DAA Experiment - 1

1. **Fibonacci series using recursive and non- recursive approach**

i) Nonrecursive:

TC -> O(n), SC -> O(1)

```
public static void main(String args[]){
        Scanner sc=new Scanner(System.in);
        int N=sc.nextInt();
        int f1=0;
        int f2=1;
        System.out.println(f1);
        System.out.println(f2);
        for(int i=2 ; i<N ; i++){
            int f3=f1+f2;
            System.out.println(f3);
            f1=f2;
            f2=f3;
        }
}
```

ii) Recursive

TC -> O(2^n *n), SC -> O(n)

```
public static int Fib(int n){
        if(n==0 || n==1){
            return n;
        }
        return Fib(n-1)+Fib(n-2);
}
public static void main(String args[]){
        Scanner sc=new Scanner(System.in);
          int N =sc.nextInt();
        for(int i=0;i<N;i++){
            System.out.println(Fib(i));
 }
```

DAA Experiment -2

Huffman Encoding Using Greedy Strategy

T.C -> O(n log n), S.C -> O(n)

```
import java.util.*;

class Huffman {
public static void printCode (HuffmanNode root, String s)
{
        if (root.left == null && root.right == null &&
        Character.isLetter(root.c)) {
                System.out.println(root.c + ":" + s);
                return;
        }
        printCode(root.left, s + "0");
        printCode(root.right, s + "1");
}

public static void main(String[] args){
        Scanner s = new Scanner(System.in);
        int n = 6;
        char[] charArray = { 'a', 'b', 'c', 'd', 'e', 'f' };
        int[] charfreq = {50, 10, 30, 5, 3, 2 };

        PriorityQueue<HuffmanNode> q = new
        PriorityQueue<HuffmanNode>(

        n, new MyComparator());


        for (int i = 0; i < n; i++) {
                HuffmanNode hn = new HuffmanNode();
                hn.c = charArray[i];
                hn.data = charfreq[i];
                hn.left = null;
                hn.right = null;
                q.add(hn);
        }
}


    HuffmanNode root = null;

    while (q.size() > 1) {

        HuffmanNode x = q.peek();
        q.poll();
        HuffmanNode y = q.peek();
        q.poll();
        HuffmanNode f = new HuffmanNode();
        f.data = x.data + y.data;
        f.c = '-';
        f.left = x;
        f.right = y;
        root = f;
        q.add(f);
    }
    printCode(root, "");
  }
}

// node class is the basic structure
// of each node present in the Huffman - tree.
class HuffmanNode {

    int data;
    char c;
    HuffmanNode left;
    HuffmanNode right;
}
class MyComparator implements Comparator<HuffmanNode> {
    public int compare(HuffmanNode x, HuffmanNode y){
        return x.data - y.data;
    }
}
```

DAA Experiment – 3

Fractional Knapsack problem using Greedy Method

T.C -> O(2^N), S.C -> O(n)

```
import java.lang.*;

import java.util.Arrays;

import java.util.Comparator;

// Greedy approach

public class FractionalKnapSack {

        // Function to get maximum value

        private static double getMaxValue(ItemValue[] arr, int capacity)

        {

                // Sorting items by profit/weight ratio;

                Arrays.sort(arr, new Comparator<ItemValue>() {

                        @Override

                        public int compare(ItemValue item1, ItemValue item2)

                        {

                                double cpr1= new Double((double)item1.profit

                                                                / (double)item1.weight);

                                double cpr2= new Double((double)item2.profit

                                                                / (double)item2.weight);

                                if (cpr1 < cpr2)

                                        return 1;

                                else

                                        return -1;

                        }
});
        double totalValue = 0d;
        for (ItemValue i : arr) {
            int curWt = (int)i.weight;
            int curVal = (int)i.profit;
            if (capacity - curWt >= 0) {
                capacity = capacity - curWt;
                totalValue += curVal;
            }
            else {
                double fraction = ((double)capacity / (double)curWt);
                totalValue += (curVal * fraction);
                capacity= (int)(capacity - (curWt * fraction));
                break;
            }
        }
        return totalValue;
    }
    static class ItemValue {
        int profit, weight;
        public ItemValue(int val, int wt)
        {
            this.weight = wt;
            this.profit = val;
        }
    }
```

```
        // Driver code
        public static void main(String[] args)
        {
                ItemValue[] arr = { new ItemValue(60, 10),new ItemValue(100,
20),new ItemValue(120, 30) };
                int capacity = 50;
                double maxValue = getMaxValue(arr, capacity);
                System.out.println(maxValue);
        }
}
```

DAA Experiment – 4

Knapsack problem using Branch and bound

```
import java.util.*;
class Item {
        float weight;
        int value;
        int idx;
        public Item() {}
        public Item(int value, float weight,int idx){
                this.value = value;
                this.weight = weight;
```

```
                    this.idx = idx;
          }
}

class Node {
          float ub;   // upperBound
          float lb;   //lowerbound
          int level;
          boolean flag;
          float tv;   //total value
          float tw;       //Total weight
          public Node() {}
          public Node(Node cpy)
          {
                    this.tv = cpy.tv;
                    this.tw = cpy.tw;
                    this.ub = cpy.ub;
                    this.lb = cpy.lb;
                    this.level = cpy.level;
                    this.flag = cpy.flag;
          }
}

// Comparator to sort based on lower bound
class sortByC implements Comparator<Node> {
          public int compare(Node a, Node b)
```

```
    {
        boolean temp = a.lb > b.lb;
        return temp ? 1 : -1;
    }
}

class sortByRatio implements Comparator<Item> {
    public int compare(Item a, Item b)
    {
        boolean temp = (float)a.value/ a.weight > (float)b.value/ b.weight;
        return temp ? -1 : 1;
    }
}
class knapsack {
    private static int size;
    private static float capacity;
    static float upperBound(float tv, float tw, int idx, Item arr[])
    {
        float value = tv;
        float weight = tw;
        for (int i = idx; i < size; i++) {
            if (weight + arr[i].weight <= capacity) {
                weight += arr[i].weight;
                value -= arr[i].value;
            }
            else {
```

```
            value -= (float)(capacity- weight)/ arr[i].weight
                    * arr[i].value;
            break;
        }
    }
    return value;
}
static float lowerBound(float tv, float tw,int idx, Item arr[])
{
    float value = tv;
    float weight = tw;
    for (int i = idx; i < size; i++) {
        if (weight + arr[i].weight <= capacity) {
            weight += arr[i].weight;
            value -= arr[i].value;
        }
        else {
            break;
        }
    }
    return value;
}

static void assign(Node a, float ub, float lb, int level, boolean flag,
                float tv, float tw)
{
        a.ub = ub;
        a.lb = lb;
        a.level = level;
        a.flag = flag;
        a.tv = tv;
        a.tw = tw;
}

public static void solve(Item arr[])
{
        Arrays.sort(arr, new sortByRatio());
        Node current, left, right;
        current = new Node();
        left = new Node();
        right = new Node();
        float minLB = 0, finalLB= Integer.MAX_VALUE;
        current.tv = current.tw = current.ub= current.lb = 0;
        current.level = 0;
        current.flag = false;
        PriorityQueue<Node> pq = new PriorityQueue<Node>(
                new sortByC());
        pq.add(current);
        boolean currPath[] = new boolean[size];
        boolean finalPath[] = new boolean[size];

        while (!pq.isEmpty()) {
```

```
current = pq.poll();
if (current.ub > minLB || current.ub >= finalLB) {
    continue;
}
if (current.level != 0)
    currPath[current.level - 1]
        = current.flag;

if (current.level == size) {
    if (current.lb < finalLB) {
        for (int i = 0; i < size; i++)
            finalPath[arr[i].idx]
                = currPath[i];
        finalLB = current.lb;
    }
    continue;
}
int level = current.level;
// right node -> Excludes current item
// Hence, cp, cw will obtain the value
// of that of parent
assign(right, upperBound(current.tv,
                        current.tw,
                        level + 1, arr),
    lowerBound(current.tv, current.tw,
                level + 1, arr),
        level + 1, false,
        current.tv, current.tw);

if (current.tw + arr[current.level].weight<= capacity) {
        left.ub = upperBound(
                current.tv
                        - arr[level].value,
                current.tw
                        + arr[level].weight,
                level + 1, arr);
        left.lb = lowerBound(
                current.tv
                        - arr[level].value,
                current.tw
                        + arr[level].weight,
                level + 1,
                arr);
        assign(left, left.ub, left.lb,
                level + 1, true,
                current.tv - arr[level].value,
                current.tw
                        + arr[level].weight);
}

else {
        left.ub = left.lb = 1;
        }

        // Update minLB
        minLB = Math.min(minLB, left.lb);
        minLB = Math.min(minLB, right.lb);

        if (minLB >= left.ub)
            pq.add(new Node(left));
        if (minLB >= right.ub)
            pq.add(new Node(right));
    }
    System.out.println("Items taken"
                    + "into the knapsack are");
    for (int i = 0; i < size; i++) {
        if (finalPath[i])
            System.out.print("1 ");
        else
            System.out.print("0 ");
    }
    System.out.println("\nMaximum profit"
                    + " is " + (-finalLB));
}
public static void main(String args[])
{
    size = 4;
    capacity = 15;
```

```
        Item arr[] = new Item[size];
        arr[0] = new Item(10, 2, 0);
        arr[1] = new Item(10, 4, 1);
        arr[2] = new Item(12, 6, 2);
        arr[3] = new Item(18, 9, 3);

        solve(arr);
    }
}
```

DAA Experiment – 5

N- Queen Matrix using Backtracking

T.C -> O(n!), S.C -> O(n^2)

```
public class NQueenProblem {
    final int N = 4;
    void printSolution(int board[][])
    {
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++)
                System.out.print(" " + board[i][j]+ " ");
            System.out.println();
    }
}
boolean isSafe(int board[][], int row, int col)
{
    int i, j;
    /* Check this row on left side */
    for (i = 0; i < col; i++)
        if (board[row][i] == 1)
            return false;
    /* Check upper diagonal on left side */
    for (i = row, j = col; i >= 0 && j >= 0; i--, j--)
        if (board[i][j] == 1)
            return false;
    /* Check lower diagonal on left side */
    for (i = row, j = col; j >= 0 && i < N; i++, j--)
        if (board[i][j] == 1)
            return false;

    return true;
}
/* A recursive utility function to solve N Queen problem */
boolean solveNQUtil(int board[][], int col)
{
    /* base case: If all queens are placed then return true */
    if (col >= N)
        return true;
        /* Consider this column and try placing this queen in all rows one by one */
        for (int i = 0; i < N; i++) {
            /* Check if the queen can be placed on board[i][col] */
            if (isSafe(board, i, col)) {
                /* Place this queen in board[i][col] */
                board[i][col] = 1;
                /* recur to place rest of the queens */
                if (solveNQUtil(board, col + 1) == true)
                    return true;
                /* If placing queen in board[i][col]
                doesn't lead to a solution then
                remove queen from board[i][col] */
                board[i][col] = 0; // BACKTRACK
            }
        }
        return false;
    }
    boolean solveNQ()
    {
        int board[][] = { { 0, 0, 0, 0 },
                        { 0, 0, 0, 0 },
                        { 0, 0, 0, 0 },
                        { 0, 0, 0, 0 } };

        if (solveNQUtil(board, 0) == false) {
            System.out.print("Solution does not exist");
            return false;
        }

        printSolution(board);
        return true;
    }
    public static void main(String args[])
    {
        NQueenProblem Queen = new NQueenProblem();
        Queen.solveNQ();
    }
}
```