C++/Java 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, there are two solutions {1, 2, 6} and {1, 8}. Display a suitable message, if the given problem instance doesn't have a solution.
2. Design and implement C++/Java Program to find all Hamiltonian Cycles in a connected undirected Graph G of n vertices using backtracking principle.

Module 1

1. **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. Demonstrate using C++/Java how the brute force method works along with its time complexity analysis: worst case, average case and best case.**

```
import java.util.Random;
import java.util.Scanner;
public class SelectionSort

{
    public static int SIZE = 7000;
    public static void main(String[] args) throws ArrayIndexOutOfBoundsException
    {
        int a[] = new int[SIZE];
          System.out.println("Enter the total number of elements for sorting:");
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        Random m = new Random();

         for (int i = 0; i < n; i++)

        {

              a[i] = m.nextInt(10) + 1;

        }

        System.out.println("\nThe elements before sorting....");

        for (int i = 0; i < n; i++)

        {

          System.out.println("" + a[i]);

        }

        long start_time, end_time;

        start_time = System.nanoTime();

        selectionSort(a, n);

        end_time = System.nanoTime();

        System.out.println("\nThe elements after sorting");

        for (int i = 0; i < n; i++)
```

```
                {
                        System.out.println("" + a[i]);
                }
                System.out.println("\nThe time required for sorting " + n + " numbers is: " + (end_time -
start_time) + " ns");
        }
        static void selectionSort(int[] a, int n)
        {
                for (int i = 0; i <n - 1; i++)
                {
                        int minIndex = i;
                        for (int j = i + 1; j <n ; j++)
                        {
                                if (a[j] < a[minIndex])
                                {           minIndex = j;
                                }
                        }
                        int temp = a[i];
                        a[i] = a[minIndex];
                        a[minIndex] = temp;
                }
        }
}
```

OUTPUT

Enter the total number of elements for sorting:

10

The elements before sorting....

73

86

6

56

62

68

60

59

40

19

The elements after sorting

6

19

40

56

59

60

62

68

73

86

The time required for sorting 10 numbers is: 4840 ns

****

Module 2

1. **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* on graph sheet. The elements can be read from a file or can be generated using the random number generator. Demonstrate using Java how the divide -and- conquer method works along with its time complexity analysis: worst case, average case and best case.**

```
import java.util.Random;
import java.util.Scanner;


public class quicksort {
static int max=2000;
        int partition (int[] a, int low,int high)
        {
                int p,i,j,temp;
                p=a[low];
                i=low+1; j=high;
                while(low<high)
                {
                        while(a[i]<=p&&i<high)
                                i++;
                        while(a[j]>p)
                                j--;
                        if(i<j)
                        {
                                temp=a[i];
                                a[i]=a[j];
                                a[j]=temp;
                        }
                        else
                        {
                                temp=a[low];
                                a[low]=a[j];
                                a[j]=temp;
                                return j;
                        }
                }
         return j;
        }
```

```
void sort(int[] a,int low,int high)
{

        if(low<high)
        {
        int s=partition(a,low,high);
              sort(a,low,s-1);
              sort(a,s+1,high);
        }
}

public static void main(String[] args) {
        int[] a;
        int i;
        System.out.println("Enter the array size");
        Scanner sc =new Scanner(System.in);
        int n=sc.nextInt();
        a= new int[max];
        Random generator=new Random();
        for( i=0;i<n;i++)
        a[i]=generator.nextInt(20);
        System.out.println("Array before sorting");
        for( i=0;i<n;i++)
                System.out.println(a[i]+" "); long
        startTime=System.nanoTime();

        quicksort m=new quicksort();
        m.sort(a,0,n-1);
        long stopTime=System.nanoTime(); long
        elapseTime=(stopTime-startTime);
            System.out.println("Time taken to sort array is:"+elapseTime+"nano seconds");

        System.out.println("Sorted array is");
        for(i=0;i<n;i++)
}               System.out.println(a[i]);


}
```

OUTPUT:

Enter the array size 10
Array before sorting 17
17
12
2
10
3
18
15
15
17
Time taken to sort array is:16980 nano seconds
Sorted array is
2
3
10
12
15
15
17
17
17
18

2. 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* on graph sheet. The elements can be read from a file or can be generated using the random number generator. Demonstrate using Java how the divideand- conquer method works along with its time complexity analysis: worst case, average case and best case.

```
import java.util.Random;
import  java.util.Scanner;
public class mergesort {
        static int max=10000;
         void merge( int[] array,int low, int mid,int high)
        {
                int  i=low;
                int j=mid+1;
                int k=low;
                int[]resarray;
                resarray=new int[max];

                while(i<=mid&&j<=high)
                {
                        if(array[i]<array[j])
                        {


                                resarray[k]=array[i];
                                i++;
                                k++;
                        }
                        else
                        {
                                resarray[k]=array[j];
                                j++;
                                k++;
                        }
                }
```

```
        while(i<=mid)
            resarray[k++]=array[i++];
        while(j<=high)
            resarray[k++]=array[j++];
        for(int m=low;m<=high;m++)
            array[m]=resarray[m];
    }


     void sort( int[] array,int low,int high)
    {
        if(low<high){
            int mid=(low+high)/2;
            sort(array,low,mid);
            sort(array,mid+1,high);
            merge(array,low,mid,high);
        }
    }

    public static void main(String[] args) {
      int[] array;
      int i;
        System.out.println("Enter the array size");
        Scanner sc =new Scanner(System.in);
        int n=sc.nextInt();
        array= new int[max];
        Random generator=new Random();
        for( i=0;i<n;i++)
        array[i]=generator.nextInt(20);
        System.out.println("Array before sorting");
        for( i=0;i<n;i++)
            System.out.println(array[i]+" ");
        long startTime=System.nanoTime();
        mergesort m=new mergesort();
        m.sort(array,0,n-1);
        long stopTime=System.nanoTime();
        long elapseTime=(stopTime-startTime);
        System.out.println("Time taken to sort array is:"+elapseTime+"nano
          seconds");
```

```
            System.out.println("Sorted array is");
            for(i=0;i<n;i++)
                System.out.println(array[i]);
    }
}
```

Output:

Enter the array size 10
Array before sorting 13
9
13
16
13
3
0
6
4
5
Time taken to sort array is:171277nano seconds
Sorted array is
0
3
4
5
6
9
13
13
13
16

****

Module 3

1. Write and Execute C++/Java Program to solve Knapsack problem using Greedy method.

```
import java.util.Scanner;
public class KnapsackGreedy {
public static void main(String[] args) {
    int i, j = 0, max_qty, m, n;
    float sum = 0, max;
    Scanner sc = new Scanner(System.in);
    int array[][] = new int[2][20];

    System.out.println("Enter no of items");
    n = sc.nextInt();

    System.out.println("Enter the weights of each item");
    for (i = 0; i < n; i++)
        array[0][i] = sc.nextInt();

    System.out.println("Enter the values of each item");
    for (i = 0; i < n; i++)
        array[1][i] = sc.nextInt();

    System.out.println("Enter maximum volume of knapsack:");
    max_qty = sc.nextInt();
    m = max_qty;

    while (m >= 0) {
        max = 0;
        for (i = 0; i < n; i++) {
            if (((float) array[1][i]) / ((float) array[0][i]) > max) {
                max = ((float) array[1][i]) / ((float) array[0][i]);
                j = i;
            }
```

```
            }
            if (array[0][j] > m) {
                System.out.println("Quantity of item number: " + (j + 1) + " added is " + m);
                sum += m * max;
                m = -1;
            } else {
                System.out.println("Quantity of item number: " + (j + 1) + " added is " + array[0][j]);
                m -= array[0][j];
                sum += (float) array[1][j];
                array[1][j] = 0;
            }
        }
        System.out.println("The total profit is " + sum);
        sc.close();
    }
}
```

Output:

```
Enter no of items 4
Enter the weights of each items
2
1
3
2
Enter the values of each items
12
10
20
15
Enter maximum volume of knapsack : 5
Quantity of item number: 2 added is 1
Quantity of item number: 4 added is 2
Quantity of item number: 3 added is 2
The total profit is 38.333332
```

2. Write & Execute C++/Java Program To find shortest paths to other vertices from a given vertex in a weighted connected graph, using Dijkstra's algorithm.

```
import java.util.Scanner;

public class Dijkstra {

        int d[] = new int[10];
        int p[] = new int[10];
        int visited[] = new int[10];

        public void dijk(int[][] a, int s, int n) {
                int u = -1, v, i, j, min;

                for (v = 0; v < n; v++) {
                        d[v] = 99;
                        p[v] = -1;
                }

                d[s] = 0;

                for (i = 0; i < n; i++) {
                        min = 99;

                        for (j = 0; j < n; j++) {
                                if (d[j] < min && visited[j] == 0) {
                                        min = d[j];
                                        u = j;
                                }
                        }

                        visited[u] = 1;

                        for (v = 0; v < n; v++) {
                                if ((d[u] + a[u][v] < d[v]) && (u != v) && visited[v] == 0) {
                                        d[v] = d[u] + a[u][v];
                                        p[v] = u;
                                }
                        }
                }
        }

        void path(int v, int s) {
                if (p[v] != -1)
                        path(p[v], s);

                if (v != s)
                        System.out.print("->" + v + " ");
        }
```

```
void display(int s, int n) {
		int i;
		for (i = 0; i < n; i++) {
			if (i != s) {
				System.out.print(s + " ");
				path(i, s);
			}

			if (i != s)
				System.out.print("=" + d[i] + " ");

			System.out.println();
		}
	}

	public static void main(String[] args) {
		int a[][] = new int[10][10];
		int i, j, n, s;

		System.out.println("enter the number of vertices");
		Scanner sc = new Scanner(System.in);
		n = sc.nextInt();

		System.out.println("enter the weighted matrix");

		for (i = 0; i < n; i++)
			for (j = 0; j < n; j++)
				a[i][j] = sc.nextInt();

		System.out.println("enter the source vertex");
		s = sc.nextInt();

		Dijkstra tr = new Dijkstra();
		tr.dijk(a, s, n);

		System.out.println("the shortest path between source " + s + " to remaining vertices
are");
		tr.display(s, n);

		sc.close();
	}
}
```

Output:

enter the number of vertices 5
enter the weighted matrix

0	3	99	7	99
3	0	4	2	99
99	4	0	5	6
5	2	5	0	4
99	99	6	4	0

enter the source vertex 0
the shortest path between source0to remaining vertices are

0 ->1 =3
0 ->1 ->2 =7
0 ->1 ->3 =5
0 ->1 ->3 ->4 =9

3. Write & Execute C++/Java Program to find Minimum Cost Spanning Tree of a given connected undirected graph using Kruskal's algorithm. Use Union-Find algorithms in your program

```
import java.util.Scanner;

public class kruskal {
    int parent[] = new int[10];

    int find(int m) {
        int p = m;
        while (parent[p] != 0)
            p = parent[p];
        return p;
    }

    void union(int i, int j) {
        if (i < j)
            parent[i] = j;
        else
            parent[j] = i;
    }

    void krkl(int[][] a, int n) {
        int u = 0, v = 0, min, k = 0, i, j, sum = 0;
        while (k < n - 1) {
            min = 99;
            for (i = 1; i <= n; i++)
                for (j = 1; j <= n; j++)
                    if (a[i][j] < min && i != j) {
                    }
            i = find(u);
            j = find(v);
            if (i != j) {
                min = a[i][j];
                u = i;
                v = j;
                union(i, j);
                System.out.println("(" + u + "," + v + ")" + "=" + a[u][v]);
                sum = sum + a[u][v];
                k++;
            }
            a[u][v] = a[v][u] = 99;
        }
        System.out.println("The cost of minimum spanning tree = " + sum);
    }

    public static void main(String[] args) {
        int a[][] = new int[10][10];
        int i, j;
        System.out.println("Enter the number of vertices of the graph");
```

```
Scanner sc = new Scanner(System.in);
int n;
n=sc.nextInt();
System.out.println("Enter the wieghted matrix");
for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
                a[i][j]=sc.nextInt();
kruskal k=new kruskal();
k.krkl(a,n);
sc.close();
}
}
```

Output:

```
Enter the number of vertices of the graph 6
Enter the wieghted matrix
0    3    99   99   6    5
3    0    1    99   99   4
99   1    0    6    99   4
99   99   6    0    8    5
6    99   99   8    0    2
5    4    4    5    2    0
(2,3)=1
(5,6)=2
(1,2)=3
(2,6)=4
(4,6)=5
The cost of minimum spanning tree = 15
```

4. Write & Execute C++/Java Program to find Minimum Cost Spanning Tree of a given connected undirected graph using Prim's algorithm.

```
import java.util.Scanner;
 public class prims {
     public static void main(String[] args) {
         int w[][] = new int[10][10];
         int n, i, j, s, k = 0;
         int min;
         int sum = 0;
         int u = 0, v = 0;
         int flag = 0;
         int sol[] = new int[10];

         System.out.println("Enter the number of vertices");
         Scanner sc = new Scanner(System.in);
         n = sc.nextInt();

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

         System.out.println("Enter the weighted graph");
         for (i = 1; i <= n; i++)
             for (j = 1; j <= n; j++)
                 w[i][j] = sc.nextInt();

         System.out.println("Enter the source vertex");
         s = sc.nextInt();
         sol[s] = 1;
         k = 1;

         while (k <= n - 1) {
             min = 99;
             for (i = 1; i <= n; i++)
                 for (j = 1; j <= n; j++)
                     if (sol[i] == 1 && sol[j] == 0)
                         if (i != j && min > w[i][j]) {
                         }
             sol[v] = 1;
             min = w[i][j];
```

```
            u = i;
            v = j;
            sum = sum + min;
            k++;
            System.out.println(u + "->" + v + "=" + min);
        }

        for (i = 1; i <= n; i++)
            if (sol[i] == 0)
                flag = 1;

        if (flag == 1)
            System.out.println("No spanning tree");
        else {
            sc.close();
            System.out.println("The cost of the minimum spanning tree is " + sum);
        }
    }
}
```

Output:

```
Enter the number of vertices 6
Enter the weighted graph
0   3   99  99  6   5
3   0   1   99  99  4
99  1   0   6   99  4
99  99  6   0   8   5
6   99  99  8   0   2
5   4   4   5   2   0
Enter the source vertex 1
1->2=3
2->3=1
2->6=4
6->5=2
6->4=5
The cost of minimum spanning tree is15
```

****

Module 4

1. **Write C++/Java programs to solve All-Pairs Shortest Paths problem using Floyd's algorithm**.

```
import java.util.Scanner;

public class floyd {

	void flyd(int[][] w,int n)
	{
		int i,j,k;
		for(k=1;k<=n;k++)
			for(i=1;i<=n;i++)
				for(j=1;j<=n;j++)
					w[i][j]=Math.min(w[i][j], w[i][k]+w[k][j]);
	}

	public static void main(String[] args) {
		int a[][]=new int[10][10]; int
		n,i,j;
		System.out.println("enter the number of vertices"); Scanner
		sc=new Scanner(System.in);
		n=sc.nextInt();

		System.out.println("Enter the weighted matrix");

		for(i=1;i<=n;i++)
			for(j=1;j<=n;j++)
				a[i][j]=sc.nextInt(); floyd

		f=new floyd();
		f.flyd(a, n);

		System.out.println("The shortest path matrix is");
		for(i=1;i<=n;i++)
		{
			for(j=1;j<=n;j++)
			{
				System.out.print(a[i][j]+" ");
			}
			System.out.println();
		}
		sc.close();

	}

}
```

Output:

enter the number of vertices 4
Enter the weighted matrix

0	99	3	99
2	0	99	99
99	7	0	1
6	99	99	0

The shortest path matrix is

0	10	3	4
2	0	5	6
7	7	0	1
6	16	9	0

2. Write & Execute C++/Java Program to Solve Travelling Salesperson problem using Dynamic programming:

```
import java.util.Scanner;

  class TSPExp {
    int weight[][], n, tour[], finalCost;
    final int INF = 1000;

    TSPExp() {
      Scanner s = new Scanner(System.in);
      System.out.println("Enter the number of nodes:");
      n = s.nextInt();
      weight = new int[n][n];
      tour = new int[n - 1];

      for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
          if (i != j) {
            System.out.println("Weight from node " + (i + 1) + " to " + (j + 1) + ":");
            weight[i][j] = s.nextInt();
          }
        }
      }

      System.out.println();
      System.out.println("Starting node assumed to be node 1.");
      eval();
    }

    public int COST(int currentNode, int inputSet[], int setSize) {
      if (setSize == 0)
        return weight[currentNode][0];
      int min = INF;
      int setToBePassedOnToNextCallOfCOST[] = new int[n - 1];

      for (int i = 0; i < setSize; i++) {
        int k = 0; // initialize new set
        for (int j = 0; j < setSize; j++) {
          if (inputSet[i] != inputSet[j])
            setToBePassedOnToNextCallOfCOST[k++] = inputSet[j];
        }

        int temp = COST(inputSet[i], setToBePassedOnToNextCallOfCOST, setSize - 1);
        if ((weight[currentNode][inputSet[i]] + temp) < min) {
          min = weight[currentNode][inputSet[i]] + temp;
        }
      }
      return min;
    }

    public int MIN(int currentNode, int inputSet[], int setSize) {
```

```
    if (setSize == 0)
        return weight[currentNode][0];
    int min = INF, minindex = 0;
    int setToBePassedOnToNextCallOfCOST[] = new int[n - 1];

    for (int i = 0; i < setSize; i++) { // considers each node of inputSet
        int k = 0;
        for (int j = 0; j < setSize; j++) {
            if (inputSet[i] != inputSet[j])
                setToBePassedOnToNextCallOfCOST[k++] = inputSet[j];
        }

        int temp = COST(inputSet[i], setToBePassedOnToNextCallOfCOST, setSize - 1);
        if ((weight[currentNode][inputSet[i]] + temp) < min) {
            min = weight[currentNode][inputSet[i]] + temp;
            minindex = inputSet[i];
        }
    }
    return minindex;
}

public void eval() {
    int dummySet[] = new int[n - 1];
    for (int i = 1; i < n; i++)
        dummySet[i - 1] = i;
    finalCost = COST(0, dummySet, n - 1);
    constructTour();
}

public void constructTour() {
    int previousSet[] = new int[n - 1];
    int nextSet[] = new int[n - 2];
    for (int i = 1; i < n; i++)
        previousSet[i - 1] = i;
    int setSize = n - 1;
    tour[0] = MIN(0, previousSet, setSize);

    for (int i = 1; i < n - 1; i++) {
        int k = 0;
        for (int j = 0; j < setSize; j++) {
            if (tour[i - 1] != previousSet[j])
                nextSet[k++] = previousSet[j];
        }
        setSize--;
        tour[i] = MIN(tour[i - 1], nextSet, setSize);
        for (int j = 0; j < setSize; j++)
            previousSet[j] = nextSet[j];
    }
    display();
}

public void display() {
    System.out.println();
```

```
        System.out.print("The tour is 1-");
        for (int i = 0; i < n - 1; i++)
            System.out.print((tour[i] + 1) + "-");
        System.out.print("1");
        System.out.println();
        System.out.println("The final cost is " + finalCost);
    }
}

class TSP {
    public static void main(String args[]) {
        TSPExp obj = new TSPExp();
    }
}
```

Output:

```
Enter weight of 1 to 2:=>2
Enter weight of 1 to 3:=>5
Enter weight of 1 to 4:=>7
Enter weight of 2 to 1:=>2
Enter weight of 2 to 3:=>8
Enter weight of 2 to 4:=>3
Enter weight of 3 to 1:=>5
Enter weight of 3 to 2:=>8
Enter weight of 3 to 4:=>1
Enter weight of 4 to 1:=>7
Enter weight of 4 to 2:=>3
Enter weight of 4 to 3:=>1

Enter no. of nodes:=> 4

Starting node assumed to be node 1.
The tour is 1-2-4-3-1
The final cost is 11
```

3. **Write C++/ Java programs to Solve 0/1 Knapsack problem using Dynamic Programming method**.

```
import java.util.Scanner;

public class lab6a
{
   static int max(int a, int b)
   {
      return (a > b)? a : b;
   }
   static int knapSack(int W, int wt[], int val[], int n)
   {
      int i, w;
      int [][]K = new int[n+1][W+1];
      for (i = 0; i <= n; i++)
      {
         for (w = 0; w <= W; w++)
         {
            if (i==0 || w==0)
               K[i][w] = 0;
            else if (wt[i-1] <= w)
   K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]],  K[i-1][w]);
            else
               K[i][w] = K[i-1][w];
         }
      }
      return K[n][W];
   }

   public static void main(String args[])
   {
      Scanner sc = new Scanner(System.in);
      System.out.println("Enter the number of items: ");
      int n = sc.nextInt();
      System.out.println("Enter the items weights: ");
      int []wt = new int[n];
      for(int i=0; i<n; i++)
         wt[i] = sc.nextInt();

      System.out.println("Enter the items values: ");
      int []val = new int[n];
      for(int i=0; i<n; i++)
         val[i] = sc.nextInt();

      System.out.println("Enter the maximum capacity: ");
      int W = sc.nextInt();
  System.out.println("The maximum value that can be put in a knapsack of capacity W is: " +
knapSack(W, wt, val, n));
      sc.close();
   }
}
```

OUTPUT:

Enter number of elements

4

Enter weight for 4 elements

2

1

3

2

Enter value for 4 elements

12

10

20

15

Enter knapsack weight 5

The optimal solution is 37

Items selected : 1 2 4

****

Module 5

1. Design and implement in Java 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, there are two solutions {1,2,6}and {1,8}. Display a suitable message, if the given problem instance doesn't have a solution.

```
import java.util.Scanner;

import static java.lang.Math.pow;

public class subSet {

	/**
	 * @param args
	 */
	void subset(int num,int n, int x[])
	{
		int i;
		for(i=1;i<=n;i++)
			x[i]=0;
		for(i=n;num!=0;i--)
		{
			x[i]=num%2;
			num=num/2;
		}
	}

	public static void main(String[] args) {
		// TODO Auto-generated method stub
	int a[]=new int[10];
	int x[]=new int[10];
	int n,d,sum,present=0;
	int j;
	System.out.println("enter the number of elements of set");
	Scanner sc=new Scanner(System.in);
	n=sc.nextInt();

	System.out.println("enter the elements of set");
	for(int i=1;i<=n;i++)
	a[i]=sc.nextInt();

	System.out.println("enter the positive integer sum");
	d=sc.nextInt();

	if(d>0)
	{
	   for(int i=1;i<=Math.pow(2,n)-1;i++)
	   {
```

```
                subSet s=new subSet();
                s.subset(i,n,x);
                sum=0;

                for(j=1;j<=n;j++)
                if(x[j]==1)
                    sum=sum+a[j];
                if(d==sum)
                {
                    System.out.print("Subset={");
                    present=1;
                    for(j=1;j<=n;j++)
                        if(x[j]==1)
                            System.out.print(a[j]+",");


                    System.out.print("}="+d);
                    System.out.println();
                }

            }

        }
        if(present==0)
                System.out.println("Solution does not exists");

      }

}
```

Output:

```
enter the number of elements of set 5
enter the elements of set 1 2 5 6 8
enter the positive integer sum 9
Subset={1,8,}=9
Subset={1,2,6,}=9
```

2. Design and implement the presence of Hamiltonian Cycle in an undirected Graph G of *n* vertices.

```
import java.util.*;

class Hamiltoniancycle {
        private int adj[][], x[], n;

public Hamiltoniancycle() {
         Scanner src = new Scanner(System.in);
        System.out.println("Enter the number of nodes");
        n = src.nextInt();
        x = new int[n];
        x[0] = 0;
        for (int i = 1; i < n; i++)
                x[i] = -1;
        adj = new int[n][n];
        System.out.println("Enter the adjacency matrix");
        for (int i = 0; i < n; i++)
                for (int j = 0; j < n; j++)
                        adj[i][j] = src.nextInt();
}

public void nextValue(int k) {
  int i = 0;
  while (true) {
        x[k] = x[k] + 1;
        if (x[k] == n)
                x[k] = -1;
        if (x[k] == -1)
                 return;
        if (adj[x[k - 1]][x[k]] == 1)
                for (i = 0; i < k; i++)
                        if (x[i] == x[k])
                                break;
        if (i == k)
        if (k < n - 1 || (k == n - 1 && adj[x[n - 1]][0] == 1))
                return;
  }
}

public void getHCycle(int k) {
  while (true) {
        nextValue(k);
        if (x[k] == -1)
                 return;
        if (k == n - 1)
        {
                System.out.println("\nSolution : ");
        for (int i = 0; i < n; i++)
                System.out.print((x[i] + 1) + " ");
      System.out.println(1);
        }
```

```
            else
                    getHCycle(k + 1);
        }
    }
}

class HamiltoniancycleExp {
    public static void main(String args[]) {
        Hamiltoniancycle obj = new Hamiltoniancycle();
        obj.getHCycle(1);
    }
}
```

Output:

```
Enter the number of nodes 6
Enter the adjacency matrix
 0 1 1 1 0 0
 1 0 1 0 0 1
 1 1 0 1 1 0
 1 0 1 0 1 0
 0 0 1 1 0 1
 0 1 0 0 1 0

Solution :
1 2 6 5 3 4 1

Solution :
1 2 6 5 4 3 1

Solution :
1 3 2 6 5 4 1

Solution :
1 3 4 5 6 2 1

Solution :
1 4 3 5 6 2 1

Solution :
1 4 5 6 2 3 1
```

****