

Date: 11/11/24

DSA Practice Problems

1. Floor in Sorted Array

Code:

```
public class FloorInSortedArray {  
  
    public static int findFloor(int[] arr, int k) {  
  
        int left = 0, right = arr.length - 1, result = -1;  
  
        while (left <= right) {  
  
            int mid = left + (right - left) / 2;  
  
            if (arr[mid] <= k) {  
  
                result = mid;  
  
                left = mid + 1;  
  
            } else {  
  
                right = mid - 1;  
  
            }  
  
        }  
  
        return result;  
  
    }  
  
  
    public static void main(String[] args) {  
  
        int[] arr = {1, 2, 8, 10, 11, 12, 19};  
  
        int k = 5;  
  
        System.out.println(findFloor(arr, k));  
  
    }  
  
}
```

Output:

1
PS C:\Users\Sandiipnish P

Time Complexity: $O(n)$

2. 0-1 Knapsack Problem

Code:

```
public class Knapsack {  
  
    public static int knapsack(int[] val, int[] wt, int capacity) {  
  
        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(dp[i - 1][w], val[i - 1] + dp[i - 1][w - wt[i - 1]]);  
  
                } else {  
  
                    dp[i][w] = dp[i - 1][w];  
  
                }  
  
            }  
  
        }  
  
        return dp[n][capacity];  
  
    }  
  
    public static void main(String[] args) {  
  
        int[] val = {10, 40, 30, 50};  
  
        int[] wt = {5, 4, 6, 3};  
  
        int capacity = 5;  
  
        System.out.println(knapsack(val, wt, capacity));  
  
    }  
}
```

```
}
```

Output:



```
50
PS C:\Users\Sandiipnish P
```

Time Complexity: $O(n)$

3. Check Equal Arrays

Code:

```
import java.util.HashMap;
```

```
public class CheckEqualArrays{

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

        if (arr1.length != arr2.length) return false;

        HashMap<Integer, Integer> freqMap = new HashMap<>();

        for (int num : arr1) freqMap.put(num, freqMap.getOrDefault(num, 0) + 1);

        for (int num : arr2) {

            if (!freqMap.containsKey(num)) return false;

            freqMap.put(num, freqMap.get(num) - 1);

            if (freqMap.get(num) == 0) freqMap.remove(num);

        }

        return freqMap.isEmpty();

    }

    public static void main(String[] args) {

        int[] arr1 = {1, 2, 5, 4, 0};
```

```

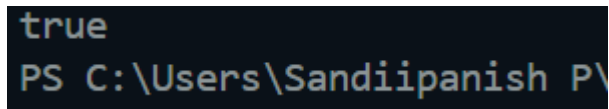
        int[] arr2 = {2, 4, 5, 0, 1};

        System.out.println(areEqual(arr1, arr2));

    }
}

```

Output:



```

true
PS C:\Users\Sandiipnish P\

```

Time Complexity: $O(n)$

4. Palindrome linked list

Code:

```

class ListNode {

    int data;

    ListNode next;

    ListNode(int data) {

        this.data = data;

        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 secondHalf = reverse(slow);
```

```
ListNode firstHalf = head;
```

```
boolean isPalindrome = true;
```

```
while (secondHalf != null) {
```

```
    if (firstHalf.data != secondHalf.data) {
```

```
        isPalindrome = false;
```

```
        break;
```

```
    }
```

```
    firstHalf = firstHalf.next;
```

```
    secondHalf = secondHalf.next;
```

```
}
```

```
reverse(slow);
```

```
return isPalindrome;
```

```
}
```

```
private static ListNode reverse(ListNode head) {
```

```
    ListNode prev = null;
```

```
    while (head != null) {
```

```
        ListNode next = head.next;
```

```
        head.next = prev;
```

```
        prev = head;
```

```
        head = next;
```

```
}
```

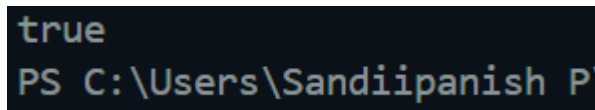
```

        return prev;
    }

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

```

Output:



```

true
PS C:\Users\Sandiipnish P

```

Time Complexity: $O(n)$

5. Balanced Tree Check

Code:

```

class TreeNode {
    int data;
    TreeNode left, right;
    TreeNode(int data) {
        this.data = data;
        left = right = null;
    }
}

```

```
public class BalancedBinaryTree {

    public static boolean isBalanced(TreeNode root) {

        return checkHeight(root) != -1;

    }

    private static int checkHeight(TreeNode node) {

        if (node == null) return 0;

        int leftHeight = checkHeight(node.left);

        if (leftHeight == -1) return -1;

        int rightHeight = checkHeight(node.right);

        if (rightHeight == -1) return -1;

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

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

    }

    public static void main(String[] args) {

        TreeNode root = new TreeNode(10);

        root.left = new TreeNode(20);

        root.right = new TreeNode(30);

        root.left.left = new TreeNode(40);

        root.left.right = new TreeNode(60);

    }

}
```

```
        System.out.println(isBalanced(root) ? 1 : 0);
    }
}
```

Output:



```
1
PS C:\Users\Sandiipanish P
```

Time Complexity: $O(n)$

6. Triplet Sum in Array

Code:

```
import java.util.Arrays;
```

```
public class TripletSum {

    public static int find3Numbers(int[] arr, int n, int x) {

        Arrays.sort(arr);

        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;

    }

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



```
int[] arr = {1, 4, 45, 6, 10, 8};  
  
int x = 13;  
  
System.out.println(find3Numbers(arr, arr.length, x));  
  
}  
  
}
```

Output:

A screenshot of a terminal window with a dark background. The first line shows the number '1' in a light blue font. The second line shows the command prompt path 'PS C:\Users\Sandiipanih P' in a light blue font.

```
1  
PS C:\Users\Sandiipanih P
```

Time Complexity: $O(n^2)$