PRACTICE SET – 2

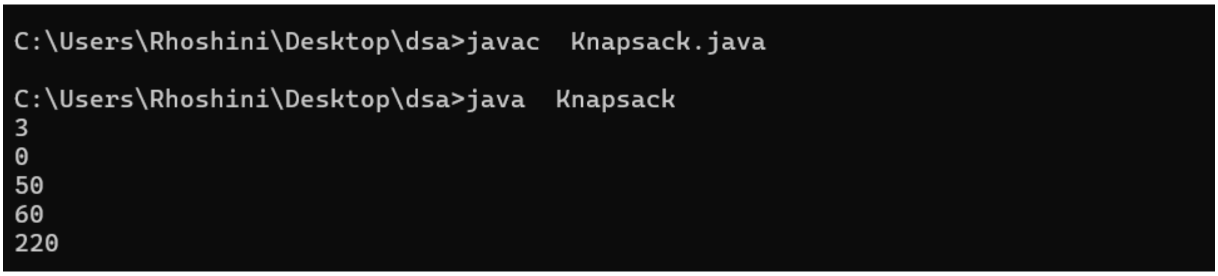
Meena Rhoshini C

Nov 11, 24

1. 0 - 1 Knapsack Problem

```
class Knapsack {
    public static void main(String[] args) {
        int capacity = 4;
        int[] val = {1, 2, 3};
        int[] wt = {4, 5, 1};
        System.out.println(knapsack(capacity, val, wt));
        int capacity1 = 3;
        int[] val1 = {1, 2, 3};
        int[] wt1 = {4, 5, 6};
        System.out.println(knapsack(capacity1, val1, wt1));
        int capacity2 = 5;
        int[] val2 = {10, 40, 30, 50};
        int[] wt2 = {5, 4, 6, 3};
        System.out.println(knapsack(capacity2, val2, wt2));
        int capacity3 = 10;
        int[] val3 = {60, 100, 120};
        int[] wt3 = {10, 20, 30};
        System.out.println(knapsack(capacity3, val3, wt3));
        int capacity4 = 50;
        int[] val4 = {60, 100, 120};
        int[] wt4 = {10, 20, 30};
        System.out.println(knapsack(capacity4, val4, wt4));
    }
    static int knapsack(int capacity, int[] val, int[] wt) {
        int n = val.length;
        int[][] dp = new int[n + 1][capacity + 1];
        for (int i = 1; i <= n; i++) {
            for (int w = 1; w <= capacity; w++) {
                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][capacity];
    }
}
```

OUTPUT:



```
C:\Users\Rhoshini\Desktop\dsa>javac  Knapsack.java

C:\Users\Rhoshini\Desktop\dsa>java  Knapsack
3
0
50
60
220
```

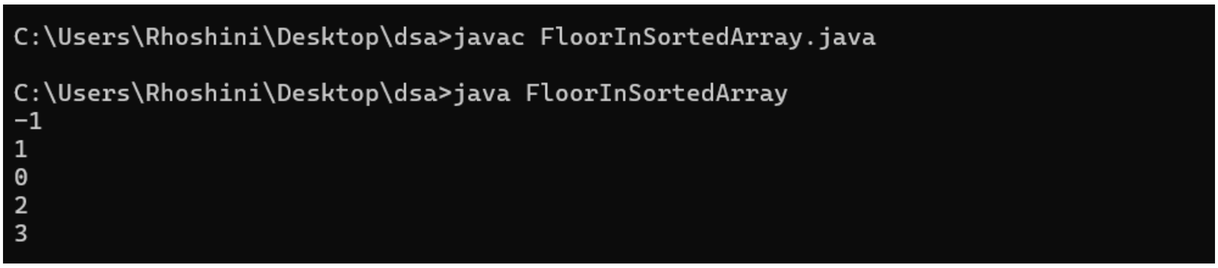
Time complexity: O(n * capacity)

Space complexity: O(n * capacity)

2. Floor in a Sorted Array

```
class FloorInSortedArray {
    public static void main(String[] args) {
        int[] arr = {1, 2, 8, 10, 11, 12, 19};
        int k = 0;
        System.out.println(floor(arr, k));
        int[] arr1 = {1, 2, 8, 10, 11, 12, 19};
        int k1 = 5;
        System.out.println(floor(arr1, k1));
        int[] arr2 = {1, 2, 8};
        int k2 = 1;
        System.out.println(floor(arr2, k2));
        int[] arr3 = {2, 4, 6, 8, 10};
        int k3 = 7;
        System.out.println(floor(arr3, k3));
        int[] arr4 = {5, 10, 15, 20};
        int k4 = 25;
        System.out.println(floor(arr4, k4));
    }
    static int floor(int[] arr, int k) {
        int low = 0, high = arr.length - 1, result = -1;
        while (low <= high) {
            int mid = low + (high - low) / 2;
            if (arr[mid] == k) return mid;
            if (arr[mid] < k) {
                result = mid;
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }
        return result;
    }
}
```

OUTPUT:



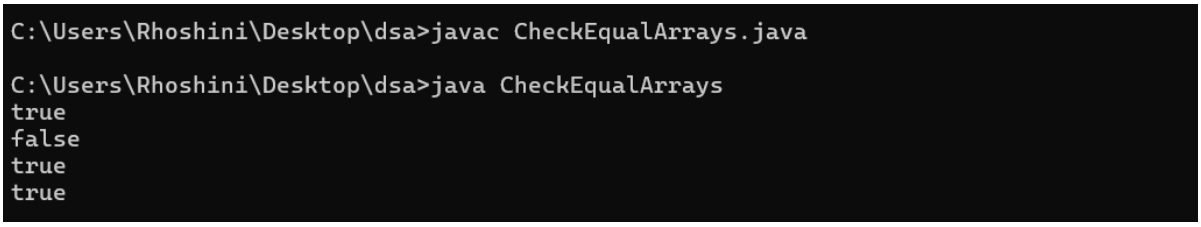
```
C:\Users\Rhoshini\Desktop\dsa>javac FloorInSortedArray.java

C:\Users\Rhoshini\Desktop\dsa>java FloorInSortedArray
-1
1
0
2
3
```

Time Complexity: O(log n)
Space Complexity: O(1)

3. Check Equal Arrays

```
class CheckEqualArrays {
    public static void main(String[] args) {
        int[] arr1 = {1, 2, 5, 4, 0};
        int[] arr2 = {2, 4, 5, 0, 1};
        System.out.println(equal(arr1, arr2));
        int[] arr3 = {1, 2, 5};
        int[] arr4 = {2, 4, 15};
        System.out.println(equal(arr3, arr4));
        int[] arr5 = {3, 4, 2, 1};
        int[] arr6 = {1, 2, 3, 4};
        System.out.println(equal(arr5, arr6));
        int[] arr7 = {10, 20, 30};
        int[] arr8 = {20, 30, 10};
        System.out.println(equal(arr7, arr8));
    }
    static boolean equal(int[] arr1, int[] arr2) {
        if (arr1.length != arr2.length) return false;
        int[] count = new int[1000000];
        for (int num : arr1) count[num]++;
        for (int num : arr2) {
            if (count[num] == 0) return false;
            count[num]--;
        }
        return true;
    }
}
```

OUTPUT:



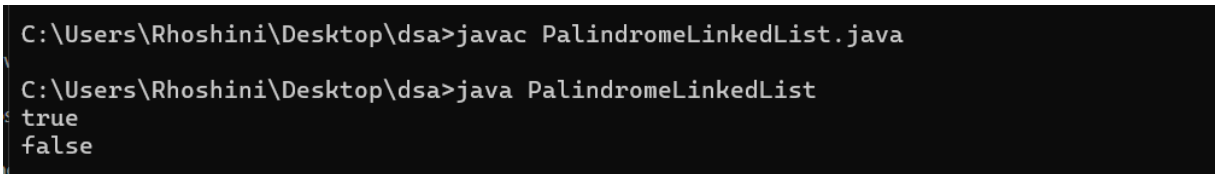
```
C:\Users\Rhoshini\Desktop\dsa>javac CheckEqualArrays.java

C:\Users\Rhoshini\Desktop\dsa>java CheckEqualArrays
true
false
true
true
```

Time Complexity: O(n)
Space Complexity: O(n)

4. Palindrome Linked List

```
class PalindromeLinkedList {
    static class ListNode {
        int val;
        ListNode next;
        ListNode(int val) {
            this.val = val;
            this.next = null;
        }
    }
    public static void main(String[] args) {
        ListNode head1 = new ListNode(1);
        head1.next = new ListNode(2);
        head1.next.next = new ListNode(1);
        head1.next.next.next = new ListNode(1);
        head1.next.next.next.next = new ListNode(2);
        head1.next.next.next.next.next = new ListNode(1);
        System.out.println(isPalindrome(head1));
        ListNode head2 = new ListNode(1);
        head2.next = new ListNode(2);
        head2.next.next = new ListNode(3);
        head2.next.next.next = new ListNode(4);
        System.out.println(isPalindrome(head2));
    }
    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;
        }
        slow = reverse(slow);
        fast = head;
        while (slow != null) {
            if (slow.val != fast.val) return false;
            slow = slow.next;
            fast = fast.next;
        }
        return true;
    }
    static ListNode reverse(ListNode head) {
        ListNode prev = null, curr = head;
        while (curr != null) {
            ListNode next = curr.next;
            curr.next = prev;
            prev = curr;
            curr = next;
        }
        return prev;
    }
}
```

OUTPUT:



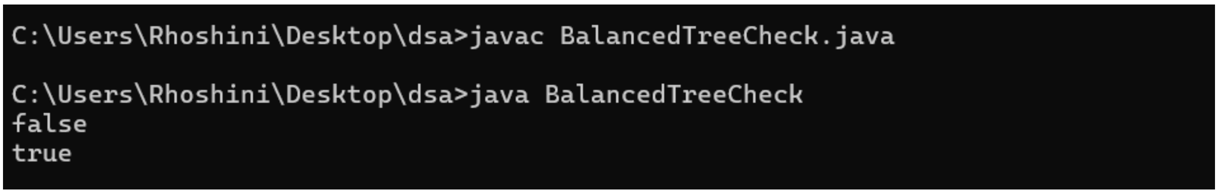
```
C:\Users\Rhoshini\Desktop\dsa>javac PalindromeLinkedList.java

C:\Users\Rhoshini\Desktop\dsa>java PalindromeLinkedList
true
false
```

Time Complexity: O(n)
Space Complexity: O(1)

5. Balanced Tree Check

```
class BalancedTreeCheck {
    static class TreeNode {
        int val;
        TreeNode left, right;
        TreeNode(int val) {
            this.val = val;
            left = right = null;
        }
    }
    public static void main(String[] args) {
        TreeNode root1 = new TreeNode(1);
        root1.left = new TreeNode(2);
        root1.left.right = new TreeNode(3);
        System.out.println(isBalanced(root1));
        TreeNode root2 = new TreeNode(10);
        root2.left = new TreeNode(20);
        root2.right = new TreeNode(30);
        root2.left.left = new TreeNode(40);
        root2.left.right = new TreeNode(60);
        System.out.println(isBalanced(root2));
    }
    static boolean isBalanced(TreeNode root) {
        return height(root) != -1;
    }
    static int height(TreeNode root) {
        if (root == null) return 0;
        int leftHeight = height(root.left);
        if (leftHeight == -1) return -1;
        int rightHeight = height(root.right);
        if (rightHeight == -1) return -1;
        if (Math.abs(leftHeight - rightHeight) > 1) return -1;
        return Math.max(leftHeight, rightHeight) + 1;
    }
}
```

OUTPUT:



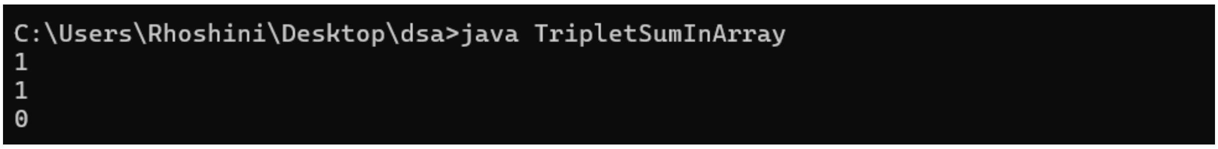
```
C:\Users\Rhoshini\Desktop\dsa>javac BalancedTreeCheck.java

C:\Users\Rhoshini\Desktop\dsa>java BalancedTreeCheck
false
true
```

Time Complexity: O(n)
Space Complexity: O(h)

6. **Triplet Sum in Array**

```
class TripletSumInArray {
    public static void main(String[] args) {
        int n1 = 6, x1 = 13;
        int[] arr1 = {1, 4, 45, 6, 10, 8};
        System.out.println(tripletSum(arr1, n1, x1));
        int n2 = 6, x2 = 10;
        int[] arr2 = {1, 2, 4, 3, 6, 7};
        System.out.println(tripletSum(arr2, n2, x2));
        int n3 = 6, x3 = 24;
        int[] arr3 = {40, 20, 10, 3, 6, 7};
        System.out.println(tripletSum(arr3, n3, x3));
    }
    static int tripletSum(int[] arr, int n, int x) {
        for (int i = 0; i < n - 2; i++) {
            int left = i + 1, right = n - 1;
            while (left < right) {
                int sum = arr[i] + arr[left] + arr[right];
                if (sum == x) return 1;
                else if (sum < x) left++;
                else right--;
            }
        }
        return 0;
    }
}
```

OUTPUT:



```
C:\Users\Rhoshini\Desktop\dsa>java TripletSumInArray
1
1
0
```

Time Complexity: O(n^2)
Space Complexity: O(1)