Name: BHARATH V

Dept: CSBS

0-1 Knapsack problem

```
public class Knapsack {
   static int knapSack(int W, int wt[], int val[], int n) {
      int[][] dp = new int[n + 1][W + 1];
      for (int i = 0; i <= n; i++) {
         for (int w = 0; w <= W; w++) {
            if (i == 0 || w == 0)
               dp[i][w] = 0;
            else if (wt[i - 1] <= w)
               dp[i][w] = Math.max(val[i - 1] + dp[i - 1][w - wt[i - 1]], dp[i - 1][w]);
            else
               dp[i][w] = dp[i - 1][w];
         }
      }

      return dp[n][W];
   }
}
```

Input

Profit: [60,100,120]

Weight: [10,20,30]

W=50

Output: 220

Complexity:

Time: O(n*w)

Space: O(n*w)

2.Floor in a Sorted Array

```
public class FloorInSortedArray {
   static int findFloor(int[] arr, int n, int x) {
      int low = 0, high = n - 1;
      int floorIndex = -1;
      while (low <= high) {
         int mid = low + (high - low) / 2;
         if (arr[mid] == x) {
            return mid;
         } else if (arr[mid] < x) {
            floorIndex = mid;
            low = mid + 1;
         } else {
            high = mid – 1;
         }
      }

      return floorIndex;
   }

   public static void main(String[] args) {
      int[] arr = {1, 2, 8, 10, 10, 12, 19};
```

```
        int x = 5
        int n = arr.length;
        int floorIndex = findFloor(arr, n, x);
        if (floorIndex == -1) {
            System.out.println(x);
        } else {
            System.out.println("Floor of " + x + " is " + arr[floorIndex] + " at index " +
floorIndex);
        }
    }
}
```

OUTPUT

2

Complexity

Time: O(log n)

Space: O(1)

3.Check Equal Arrays

```
import java.util.Arrays;
public class EqualArrays {
    static boolean areEqual(int[] arr1, int[] arr2) {

        if (arr1.length != arr2.length) {
            return false;
        }
        Arrays.sort(arr1);
```

```
        Arrays.sort(arr2);
        for (int i = 0; i < arr1.length; i++) {
            if (arr1[i] != arr2[i]) {
                return false;
            }
        }

        return true;
    }

    public static void main(String[] args) {
        int[] arr1 = {1, 2, 3, 4};
        int[] arr2 = {4, 3, 2, 1};

        if (areEqual(arr1, arr2)) {
            System.out.println("The arrays are equal.");
        } else {
            System.out.println("The arrays are not equal.");
        }
    }
}
```

OUTPUT

The arrays are equal

Complexity

Time: O(n logn)

Space: O(logn)

4. **Palindrome Linked List**

```
class ListNode {
   int val;
   ListNode next;

   ListNode(int val) {
      this.val = val;
      this.next = null;
   }
}

public class PalindromeLinkedList {
   public static boolean isPalindrome(ListNode head) {
      if (head == null || head.next == null) {
         return true;
      }
      ListNode slow = head, fast = head;
      while (fast != null && fast.next != null) {
         slow = slow.next;
         fast = fast.next.next;
      }
      ListNode secondHalfStart = reverseList(slow);
      ListNode firstHalfStart = head;
      ListNode secondHalfIter = secondHalfStart;
      boolean isPalindrome = true;

      while (secondHalfIter != null) {
         if (firstHalfStart.val != secondHalfIter.val) {
            isPalindrome = false;
            break;
         }
         firstHalfStart = firstHalfStart.next;
         secondHalfIter = secondHalfIter.next;
      }

      reverseList(secondHalfStart);

      return isPalindrome;
   }
```

```
    private static ListNode reverseList(ListNode head) {
        ListNode prev = null;
        while (head != null) {
            ListNode nextNode = head.next;
            head.next = prev;
            prev = head;
            head = nextNode;
        }
        return prev;
    }

    public static void main(String[] args) {
        ListNode head = new ListNode(1);
        head.next = new ListNode(2);
        head.next.next = new ListNode(2);
        head.next.next.next = new ListNode(1);

        if (isPalindrome(head)) {
            System.out.println("The linked list is a palindrome.");
        } else {
            System.out.println("The linked list is not a palindrome.");
        }
    }
}
```

OUTPUT:

The linked list is a palindrome

Complexity:

Time: O(n)

Space: O(1)

5.**Balanced Tree Check**

```
class TreeNode {
    int val;
    TreeNode left, right;
```

```
    TreeNode(int val) {
        this.val = val;
        this.left = this.right = null;
    }
}

public class BalancedTree {

    public static int isBalancedHelper(TreeNode root) {
        if (root == null) {
            return 0;
        }

        int leftHeight = isBalancedHelper(root.left);
        if (leftHeight == -1) {
            return -1;
        }

        int rightHeight = isBalancedHelper(root.right);
        if (rightHeight == -1) {
            return -1;
        }

        if (Math.abs(leftHeight - rightHeight) > 1) {
            return -1;
```

```
        }

        return Math.max(leftHeight, rightHeight) + 1;
    }

    public static boolean isBalanced(TreeNode root) {
        return isBalancedHelper(root) != -1;
    }

    public static void main(String[] args) {
        TreeNode root = new TreeNode(1);
        root.left = new TreeNode(2);
        root.right = new TreeNode(3);
        root.left.left = new TreeNode(4);
        root.left.right = new TreeNode(5);
        root.left.left.left = new TreeNode(6);

        System.out.println("Is the tree balanced? " + isBalanced(root));
    }
}
```

OUTPUT

Is the tree balanced? True

Complexity

Time: O(n)

Space: O(1)

6.**Triplet Sum Array**

```
import java.util.Arrays;
public class TripletSum {
    public static boolean findTriplet(int[] arr, int target) {
        Arrays.sort(arr);

        for (int i = 0; i < arr.length - 2; i++)
            if (i > 0 && arr[i] == arr[i - 1]) continue;
            int left = i + 1;
            int right = arr.length - 1;
            while (left < right) {
                int sum = arr[i] + arr[left] + arr[right];
                if (sum == target) {
                    System.out.println("Triplet: " + arr[i] + ", " + arr[left] + ", " +
arr[right]);
                    return true;
                }
                else if (sum < target) {
                    left++;
                }
                else {
                    right--;
                }
            }
        }
```

```
        return false;
    }

    public static void main(String[] args) {
        int[] arr = {12, 3, 4, 1, 6, 9};
        int target = 24;
        if (!findTriplet(arr, target)) {
            System.out.println("No triplet found with the target sum.");
        }
    }
}
```

OUTPUT

Triplet: 12,3,9

Complexity

Time: $O(n^2)$

Space: O(1)