**1) 0-1 Knapsack Problem**

// Java program for the above approach

import java.util.\*;

class GFG {

static int knapSack(int W, int wt[], int val[], int n)

{

int[] dp = new int[W + 1];

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

for (int w = W; w >= 0; w--) {

if (wt[i - 1] <= w)

dp[w]

= Math.max(dp[w], dp[w - wt[i - 1]]

+ val[i - 1]);

}

}

return dp[W];

}

public static void main(String[] args)

{

int profit[] = { 60, 100, 120 };

int weight[] = { 10, 20, 30 };

int W = 50;

int n = profit.length;

System.out.print(knapSack(W, weight, profit, n));

}

}

Input:

profit[] = { 60, 100, 120 }

weight[] = { 10, 20, 30 }

W = 50

**Output**:

220

**Time Complexity**: O(2^N)  
**Auxiliary Space**: O(N)

**2) Floor in Sorted Array**

class FloorSearchProblem {

static int floorSearch(int arr[], int n, int x) {

if (x >= arr[n - 1])

return n - 1;

if (x < arr[0])

return -1;

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

if (arr[i] > x)

return (i - 1);

return -1;

}

public static void main(String[] args) {

int arr[] = { 1, 2, 4, 6, 10, 12, 14 };

int n = arr.length;

int x = 7;

int index = floorSearch(arr, n, x);

if (index == -1)

System.out.print("Floor of " + x + " doesn't exist in array ");

else

System.out.print("Floor of " + x + " is " + arr[index]);

}

}

**Input**:

arr = { 1, 2, 4, 6, 10, 12, 14 }

x = 7

**Output**:

Floor of 7 is 6

**Time Complexity**: O(N)  
**Auxiliary Space**: O(1)

**3) Check Equal Arrays**

class EqualArraysCheck {

public static boolean areEqual(int arr1[], int arr2[]) {

int N = arr1.length;

int M = arr2.length;

if (N != M)

return false;

Arrays.sort(arr1);

Arrays.sort(arr2);

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

if (arr1[i] != arr2[i])

return false;

return true;

}

public static void main(String[] args) {

int arr1[] = { 3, 5, 2, 5, 2 };

int arr2[] = { 2, 3, 5, 5, 2 };

if (areEqual(arr1, arr2))

System.out.println("Yes");

else

System.out.println("No");

}

}

**Input**:

arr1[] = { 3, 5, 2, 5, 2 }

arr2[] = { 2, 3, 5, 5, 2 }

**Output**:

Yes

**Time Complexity**: O(N log N)  
**Auxiliary Space**: O(1)

**4) Palindrome Linked List**

class PalindromeLinkedList {

static class Node {

int data;

Node next;

Node(int d) {

data = d;

next = null;

}

}

static Node reverseList(Node head) {

Node prev = null;

Node curr = head;

Node next;

while (curr != null) {

next = curr.next;

curr.next = prev;

prev = curr;

curr = next;

}

return prev;

}

static boolean isIdentical(Node n1, Node n2) {

while (n1 != null && n2 != null) {

if (n1.data != n2.data)

return false;

n1 = n1.next;

n2 = n2.next;

}

return true;

}

static boolean isPalindrome(Node head) {

if (head == null || head.next == null)

return true;

Node slow = head, fast = head;

while (fast.next != null && fast.next.next != null) {

slow = slow.next;

fast = fast.next.next;

}

Node head2 = reverseList(slow.next);

slow.next = null;

boolean ret = isIdentical(head, head2);

head2 = reverseList(head2);

slow.next = head2;

return ret;

}

public static void main(String[] args) {

Node head = new Node(1);

head.next = new Node(2);

head.next.next = new Node(3);

head.next.next.next = new Node(2);

head.next.next.next.next = new Node(1);

if (isPalindrome(head))

System.out.println("true");

else

System.out.println("false");

}

}

**Input**:

Linked list: 1 -> 2 -> 3 -> 2 -> 1

**Output**:

true

**Time Complexity**: O(N)  
**Auxiliary Space**: O(1)

**5) Balanced Tree Check**

java

Copy code

class BalancedTreeCheck {

static class Node {

int data;

Node left, right;

Node(int d) {

data = d;

left = right = null;

}

}

Node root;

boolean isBalanced(Node node) {

if (node == null)

return true;

int lh = height(node.left);

int rh = height(node.right);

if (Math.abs(lh - rh) <= 1 && isBalanced(node.left) && isBalanced(node.right))

return true;

return false;

}

int height(Node node) {

if (node == null)

return 0;

return 1 + Math.max(height(node.left), height(node.right));

}

public static void main(String args[]) {

BalancedTreeCheck tree = new BalancedTreeCheck();

tree.root = new Node(1);

tree.root.left = new Node(2);

tree.root.right = new Node(3);

tree.root.left.left = new Node(4);

tree.root.left.right = new Node(5);

tree.root.left.left.left = new Node(8);

if (tree.isBalanced(tree.root))

System.out.println("Tree is balanced");

else

System.out.println("Tree is not balanced");

}

}

**Input**:

[1, 2, 3, 4, 5, null, null, 8]

**Output**:

Tree is not balanced

**Time Complexity**: O(N^2)  
**Auxiliary Space**: O(N)

**6) Triplet Sum in Array**

class TripletSumArray {

static boolean find3Numbers(int[] arr, int sum) {

int n = arr.length;

for (int i = 0; i < n - 2; i++) {

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

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

if (arr[i] + arr[j] + arr[k] == sum) {

System.out.println("Triplet is " + arr[i] + ", " + arr[j] + ", " + arr[k]);

return true;

}

}

}

}

return false;

}

public static void main(String[] args) {

int[] arr = { 1, 4, 45, 6, 10, 8 };

int sum = 22;

find3Numbers(arr, sum);

}

}

**Input**:

arr[] = { 1, 4, 45, 6, 10, 8 }

sum = 22

**Output**:

Triplet is 4, 10, 8

**Time Complexity**: O(N^3)  
**Auxiliary Space**: O(1)