CATHERINE JESSIE J
22CS035 - CSE

1) **0-1 Knapsack Problem**

```
class Solution {
    static int knapSack(int capacity, int val[], int wt[]) {
        int n = val.length;
        int[] dp = new int[capacity + 1];

        for (int i = 0; i < n; i++) {
            for (int j = capacity; j >= wt[i]; j--) {
                dp[j] = Math.max(dp[j], val[i] + dp[j - wt[i]]);
            }
        }

        return dp[capacity];
    }
}
```

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

2) **Floor in Sorted Array**

```
class Solution {
    static int findFloor(int[] arr, int k) {
        int f = -1;
        int mf = Integer.MIN_VALUE;

        for (int i = 0; i < arr.length; i++) {
            if (arr[i] <= k && arr[i] > mf) {
                mf = arr[i];
                f = i;
            }
        }

        return f;
    }

}
```

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

3) **Check Equal Arrays**

```
class Solution {
    public static boolean check(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) || freqMap.get(num) == 0) {
                return false;
            }
            freqMap.put(num, freqMap.get(num) - 1);
        }

        for (int count : freqMap.values()) {
            if (count != 0) {
                return false;
            }
        }

        return true;
    }
}
```

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

4) **Palindrome Linked List**

```
class Node
{
    int data;
    Node next;

    Node(int d)
    {
        data = d;
        next = null;
    }
}
```

```
class Solution {

    boolean isPalindrome(Node head) {
      if (head == null || head.next == null) {
          return true;
       }

       Node slow = head, fast = head;

       while (fast != null && fast.next != null) {
           slow = slow.next;
           fast = fast.next.next;
       }

       Node prev = null;
       Node curr = slow;
       while (curr != null) {
           Node nextTemp = curr.next;
           curr.next = prev;
           prev = curr;
           curr = nextTemp;
       }

       Node fHalf = head;
       Node sHalf = prev;

       while (sHalf != null) {
           if (fHalf.data != sHalf.data) {
               return false;
           }
           fHalf = fHalf.next;
           sHalf = sHalf.next;
       }

       return true;
    }
}
```

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

5) Balanced Tree Check

```
class Tree
{
        boolean isBalanced(Node r) {
        if (r == null) {
            return true;
        }

        int lh = getHeight(r.left);
        int rh = getHeight(r.right);

        if (Math.abs(lh - rh) > 1) {
            return false;
        }

        return isBalanced(r.left) && isBalanced(r.right);
    }

    private int getHeight(Node r) {
        if (r == null) {
            return 0;
        }

        int lh = getHeight(r.left);
        int rh = getHeight(r.right);

        return Math.max(lh, rh) + 1;
    }

}
```

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

6) Triplet Sum in Array

```
class Solution {

    public static boolean find3Numbers(int arr[], int n, int x) {
        Arrays.sort(arr);

        for (int i = 0; i < n - 2; i++) {
            int l = i + 1;
            int r = n - 1;

            while (l < r) {
                int s = arr[i] + arr[l] + arr[r];
                if (s == x) {
                    return true;
                } else if (s < x) {
                    l++;
                } else {
                    r--;
                }
            }
        }

        return false;
    }
}
```

Time Complexity: O(n^3)
Auxiliary Space: O(1)