Day - 1 : 11/11/2024

1. **Linked List Palindrome:**

```
class Solution {
public:
    bool isPalindrome(Node* headNode) {
        vector<int> nodeData;
        Node* currNode = headNode;
        while (currNode != nullptr) {
            nodeData.push_back(currNode->data);
            currNode = currNode->next;
        }
        int len = nodeData.size();
        for (int idx = 0; idx < len / 2; idx++) {
            if (nodeData[idx] != nodeData[len - idx - 1]) {
                return false;
            }
        }
        return true;
    }
};
```

Time Complexity: O(N)

2. **Equal Array:**

```
class Solution {
  public:
    bool check(vector<int>& arr1, vector<int>& arr2) {
        sort(arr1.begin(),arr1.end());
        sort(arr2.begin(),arr2.end());
        return arr1 == arr2;
    }
};
```

Time Complexity: O(N log N)

3. **Floor in Sorted Array:**

```
class Solution {
public:
    int findFloor(vector<int>& numArray, int targetVal) {
        int maxFloor = -1;
        int resultIdx = -1;
        for (int idx = 0; idx < numArray.size(); idx++) {
            if (numArray[idx] <= targetVal && numArray[idx] > maxFloor) {
                maxFloor = numArray[idx];
                resultIdx = idx;
            }
        }
        return resultIdx;
    }
};
```

Time Complexity: O(N)

4. **Triplet Sum:**

```
class Solution {
public:
    bool find3Numbers(int dataArr[], int arrSize, int sumVal) {
        sort(dataArr, dataArr + arrSize);

        for (int idx = 0; idx < arrSize - 2; idx++) {
            int leftPtr = idx + 1;
            int rightPtr = arrSize - 1;

            while (leftPtr < rightPtr) {
                int currentTotal = dataArr[idx] + dataArr[leftPtr] + dataArr[rightPtr];

                if (currentTotal == sumVal) {
                    return true;
                } else if (currentTotal < sumVal) {
                    leftPtr++;
                } else {
                    rightPtr--;
                }
            }
        }

        return false;
    }
};
```

Time Complexity: O(N^2)

5. **Balanced tree check**

```
class Solution {
public:
    bool isBalanced(TreeNode* rootNode) {
        return getHeight(rootNode) != -1;
    }
private:
    int getHeight(TreeNode* currentNode) {
        if (!currentNode) return 0;
        int leftSubtreeHeight = getHeight(currentNode->left);
        if (leftSubtreeHeight == -1) return -1;
        int rightSubtreeHeight = getHeight(currentNode->right);
        if (rightSubtreeHeight == -1) return -1;
        if (abs(leftSubtreeHeight - rightSubtreeHeight) > 1) return -1;
        return max(leftSubtreeHeight, rightSubtreeHeight) + 1;
    }
};
```

Time Complexity: O(N)

6. 0/1 Knapsack Problem:

```
class Solution {
public:
    int knapSack(int capacity, vector<int>& val, vector<int>& wt) {
        int n = val.size();
        vector<vector<int>> dp(n + 1, vector<int>(capacity + 1, 0));
        for (int i = 1; i <= n; i++) {
            for (int j = 0; j <= capacity; j++) {
                dp[i][j] = dp[i - 1][j];
                if (j >= wt[i - 1]) {
                    dp[i][j] = max(dp[i][j], dp[i - 1][j - wt[i - 1]] + val[i - 1]);
                }
            }
        }
        return dp[n][capacity];
    }
};
```

Time Complexity: O(N * Capacity)

Output: