DAA LAB

1. Implement the concept of Linear Search

```
import java.util.Scanner;
public class LinearSearch {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter Number of Elements: ");
        int n = sc.nextInt();
        int[] arr = new int[n]; // Declare array of size n
        System.out.println("Enter the Elements:");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt(); // Input elements into the array
        }
        System.out.print("Enter the search Element: ");
        int key = sc.nextInt();
        boolean found = false;
        for (int i = 0; i < n; i++) {
            if (key == arr[i]) {
                System.out.println(key + " found at position " + (i + 1));
                found = true;
                break; // Exit loop once key is found
            }
        }
        if (!found) {
            System.out.println(key + " not found!");
        }
    }
}
```

2. There are 5 books in the shelf, find the number of ways to select 3 books from 5 books on the shelf using the NCR with recursion.

```
import java.util.Scanner;
public class Exam {
    public static int nCr(int n, int r) {
        if (r == 0 || r == n) {
            return 1;
        } else {
            return nCr(n - 1, r - 1) + nCr(n - 1, r);
        }
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter number of items to choose from: ");
        int total = sc.nextInt();
        System.out.print("Enter number of items to be chosen: ");
```

```
        int choose = sc.nextInt();
        System.out.println("No of ways: " + nCr(total, choose));
        sc.close();
    }
}
```

3. Find the next three terms of the sequence 15, 23, 38, 61, … Fibonacci series of the given number using recursion

```
import java.util.Scanner;
public class Fib {
    static int fib(int x) {
        return (x == 1) ? 15 : (x == 2) ? 23 : fib(x - 1) + fib(x - 2);
    }
    public static void main(String[] args) {
        System.out.println("The next 3 terms of the series 15, 23, 38, 61 is:");
        for (int i = 1; i <= 7; i++) System.out.print(fib(i) + " ");
    }
}
```

The next 3 terms of the series 15, 23, 38, 61 is:
15 23 38 61 99 160 259

4. Implement the selection sort algorithm (Brute Force Technique).

```
import java.util.Scanner;
public class SelectionSort {
    void sort(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n - 1; i++) {
            int minId = i;
            for (int j = i + 1; j < n; j++) {
                if (arr[j] < arr[minId]) {
                    minId = j;
                }
            }
            int temp = arr[minId];
            arr[minId] = arr[i];
            arr[i] = temp;
        }
    }
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.print("No of elements: ");
        int n = in.nextInt();
        int[] arr = new int[n];
        System.out.print("Elements: ");
        for (int i = 0; i < n; i++) {
            arr[i] = in.nextInt();
```

```
        }
        in.close();
        System.out.print("Before sorting: ");
        for (int num : arr) {
            System.out.print(num + " ");
        }
        System.out.println();
        new SelectionSort().sort(arr);
        System.out.print("After sorting: ");
        for (int num : arr) {
            System.out.print(num + " ");
        }
        System.out.println();
    }
}
```

```
no of elements: 5
Elements: 4 7 2 7 4
Before sorting: 4 7 2 7 4 ⏎
After sorting: 2 4 4 7 7 ⏎
```

```
no of elements: 5
Elements: 4 7 2 7 4
Before sorting: 4 7 2 7 4 ⏎
After sorting: 2 4 4 7 7
```

5. Write a program to search a key in a given set of elements using Binary search method and find the time required to find the key.

```
import java.util.Scanner;
public class Exam{
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter Number of Elements: ");
        int n = sc.nextInt();
        int[] arr = new int[n];
        System.out.println("Enter the Elements in ascending order:");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }
        System.out.print("Enter the search Element: ");
        int key = sc.nextInt();
        long startTime = System.nanoTime(); // Start timing
        int result = binarySearch(arr, key);
        long endTime = System.nanoTime(); // End timing
        long timeElapsed = endTime - startTime;
        if (result != -1) {
            System.out.println("The search element " + key + " is found at position: " + (result + 1));
        } else {
            System.out.println("The search element " + key + " is not found");
        }
```

```
        System.out.println("Time required to find the key: " + timeElapsed + " nanoseconds");
        sc.close();
    }
    public static int binarySearch(int[] arr, int key) {
        int low = 0, high = arr.length - 1;
        while (low <= high) {
            int mid = (low + high) / 2;
            if (arr[mid] == key) {
                return mid;
            } else if (arr[mid] > key) {
                high = mid - 1;
            } else {
                low = mid + 1;
            }
        }
        return -1; // Key not found
    }
}
```

6. Sort a given set of elements using Quick Sort method and determine the time required sort the elements. Plot a graph of number of elements versus time taken. Specify the time efficiency class of this algorithm

```
import java.util.*;
class Exam {
    static final int MAX = 5000;
    void quickSort(int arr[], int l, int h) {
        if (l < h) {
            int p = partition(arr, l, h);
            quickSort(arr, l, p - 1);
            quickSort(arr, p + 1, h);
        }
    }
    int partition(int arr[], int l, int h) {
        int pivot = arr[h];
        int i = l - 1;
        for (int j = l; j < h; j++) {
            if (arr[j] < pivot) {
                i++;
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
        int temp = arr[i + 1];
        arr[i + 1] = arr[h];
        arr[h] = temp;
```

```
        return i + 1;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter number of elements: ");
        int n = sc.nextInt();
        Random gen = new Random();
        int arr[] = new int[MAX];
        for (int i = 0; i < n; i++) {
            arr[i] = gen.nextInt(1000);
        }
        System.out.println("Random elements before sorting:");
        for (int i = 0; i < n; i++) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();
        long start = System.nanoTime();
        QuickSort qs = new QuickSort();
        qs.quickSort(arr, 0, n - 1);
        long stop = System.nanoTime();
        System.out.println("Array after sorting:");
        for (int i = 0; i < n; i++) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();
        System.out.println("Time taken: " + (stop - start) + " ns");
    }
}
```

7. Implement Prim's algorithm and Find Minimum Cost Spanning Tree of a given connected undirected graph.

```
import java.util.Scanner;
public class prims {
    final static int MAX = 20;
    static int n;
    static int[][] cost = new int[MAX][MAX];
    static Scanner scan = new Scanner(System.in);
    public static void main(String[] args) {
        readMatrix();
        primsAlgorithm();
    }
    static void readMatrix() {
        System.out.println("Enter the number of nodes:");
        n = scan.nextInt();
        System.out.println("Enter the adjacency matrix:");
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
```

```
            cost[i][j] = scan.nextInt();
            if (cost[i][j] == 0) cost[i][j] = 999;
        }
    }
  }
  static void primsAlgorithm() {
    int[] visited = new int[n + 1];
    int ne = 1, mincost = 0;
    visited[1] = 1;
    while (ne < n) {
      int min = 999, a = -1, b = -1;
      for (int i = 1; i <= n; i++) {
        if (visited[i] == 1) {
          for (int j = 1; j <= n; j++) {
            if (cost[i][j] < min && visited[j] == 0) {
              min = cost[i][j];
              a = i;
              b = j;
            }
          }
        }
      }
      if (a != -1 && b != -1) {
        System.out.println("Edge " + ne++ + ": (" + a + ", " + b + ") cost: " + min);
        mincost += min;
        visited[b] = 1;
        cost[a][b] = cost[b][a] = 999;
      }
    }
    System.out.println("Minimum cost: " + mincost);
  }
}
```

```
Enter the no. of vertices
5
Enter the cost adjacency matrix
0 1 5 0 2
1 0 0 4 1
5 0 0 3 0
0 4 3 0 1
2 1 0 1 0
The edges of Minimum Cost Spanning Tree are
1 edge (1,2) = 1
2 edge (2,5) = 1
3 edge (4,5) = 1
4 edge (3,4) = 3
Minimum cost: 6
```

8. Implement Kruskal's algorithm and Find Minimum Cost Spanning Tree of a given connected undirected graph.

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

```
public class Kruskals {
   static final int MAX = 20;
   static int n;
   static int[][] cost = new int[MAX][MAX];
   static Scanner scan = new Scanner(System.in);
   public static void main(String[] args) {
      readMatrix();
      kruskalsAlgorithm();
   }
   static void readMatrix() {
      System.out.println("Implementation of Kruskal's algorithm\nEnter the number of
vertices:");
      n = scan.nextInt();
      System.out.println("Enter the cost adjacency matrix:");
      for (int i = 0; i < n; i++) {
         for (int j = 0; j < n; j++) {
            cost[i][j] = scan.nextInt();
            if (cost[i][j] == 0) cost[i][j] = 999;
         }
      }
   }
   static void kruskalsAlgorithm() {
      int[] parent = new int[n];
      int ne = 0, mincost = 0;
      System.out.println("The edges of Minimum Cost Spanning Tree are:");
      while (ne < n - 1) {
         int min = 999, a = -1, b = -1;
         for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
               if (cost[i][j] < min) {
                  min = cost[i][j];
                  a = i;
                  b = j;
               }
            }
         }
         int u = find(parent, a);
         int v = find(parent, b);
         if (u != v) {
            System.out.println((ne + 1) + " edge (" + (a + 1) + "," + (b + 1) + ") = " + min);
            mincost += min;
            union(parent, u, v);
            ne++;
         }
         cost[a][b] = cost[b][a] = 999;
      }
      System.out.println("Minimum cost: " + mincost);
   }
```

```
    static int find(int[] parent, int i) {
        if (parent[i] == 0) return i;
        return find(parent, parent[i]);
    }
    static void union(int[] parent, int u, int v) {
        parent[v] = u;
    }
}
```

```
Implementation of Kruskal's algorithm
Enter the no. of vertices:
5
Enter the cost adjacency matrix:
1 0 5 0 2
1 0 0 4 1
5 0 0 3 0
0 4 3 0 1
2 1 0 1 0
The edges of Minimum Cost Spanning Tree are:
1 edge (2,1) = 1
2 edge (2,5) = 1
3 edge (4,5) = 1
4 edge (3,4) = 3
Minimum cost: 6
```

9. Implement Dijkstra's algorithm find shortest paths to other vertices from a given vertex in a weighted connected graph.

```
import java.util.Scanner;
public class Exam {
        public static void main(String[] args) {
                int i, j;
                int dist[] = new int[10], visited[] = new int[10];
                int cost[][] = new int[10][10], path[] = new int[10];
                Scanner in = new Scanner(System.in);
                System.out.print("No of nodes: ");
                int n = in.nextInt();
                System.out.println("Cost matrix:");
                for(i = 1; i <= n; i++)
                        for(j = 1; j <= n; j++)
                                cost[i][j] = in.nextInt();
                System.out.println("Cost matrix is");
                for(i = 1; i <= n; i++) {
                        for(j = 1; j <= n; j++) {
                                System.out.print(cost[i][j] + "\t");
                        }
System.out.println();
                }
                System.out.print("Source vertex: ");
                int sv = in.nextInt();
                dij(cost, dist, sv, n, path, visited);
                printpath(sv, n, dist, path, visited);
        }
```

```
	static void dij(int cost[][],int dist[],int src,int n, int path[],int visited[]) {
		// write your code here...
		int count=2,min,v=0;
		for(int i=1; i<=n;i++) {
			visited[i] = 0;
			dist[i] = cost[src][i];
			if(cost[src][i] == 999)
				path[i]=0;
			else
				path[i] = src;
		}
		visited[src] = 1;
while(count<=n) {
			min = 999;
			for(int w=1;w<=n;w++) {
				if(dist[w]<min && visited[w]==0) {
					min = dist[w];
					v=w;
				}
			}
			visited[v] = 1;
			count++;
			for(int w=1;w<=n;w++) {
				if(dist[w] > (dist[v] + cost[v][w])) {
					dist[w] = dist[v] + cost[v][w];
					path[w] = v;
				}
			}
		}
	}
	static void printpath(int src,int n,int dist[], int path[],int visited[]) {
		for(int w=1;w<=n;w++) {
if(visited[w]==1 && w!=src) {
				System.out.println("The short distance between:
"+src+"-->"+w+" is: "+dist[w]);
				System.out.print("Path is: "+w+"-->");
				int t=path[w];
				while(t!=src) {
					System.out.print(t+"-->");
					t=path[t];
				}
				System.out.print(src+"\n");
			}
		}
	}
}
```

Expected output	Actual output
No ·of ·nodes: ·4	No ·of ·nodes: ·4
Cost ·matrix:⏎	Cost ·matrix:⏎
0 10 5 999	0 10 5 999
999 0 2 1	999 0 2 1
999 3 0 999	999 3 0 999
999 999 999 0	999 999 999 0
Cost ·matrix ·is⏎	Cost ·matrix ·is⏎
0→10→5→999→⏎	0→10→5→999→⏎
999→0→2→1→⏎	999→0→2→1→⏎
999→3→0→999→⏎	999→3→0→999→⏎
999→999→999→0→⏎	999→999→999→0→⏎
Source ·vertex: ·1	Source ·vertex: ·1
The ·short ·distance ·between: ·1-->2 ·is: ·8⏎	The ·short ·distance ·between: ·1-->2 ·is: ·8⏎
Path ·is: ·2-->3-->1⏎	Path ·is: ·2-->3-->1⏎
The ·short ·distance ·between: ·1-->3 ·is: ·5⏎	The ·short ·distance ·between: ·1-->3 ·is: ·5⏎
Path ·is: ·3-->1⏎	Path ·is: ·3-->1⏎
The ·short ·distance ·between: ·1-->4 ·is: ·9⏎	The ·short ·distance ·between: ·1-->4 ·is: ·9⏎
Path ·is: ·4-->2-->3-->1⏎	Path ·is: ·4-->2-->3-->1

10. Implement all-pairs shortest paths problem using Floyd's algorithm.

```
import java.util.Scanner;
class floydss {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      System.out.println("Enter the number of vertices: ");
      int n = sc.nextInt();
      System.out.println("Enter the adjacency matrix (use 999 for infinity): ");
```

```
        int[][] adj = new int[n][n];
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                adj[i][j] = sc.nextInt();
            }
        }
        for (int k = 0; k < n; k++) {
            for (int i = 0; i < n; i++) {
                for (int j = 0; j < n; j++) {
                    adj[i][j] = Math.min(adj[i][j], adj[i][k] + adj[k][j]);
                }
            }
        }
        System.out.println("The all-pair shortest path is: ");
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                System.out.print(adj[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

```
Enter the number of vertices:
4
Enter the adjacency matrix (use 999 for infinity):
0 3 999 5
2 0 999 4
999 1 0 999
999 999 2 0
The all-pair shortest path is:
0 3 7 5
2 0 6 4
3 1 0 5
5 3 2 0
```

11. Implementation of Bellman Ford Algorithm using a directed graph.

```
import java.util.Scanner;
class Graph {
    static class Edge {
        int src, dest, weight;
        Edge(int s, int d, int w) {
            src = s;
            dest = d;
            weight = w;
        }
    }
    int V, E;
    Edge[] edge;
    Graph(int v, int e) {
        V = v;
        E = e;
```

```
        edge = new Edge[e];
    }
    void BellmanFord(int src) {
        int[] dist = new int[V];
        for (int i = 0; i < V; ++i) dist[i] = Integer.MAX_VALUE;
        dist[src] = 0;
        for (int i = 1; i < V; ++i) {
            for (Edge e : edge) {
                if (dist[e.src] != Integer.MAX_VALUE && dist[e.src] + e.weight < dist[e.dest]) {
                    dist[e.dest] = dist[e.src] + e.weight;
                }
            }
        }
        for (Edge e : edge) {
            if (dist[e.src] != Integer.MAX_VALUE && dist[e.src] + e.weight < dist[e.dest]) {
                System.out.println("Graph contains negative weight cycle");
                return;
            }
        }
        printArr(dist);
    }
    void printArr(int[] dist) {
        System.out.println("Vertex Distance from Source");
        for (int i = 0; i < V; ++i)
            System.out.println(i + "\t\t" + dist[i]);
    }
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.print("Enter number of vertices: ");
        int V = in.nextInt();
        System.out.print("Enter number of edges: ");
        int E = in.nextInt();
        Graph graph = new Graph(V, E);
        for (int i = 0; i < E; i++) {
            System.out.print("Enter src, dest and weight for edge " + (i + 1) + ": ");
            int src = in.nextInt();
            int dest = in.nextInt();
            int weight = in.nextInt();
            graph.edge[i] = new Edge(src, dest, weight);
        }
        graph.BellmanFord(0);
    }
}
```

```
Enter number of vertices: 5
Enter number of edges: 7
Enter src, dest and weight for edge 1: 0 1 -1
Enter src, dest and weight for edge 2: 1 2 3
Enter src, dest and weight for edge 3: 1 4 2
Enter src, dest and weight for edge 4: 1 3 2
Enter src, dest and weight for edge 5: 3 1 1
```

```
Enter src, dest and weight for edge 6: 3 2 5
Enter src, dest and weight for edge 7: 4 3 -3
Vertex Distance from Source
0               0
1               -1
2               2
3               -2
4               1
```

12. Implementation of Travelling Salesperson Problem using dynamic programming.

```
import java.util.Arrays;
import java.util.Scanner;
public class Exam {
public static void main(String[] args) {
                Scanner sc = new Scanner(System.in);
                System.out.print("Enter the number of cities: ");
                int n = sc.nextInt();
                System.out.println("Enter the distance between cities: ");
                int[][] distance = new int[n][n];
                for(int i=0;i<n;i++) {
                        for(int j=0;j<n;j++) {
                                distance[i][j] = sc.nextInt();
                        }
                }
                int minCost = tsp(distance, n);
                System.out.println("Minimum cost to visit all cities: "+minCost);
                sc.close();
        }
public static int tsp(int[][] distance, int n) {
                int[][] dp = new int[1 << n][n];
                final int INF = 1_000_000_000;
                for(int[] row : dp) {
                        Arrays.fill(row, INF);
                }
                dp[1][0] = 0;
                for(int mask = 1;mask < (1<<n); mask++) {
                        for(int i=0; i<n; i++) {
                        if((mask & (1<<i)) != 0) {
                        for(int j=0; j<n; j++) {
if(i!=j && (mask & (1<<j))!=0) {
dp[mask][i] = Math.min(dp[mask][i], dp[mask^(1<<i)][j] + distance[j][i]);
                                                }
                                        }
                                }
                        }
                }
                int minCost = INF;
                for(int i=1;i<n;i++) {
                        if(dp[(1<<n)-1][i] + distance[i][0] < minCost) {
                                minCost = dp[(1<<n)-1][i] + distance[i][0];
```

```
                    }
                }
                return minCost;
        }
}
```

```
Enter the number of cities: 5Enter the number of cities: 5Enter th
e distance between cities: ⏎Enter the distance between cities: ⏎0
20 30 10 110 20 30 10 1115 0 16 4 215 0 16 4 210 3 0 15 410 3
0 15 47 14 8 0 57 14 8 0 52 4 6 8 02 4 6 8
0Minimum cost to visit all cities: 25⏎Minimum cost to visit all
cities: 25⏎
```

13. Implementation of N Queen Problem using Backtracking technique.

```
public class nQueens {
    private int[] result;
    private boolean[] column, leftDiagonal, rightDiagonal;
    private int n;
    public nQueens(int n) {
        this.n = n;
        result = new int[n];
        column = new boolean[n];
        leftDiagonal = new boolean[2 * n - 1];
        rightDiagonal = new boolean[2 * n - 1];
    }
    public boolean solve() {
        return solveNQueens(0);
    }
    private boolean solveNQueens(int row) {
        if (row == n) {
            printSolution();
            return true;
        }
        for (int col = 0; col < n; col++) {
            if (isSafe(row, col)) {
                placeQueen(row, col);
                if (solveNQueens(row + 1)) return true;
                removeQueen(row, col);
            }
        }
        return false;
    }
    private boolean isSafe(int row, int col) {
        return !column[col] && !leftDiagonal[row - col + n - 1] && !rightDiagonal[row + col];
    }
    private void placeQueen(int row, int col) {
        result[row] = col;
```

```
        column[col] = leftDiagonal[row - col + n - 1] = rightDiagonal[row + col] = true;
    }
    private void removeQueen(int row, int col) {
        column[col] = leftDiagonal[row - col + n - 1] = rightDiagonal[row + col] = false;
    }
    private void printSolution() {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                System.out.print(result[i] == j ? "Q " : ". ");
            }
            System.out.println();
        }
        System.out.println();
    }
    public static void main(String[] args) {
        nQueens queens = new nQueens(6); // Change the value of n to solve for different
sizes of the board
        if (!queens.solve()) System.out.println("No solution exists");
    }
}
```

14. Implementation of SUM-SUBSET Problem

```
import java.util.Scanner;
public class Exam {
    static int count = 0;
    static void subset(int cs, int k, int r, int[] x, int[] w, int d) {
        x[k] = 1;
        int n = w.length;
        if (cs + w[k] == d) {
            count++;
            System.out.print("Solution " + count + ": {");
            for (int i = 0; i < n; i++) {
                if (x[i] == 1) System.out.print(w[i] + " ");
            }
            System.out.println("}");
        } else if (cs + w[k] + w[k + 1] <= d) {
            subset(cs + w[k], k + 1, r - w[k], x, w, d);
        }
        if (cs + r - w[k] >= d && cs + w[k + 1] <= d) {
            x[k] = 0;
            subset(cs, k + 1, r - w[k], x, w, d);
        }
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("No of elements in the set: ");
        int n = sc.nextInt();
        int[] w = new int[n];
```

```
        int[] x = new int[n];
        int sum = 0;
        System.out.print("Enter the elements: ");
        for (int i = 0; i < n; i++) {
            w[i] = sc.nextInt();
            sum += w[i];
        }
        System.out.print("Enter the desired sum: ");
        int d = sc.nextInt();
        System.out.println("Sum is: " + sum);
        subset(0, 0, sum, x, w, d);
    }
}
```

```
No of elements in the set: 4
Enter the elements: 4 7 10 15
Enter the desired sum: 22
Sum is: 36
Solution 1: {7 15 }
```

15. Implementation of Knapsack problem using Branch and Bound method

```
import java.util.*;

public class KnapsackBB {
    static class Item {
        int weight, value;
        Item(int weight, int value) {
            this.weight = weight;
            this.value = value;
        }
    }

    static class Node {
        int level, profit, weight;
        double bound;
        Node(int level, int profit, int weight, double bound) {
            this.level = level;
            this.profit = profit;
            this.weight = weight;
            this.bound = bound;
        }
    }

    static double bound(Node u, int n, int W, Item[] arr) {
        if (u.weight >= W) return 0;
        double profit_bound = u.profit;
        int j = u.level + 1;
```

```
    int totweight = u.weight;

    while ((j < n) && (totweight + arr[j].weight <= W)) {
        totweight += arr[j].weight;
        profit_bound += arr[j].value;
        j++;
    }

    if (j < n) profit_bound += (W - totweight) * ((double) arr[j].value / arr[j].weight);

    return profit_bound;
}

static int knapsack(int W, Item[] arr, int n) {
    Arrays.sort(arr, (a, b) -> (b.value / b.weight) - (a.value / a.weight));
    Queue<Node> Q = new LinkedList<>();
    Node u = new Node(-1, 0, 0, 0), v = new Node(-1, 0, 0, 0);
    Q.add(u);
    int maxProfit = 0;

    while (!Q.isEmpty()) {
        u = Q.poll();
        if (u.level == -1) v.level = 0;
        if (u.level == n - 1) continue;

        v.level = u.level + 1;
        v.weight = u.weight + arr[v.level].weight;
        v.profit = u.profit + arr[v.level].value;
        v.bound = bound(v, n, W, arr);

        if (v.weight <= W && v.profit > maxProfit) maxProfit = v.profit;
        if (v.bound > maxProfit) Q.add(new Node(v.level, v.profit, v.weight, v.bound));

        v.weight = u.weight;
        v.profit = u.profit;
        v.bound = bound(v, n, W, arr);

        if (v.bound > maxProfit) Q.add(new Node(v.level, v.profit, v.weight, v.bound));
    }
    return maxProfit;
}

public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter number of items: ");
    int n = sc.nextInt();
    Item[] arr = new Item[n];
    System.out.print("Enter maximum weight: ");
```

```
        int W = sc.nextInt();
        System.out.println("Enter weight and value of each item:");
        for (int i = 0; i < n; i++) arr[i] = new Item(sc.nextInt(), sc.nextInt());
        System.out.println("Maximum profit: " + knapsack(W, arr, n));
    }
}
```

Enter number of items: 4
Enter maximum weight: 50
Enter weight and value of each item:
10 60
20 100
30 120
40 200
Maximum profit: 380