**//6. 0/1 Knapsack (Dynamic Programming)**

**public class KnapsackDP {**

**public static int knapsack(int[] weights, int[] values, int capacity) {**

**int n = weights.length;**

**int[][] dp = new int[n + 1][capacity + 1];**

**for (int i = 1; i <= n; i++) {**

**for (int w = 0; w <= capacity; w++) {**

**if (weights[i - 1] <= w) {**

**dp[i][w] = Math.max(dp[i - 1][w], values[i - 1] + dp[i -**

**1][w - weights[i - 1]]);**

**} else {**

**dp[i][w] = dp[i - 1][w];**

**}}}**

**return dp[n][capacity]; }**

**public static void main(String[] args) {**

**int[] values = {60, 100, 120};**

**int[] weights = {10, 20, 30};**

**int capacity = 50;**

**System.out.println("Maximum value in Knapsack: " + knapsack(weights,**

**values, capacity));**

**}**

**}**

**#LCS**

**public class LCS {**

**public static void main(String[] args) {**

**String s1 = "ABCBDAB";**

**String s2 = "BDCAB";**

**String lcs = findLCS(s1, s2);**

**System.out.println("Length of LCS: " + lcs.length());**

**System.out.println("LCS: " + lcs);}**

**public static String findLCS(String s1, String s2) {**

**int m = s1.length();**

**int n = s2.length();**

**int[][] dp = new int[m + 1][n + 1];**

**for (int i = 1; i <= m; i++) {**

**for (int j = 1; j <= n; j++) {**

**if (s1.charAt(i - 1) == s2.charAt(j - 1)) {**

**dp[i][j] = dp[i - 1][j - 1] + 1;**

**} else {**

**dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);**

**}}}**

**StringBuilder lcs = new StringBuilder();**

**int i = m, j = n;**

**while (i > 0 && j > 0) {**

**if (s1.charAt(i - 1) == s2.charAt(j - 1)) {**

**lcs.append(s1.charAt(i - 1));**

**i--;j--;**

**} else if (dp[i - 1][j] > dp[i][j - 1]) {**

**i--;**

**} else {j--;}}**

**return lcs.reverse().toString();**

**}}**

**#FLOYD WARSHELL**

**public class FloydWarshall2 {**

**static final int INF = 1000000;**

**public static void floydWarshall(int[][] graph, int V){**

**int[][] dist = new int[V][V];**

**for (int i = 0; i < V; i++) {**

**for (int j = 0; j < V; j++) {**

**dist[i][j] = graph[i][j];**

**}}**

**for (int k = 0; k < V; k++) {**

**for (int i = 0; i < V; i++) {**

**for (int j = 0; j < V; j++) {**

**if (dist[i][k] + dist[k][j] < dist[i][j]) {**

**dist[i][j] = dist[i][k] + dist[k][j];**

**}}}}**

**printSolution(dist, V);**

**}**

**private static void printSolution(int[][] dist, int V) {**

**System.out.println("Shortest distances between every pair of vertices:");**

**for (int i = 0; i < V; i++) {**

**for (int j = 0; j < V; j++) {**

**if (dist[i][j] == INF) {**

**System.out.print("INF ");**

**} else {**

**System.out.print(dist[i][j] + " ");**

**}}**

**System.out.println();**

**}}**

**public static void main(String[] args) {**

**int V = 5;**

**int[][] graph = {{0,5,INF,2,INF},**

**{INF,0,2,INF,INF},**

**{3,INF,0,INF,7},**

**{INF,INF,4,0,1},**

**{1,3,INF,INF,0}};**

**floydWarshall(graph, V);**

**}}**

**#MERGE SORT**

**public static void divide(int arr[], int si, int ei) { if(si >= ei) {**

**return;**

**}**

**int mid si+ (ei-si)/2;**

**divide(arr, si, mid);**

**divide(arr, mid+1, ei);**

**conquer(arr, si, mid, ei);**

**}**

**public static void conquer(int arr[], int si, int mid, int ei) {**

**int merged [] = new int [ei-si+1];**

**Merge Sort int idx1 si;**

**int idx2 mid+1;**

**int x = 0;**

**while(idx1 <= mid && idx2 <= ei) {**

**if(arr[idx1] <= arr[idx2]) {**

**merged [x++] = arr[idx1++];**

**}**

**else {**

**merged [x++] = arr[idx2++];**

**}**

**while(idx1 <= mid) {**

**merged [x++] = arr[idx1++];**

**}**

**while(idx2 <= ei) {**

**merged [x++] = arr[idx2++];**

**}**

**for(int i=0, j=si; i<merged.length; i++, j++) {**

**arr[j] = merged [i];**

**}**

**#QUICK SORT**

**import java.util.Arrays;**

**class GfG {**

**static int partition(int[] arr, int low, int high) {**

**int pivot = arr[high];**

**int i = low - 1;**

**for (int j = low; j <= high - 1; j++) {**

**if (arr[j] < pivot) {**

**i++;**

**swap(arr, i, j);**

**}**

**}**

**swap(arr, i + 1, high);**

**return i + 1;**

**}**

**static void swap(int[] arr, int i, int j) {**

**int temp = arr[i];**

**arr[i] = arr[j];**

**arr[j] = temp;**

**}**

**static void quickSort(int[] arr, int low, int high) {**

**if (low < high) {**

**int pi = partition(arr, low, high);**

**quickSort(arr, low, pi - 1);**

**quickSort(arr, pi + 1, high);**

**}**

**}**

**public static void main(String[] args) {**

**int[] arr = {10, 7, 8, 9, 1, 5};**

**int n = arr.length;**

**quickSort(arr, 0, n - 1);**

**for (int val : arr) {**

**System.out.print(val + " ");**

**}**

**}**

**#OPTIMAL BINARY SEARCH TREE**

**import java.util.Scanner;**

**public class OptimalBinarySearchTree {**

**public static void optimalBST(int[] keys, int[] freq, int n) {**

**int[][] cost = new int[n][n];**

**int[][] root = new int[n][n];**

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

**cost[i][i] = freq[i];**

**root[i][i] = i;}**

**for (int length = 2; length <= n; length++) {**

**for (int i = 0; i <= n - length; i++) {**

**int j = i + length - 1;**

**cost[i][j] = Integer.MAX\_VALUE;**

**int sumFreq = sum(freq, i, j);**

**for (int r = i; r <= j; r++) {**

**int leftCost = (r > i) ? cost[i][r - 1] : 0;**

**int rightCost = (r < j) ? cost[r + 1][j] : 0;**

**int totalCost = leftCost + rightCost + sumFreq;**

**if (totalCost < cost[i][j]) {**

**cost[i][j] = totalCost;**

**root[i][j] = r;}}}}**

**System.out.println("Optimal cost: " + cost[0][n - 1]);**

**System.out.println("Root of the optimal BST: " + keys[root[0][n - 1]]);**

**System.out.println("Tree structure:");**

**printOptimalBSTStructure(root, keys, 0, n - 1, "Root");**

**}**

**public static int sum(int[] freq, int i, int j) {**

**int sum = 0;**

**for (int k = i; k <= j; k++) {**

**sum += freq[k];**

**}return sum;}**

**public static void printOptimalBSTStructure(int[][] root, int[] keys, int i, int j, String relation) {**

**if (i > j) {return;}**

**int r = root[i][j];**

**System.out.println(relation + ": " + keys[r]);**

**printOptimalBSTStructure(root, keys, i, r - 1, "Left child of " + keys[r]);**

**printOptimalBSTStructure(root, keys, r + 1, j, "Right child of " + keys[r]);**

**}**

**public static void main(String[] args) {**

**Scanner scanner = new Scanner(System.in);**

**System.out.println("Enter number of keys:");**

**int n = scanner.nextInt();**

**int[] keys = new int[n];**

**int[] freq = new int[n];**

**System.out.println("Enter the keys:");**

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

**keys[i] = scanner.nextInt();**

**}**

**System.out.println("Enter the frequencies:");**

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

**freq[i] = scanner.nextInt();**

**}**

**optimalBST(keys, freq, n);**

**scanner.close();**

**}**

**}**