```
""StrassensMatrixMultiplication""
import java.util.*;
public class StrassensMatrixMultiplication
{
 public static void main(String[] args)
 {
  Scanner sc = new Scanner(System.in);
  System.out.println("Enter elements 1 row:");
  int[][] A = new int[2][2];
  for (int i = 0; i < 2; i++)
  {
   for (int j = 0; j < 2; j++)
   {
    A[i][j] = sc.nextInt();
   }
  }
  System.out.println("En elements 2 row:");
  int[][] B = new int[2][2];
  for (int i = 0; i < 2; i++)
  {
   for (int j = 0; j < 2; j++)
   {
    B[i][j] = sc.nextInt();
   }
  }
  int M1 = (A[0][0] + A[1][1]) * (B[0][0] + B[1][1]);
  int M2 = (A[1][0] + A[1][1]) * B[0][0];
  int M3 = A[0][0] * (B[0][1] - B[1][1]);
  int M4 = A[1][1] * (B[1][0] - B[0][0]);
  int M5 = (A[0][0] + A[0][1]) * B[1][1];
  int M6 = (A[1][0] - A[0][0]) * (B[0][0] + B[0][1]);
  int M7 = (A[0][1] - A[1][1]) * (B[1][0] + B[1][1]);
  int[][] C = new int[2][2];
  C[0][0] = M1 + M4 - M5 + M7;
  C[0][1] = M3 + M5;
  C[1][0] = M2 + M4;
  C[1][1] = M1 - M2 + M3 + M6;
  System.out.println("Result:");
  for (int i = 0; i < 2; i++)
  {
   for (int j = 0; j < 2; j++)
   {
    System.out.print(C[i][j] + " ");
   }
   System.out.println();
  }
  sc.close();
 }
}
""BinarySearchTree""
import java.util.*;
class BinarySearchTree
{
 static class Node
 {
```

```
 int data;
 Node left, right;
 public Node(int item)
 {
  data = item;
  left = right = null;
 }
}
Node root;
void insert(int key)
{
 root = insertRec(root, key);
}
Node insertRec(Node root, int key)
{
 if (root == null) return new Node(key);
 if (key < root.data) root.left = insertRec(root.left, key);
 else if (key > root.data) root.right = insertRec(root.right, key);
 return root;
}
boolean search(int key)
{
 return searchRec(root, key);
}
boolean searchRec(Node root, int key)
{
 if (root == null) return false;
 if (root.data == key) return true;
 return key < root.data ?
        searchRec(root.left, key) :
        searchRec(root.right, key);
}
void inorder()
{
 inorderRec(root);
}
void inorderRec(Node root)
{
 if (root != null)
 {
  inorderRec(root.left);
  System.out.print(root.data + " ");
  inorderRec(root.right);
 }
}
public static void main(String[] args)
{
 Scanner scanner=new Scanner(System.in);
 BinarySearchTree tree=new BinarySearchTree();
 System.out.println("Enter the no.of elements to insert:");
 int n = scanner.nextInt();
 System.out.println("Enter the elements:");
 for (int i = 0; i < n; i++)
 {
  tree.insert(scanner.nextInt());
```

```
  }
  System.out.println("\nIn-order traversal:");
  tree.inorder();
  System.out.println("\nEnter a number to search:");
  int search = scanner.nextInt();
  System.out.println(tree.search(search) ? "Found" : "Not Found");
  scanner.close();
 }
}
""JobSequencing""
import java.util.*;
public class JobSequencing
{
 static class Job
 {
  int id, deadline, profit;
  Job(int id, int deadline, int profit)
  {
   this.id = id;
   this.deadline = deadline;
   this.profit = profit;
  }
 }
 public static void main(String[] args)
 {
  Scanner sc = new Scanner(System.in);
  System.out.print("Enter the number of jobs: ");
  int n = sc.nextInt();
  Job[] jobs = new Job[n];
  for (int i = 0; i < n; i++)
  {
   System.out.print("Enter Job " + (i + 1) + " ID, Deadline, Profit: ");
   int id = sc.nextInt();
   int deadline = sc.nextInt();
   int profit = sc.nextInt();
   jobs[i] = new Job(id, deadline, profit);
  }
  Arrays.sort(jobs, (a, b)->b.profit-a.profit);
  boolean[] slot = new boolean[n];
  Arrays.fill(slot, false);
  int[] result = new int[n];
  for (int i = 0; i < n; i++)
  {
   for (int j = Math.min(n, jobs[i].deadline)
               - 1; j >= 0; j--)
   {
    if (!slot[j])
    {
     result[j] = jobs[i].id;
     slot[j] = true;
     break;
    }
   }
  }
  System.out.print("Job Sequence: ");
```

```
  for (int i = 0; i < n; i++)
  {
   if (slot[i])
   {
    System.out.print(result[i] + " ");
   }
  }
  int totalProfit = 0;
  for (int i = 0; i < n; i++)
  {
   if (slot[i])
   {
    totalProfit += jobs[i].profit;
   }
  }
  System.out.println("\nTotal Profit: " + totalProfit);
  sc.close();
 }
}
""DijkstraAlgorithm""
import java.util.Scanner;
import java.util.PriorityQueue;
public class Dijkstra
{
 public static void main(String[] args)
 {
  Scanner sc = new Scanner(System.in);
  int V = sc.nextInt();
  int E = sc.nextInt();
  int[][] graph = new int[V][V];
  for (int i = 0; i < E; i++)
  {
   int u = sc.nextInt(), v=sc.nextInt(),w=sc.nextInt();
   graph[u][v] = w;
   graph[v][u] = w;
  }
  int source = sc.nextInt();
  int[] dist = new int[V];
  for (int i = 0; i < V; i++) dist[i]=Integer.MAX_VALUE;
  dist[source] = 0;
  PriorityQueue<int[]> pq=new PriorityQueue<>((a, b) -> a[1] - b[1]);
  pq.offer(new int[] {source, 0});
  while (!pq.isEmpty())
  {
   int[] u = pq.poll();
   int uVertex = u[0];
   for (int v = 0; v < V; v++)
   {
    if (graph[uVertex][v] != 0 && dist[uVertex] + graph[uVertex][v] < dist[v])
    {
     dist[v] = dist[uVertex] + graph[uVertex][v];
     pq.offer(new int[] {v, dist[v]});
    }
   }
  }
```

```
  for (int i = 0; i < V; i++)
  {
   System.out.println("Vertex " + i + ": " + (dist[i] == Integer.MAX_VALUE ? "INF" : dist[i]));
  }
  sc.close();
 }
}
""BellmanFord""
import java.util.*;
public class BellmanFord
{
 public static void main(String[] args)
 {
  Scanner sc = new Scanner(System.in);
  System.out.print("Enter number of vertices: ");
  int V = sc.nextInt();
  System.out.print("Enter number of edges: ");
  int E = sc.nextInt();
  int[][] edges = new int[E][3];
  System.out.println("Enter edges (start vertex, end ver, weight):");
  for (int i = 0; i < E; i++) {
   edges[i][0] = sc.nextInt();
   edges[i][1] = sc.nextInt();
   edges[i][2] = sc.nextInt();
  }
  int[] dist = new int[V];
  for (int i = 0; i < V; i++)
  {
   dist[i] = Integer.MAX_VALUE;
  }
  dist[0] = 0;
  for (int i = 1; i < V; i++)
  {
   for (int j = 0; j < E; j++)
   {
    int u = edges[j][0];
    int v = edges[j][1];
    int weight = edges[j][2];
    if (dist[u] != Integer.MAX_VALUE && dist[u] + weight < dist[v])
    {
     dist[v] = dist[u] + weight;
    }
   }
  }
  System.out.println("Shortest distances from source vertex 0:");
  for (int i = 0; i < V; i++)
  {
   if (dist[i] == Integer.MAX_VALUE)
   {
    System.out.println("Vertex " + i + ": INF");
   } else
   {
    System.out.println("Vertex " + i + ": " + dist[i]);
   }
  }
```

```
  sc.close();
 }
}
""MultiStageGraph""
import java.util.Scanner;
import java.util.ArrayList;
public class MultiStageGraph {
 public static void main(String[] args) {
  Scanner sc = new Scanner(System.in);
  int n = sc.nextInt(), m = sc.nextInt();
  int[][] graph = new int[n][n];
  for (int i = 0; i < m; i++) graph[sc.nextInt()][sc.nextInt()] = sc.nextInt();
  int[] dist = new int[n];
  int[] next = new int[n];
  dist[n - 1] = 0;
  for (int i = n - 2; i >= 0; i--) {
   dist[i] = Integer.MAX_VALUE;
   for (int j = 0; j < n; j++) {
    if (graph[i][j] > 0 && dist[i] > graph[i][j] + dist[j]) {
     dist[i] = graph[i][j] + dist[j];
     next[i] = j;
    }
   }
  }
  System.out.println("Shortest path distances from start to end:");
  for (int d : dist) System.out.print(d + " ");
  System.out.println();
  System.out.println("Shortest path route:");
  ArrayList<Integer> path = new ArrayList<>();
  for (int i = 0; i != n - 1; i = next[i]) path.add(i);
  path.add(n - 1);
  System.out.println(path);
  sc.close();
 }
}
""AllPairsShortestPath""
import java.util.*;
public class FloydWarshall
{
 public static void main(String[] args)
 {
  Scanner sc = new Scanner(System.in);
  int n = sc.nextInt();
  int[][] dist = new int[n][n];
  for(int i = 0; i < n; i++) for(int j = 0; j < n; j++) dist[i][j] =
        (i == j) ? 0 : Integer.MAX_VALUE;
  System.out.print("Enter number of edges: ");
  int m = sc.nextInt();
  System.out.println("Enter edges in the format (u v weight):");
  for(int i = 0; i < m; i++)
  {
   int u = sc.nextInt(), v = sc.nextInt(), w = sc.nextInt();
   dist[u][v] = w;
  }
  for(int k = 0; k < n; k++)
```

```
   for(int i = 0; i < n; i++)
    for(int j = 0; j < n; j++)
     if(dist[i][k] != Integer.MAX_VALUE && dist[k][j]
           != Integer.MAX_VALUE)
       dist[i][j] = Math.min(dist[i][j], dist[i][k] + dist[k][j]);
  for(int i = 0; i < n; i++)
  {
   for(int j = 0; j < n; j++)
   {
    if(dist[i][j] == Integer.MAX_VALUE) System.out.print("INF ");
    else System.out.print(dist[i][j] + " ");
   }
   System.out.println();
  }
  sc.close();
 }
}
""MergeSort""
import java.util.*;
public class MergeSort {
 public static void mergeSort(int[] arr)
 {
  if (arr.length < 2) return;
  int mid = arr.length / 2;
  int[] left = new int[mid];
  int[] right = new int[arr.length - mid];
  System.arraycopy(arr, 0, left, 0, mid);
  System.arraycopy(arr, mid, right, 0, arr.length - mid);
  mergeSort(left);
  mergeSort(right);
  merge(arr, left, right);
 }
 private static void merge(int[] arr, int[] left, int[] right)
 {
  int i = 0, j = 0, k = 0;
  while (i < left.length && j < right.length)
  {
   arr[k++] = left[i] < right[j] ? left[i++] : right[j++];
  }
  while (i < left.length) arr[k++] = left[i++];
  while (j < right.length) arr[k++] = right[j++];
 }
 public static void main(String[] args)
 {
  Scanner scanner = new Scanner(System.in);
  System.out.print("Enter number of elements:");
  int n = scanner.nextInt();
  int[] arr = new int[n];
  System.out.print("Enter the elements:");
  for (int i = 0; i < n; i++) arr[i] = scanner.nextInt();
  mergeSort(arr);
  System.out.println("Sorted array:");
  for (int num : arr) System.out.print(num + " ");
  scanner.close();
 }
```

```
}
""Quicksort""
import java.util.*;
public class QuickSort
{
 public static void quickSort(int[] arr, int low, int high)
 {
  if (low < high)
  {
   int pivot = partition(arr, low, high);
   quickSort(arr, low, pivot - 1);
   quickSort(arr, pivot + 1, high);
  }
 }
 private static int partition(int[] arr, int low, int high)
 {
  int pivot = arr[high], i = low - 1;
  for (int j = low; j < high; j++)
  {
   if (arr[j] < pivot) {
    i++;
    int temp = arr[i];
    arr[i] = arr[j];
    arr[i] = arr[j];
    arr[j] = temp;
   }
  }
  int temp = arr[i + 1];
  arr[i + 1] = arr[high];
  arr[high] = temp;
  return i + 1;
 }
 public static void main(String[] args)
 {
  Scanner scanner = new Scanner(System.in);
  System.out.println("Enter number of elements:");
  int n = scanner.nextInt();
  int[] arr = new int[n];
  System.out.println("Enter elements:");
  for (int i = 0; i < n; i++)
   arr[i] = scanner.nextInt();
  quickSort(arr, 0, n - 1);
  System.out.print("Sorted array:");
  for (int num : arr)
   System.out.print(num + " ");
  scanner.close();
 }
}
""Knapsack""
import java.util.Scanner;
public class Knapsack {
 public static int knapsack(int W, int[] weights,int[] values, int n)
 {
  int[][] dp = new int[n + 1][W + 1];
  for (int i = 1; i <= n; i++)
```

```
  {
   for (int w = 1; w <= W; w++)
   {
    dp[i][w] = (weights[i - 1] <= w)
              ? Math.max(values[i - 1] + dp[i - 1][w - weights[i - 1]], dp[i - 1][w])
              : dp[i - 1][w];
   }
  }
  return dp[n][W];
 }
 public static void main(String[] args) {
  Scanner sc = new Scanner(System.in);
  System.out.println("Enter number of items: ");
  int n = sc.nextInt();
  int[] weights = new int[n], values= new int[n];
  System.out.println("Enter weights: ");
  for (int i = 0; i < n; i++) {
   weights[i] = sc.nextInt();
  }
  System.out.println("Enter values: ");
  for (int i = 0; i < n; i++) {
   values[i] = sc.nextInt();
  }
  System.out.println("Enter capacity: ");
  int W = sc.nextInt();

  System.out.println("Max value Knapsack: "+ knapsack(W, weights, values, n));
  sc.close();
 }
}
""SubsetSum""
import java.util.*;
public class SubsetSum {
 public static boolean isSubsetSum(int[] arr, int n, int sum)
 {
  boolean[][] dp = new boolean[n + 1][sum + 1];
  for (int i = 0; i <= n; i++)
  {
   dp[i][0] = true;
  }
  for (int i = 1; i <= n; i++)
  {
   for (int j = 1; j <= sum; j++)
   {
    if (arr[i - 1] <= j) {
     dp[i][j] = dp[i - 1][j] || dp[i - 1][j - arr[i - 1]];
    } else {
     dp[i][j] = dp[i - 1][j];
    }
   }
  }
  return dp[n][sum];
 }
 public static void main(String[] args)
 {
```

```
  Scanner sc = new Scanner(System.in);
  System.out.println("Enter number of elements in array: ");
  int n = sc.nextInt();
  System.out.println("Enter the target sum: ");
  int sum = sc.nextInt();
  int[] arr = new int[n];
  System.out.println("Enter array elements:");
  for (int i = 0; i < n; i++)
  {
   arr[i] = sc.nextInt();
  }
  System.out.println(isSubsetSum(arr, n, sum)
                     ? "Subset with the given sum exists"
                     : "No subset with the given sum");
  sc.close();
 }
}
```