# MST-Assignment

**Student Name:** Dipanshu Sehgal
**UID:** 22BCS13346
**Branch:** CSE
**Section:** IOT_625_B
**Semester:** 6th
**Date Of Perf.:** 3 April 2025
**Subject:** Advanced programming-2
**Subject Code:** 22CSP-367

**Aim:** to solve the following problems –

Two sum,longest subsequence without repeating characters,Palindrome number,detect a cycle,maximum subarray,longest subsequence,validate binary search tree,word break,maximum subarray, trapping rain water.

**Objective:** to solve the following problems with optimal approach in java.

**Problems -:**

Q1- **Two Sum**: Given an array of integers, return indices of the two numbers such that they add up to a specific target.

Code- import java.util.*;

```
class Solution {
    public int[] twoSum(int[] nums, int target) {
        Map<Integer, Integer> map = new HashMap<>();
        for (int i = 0; i < nums.length; i++) {
            if (map.containsKey(target - nums[i]))
                return new int[]{map.get(target - nums[i]), i};
            map.put(nums[i], i);
        }
        return new int[]{};
    }
}
```

Q2- **Longest Substring Without Repeating Characters:** Given a string s, find the length of the longest substring that does not contain any repeating characters.

Code- import java.util.*;

```
class Solution {
```

```
    public int lengthOfLongestSubstring(String s) {
        Set<Character> set = new HashSet<>();
        int left = 0, maxLen = 0;
        for (int right = 0; right < s.length(); right++) {
            while (set.contains(s.charAt(right)))
                set.remove(s.charAt(left++));
            set.add(s.charAt(right));
            maxLen = Math.max(maxLen, right - left + 1);
        }
        return maxLen;
    }
}
```

Q3- **Palindrome Number**: Determine whether an integer is a palindrome.

Code-
```
class Solution {
    public boolean isPalindrome(int x) {
        if (x < 0 || (x % 10 == 0 && x != 0)) return false;
        int rev = 0;
        while (x > rev) {
            rev = rev * 10 + x % 10;
            x /= 10;
        }
        return x == rev || x == rev / 10;
    }
}
```

Q4- **Detect a Cycle in a Linked List:** Given the head of a linked list, determine whether the linked list contains a cycle. A cycle occurs if a node's next pointer points to a previous node in the list.

Code-
```
class ListNode {
    int val;
    ListNode next;
    ListNode(int x) { val = x; next = null; }
}

class Solution {
    public boolean hasCycle(ListNode head) {
        ListNode slow = head, fast = head;
        while (fast != null && fast.next != null) {
```

```
            slow = slow.next;
            fast = fast.next.next;
            if (slow == fast) return true;
        }
        return false;
    }
}
```

Q5- **Maximum Subarray**: Find the contiguous subarray (containing at least one number) that has the largest sum and return its sum

Code-

```
class Solution {
    public int maxSubArray(int[] nums) {
        int maxSum = nums[0], currSum = nums[0];
        for (int i = 1; i < nums.length; i++) {
            currSum = Math.max(nums[i], currSum + nums[i]);
            maxSum = Math.max(maxSum, currSum);
        }
        return maxSum;
    }
}
```

Q6- **Longest Increasing Subsequence II:** Given an integer array nums, find the length of the longest strictly increasing subsequence. A subsequence is derived from the array by deleting some or no elements without changing the order of the remaining elements.

Code- import java.util.*;

```
class Solution {
    public int lengthOfLIS(int[] nums) {
        int[] dp = new int[nums.length];
        Arrays.fill(dp, 1);
        int maxLen = 1;
        for (int i = 1; i < nums.length; i++)
            for (int j = 0; j < i; j++)
                if (nums[i] > nums[j])
                    dp[i] = Math.max(dp[i], dp[j] + 1);
        return Arrays.stream(dp).max().getAsInt();
    }
```

```
}
```

Q7- **Validate Binary Search Tree:** Given the root of a binary tree, determine if it is a valid binary search tree (BST).

Code-
```
class TreeNode {
    int val;
    TreeNode left, right;
    TreeNode(int x) { val = x; }
}

class Solution {
    public boolean isValidBST(TreeNode root) {
        return isValid(root, Long.MIN_VALUE, Long.MAX_VALUE);
    }

    private boolean isValid(TreeNode node, long min, long max) {
        if (node == null) return true;
        if (node.val <= min || node.val >= max) return false;
        return isValid(node.left, min, node.val) && isValid(node.right, node.val, max);
    }
}
```

Q8 - **Word Break:** Given a string s and a dictionary wordDict containing a list of words, determine if s can be segmented into a space-separated sequence of one or more dictionary words. The same word can be reused multiple times

Code-
```
import java.util.*;

class Solution {
    public boolean wordBreak(String s, List<String> wordDict) {
        Set<String> set = new HashSet<>(wordDict);
        boolean[] dp = new boolean[s.length() + 1];
        dp[0] = true;
        for (int i = 1; i <= s.length(); i++)
            for (int j = 0; j < i; j++)
                if (dp[j] && set.contains(s.substring(j, i))) {
                    dp[i] = true;
                    break;
                }
        return dp[s.length()];
```

```
    }
}
```

Q9- Trapping Rain Water: Given n non-negative integers representing an elevation map where the width of each bar is 1, compute the total amount of water that can be trapped after raining.

Code-
```
class Solution {
    public int trap(int[] height) {
        int left = 0, right = height.length - 1, leftMax = 0, rightMax = 0, water = 0;
        while (left < right) {
            if (height[left] < height[right]) {
                if (height[left] >= leftMax) leftMax = height[left];
                else water += leftMax - height[left];
                left++;
            } else {
                if (height[right] >= rightMax) rightMax = height[right];
                else water += rightMax - height[right];
                right--;
            }
        }
        return water;
    }
}
```

## Output:

Successfully implemented and analyzed key algorithmic problems, including **Two Sum, Longest Substring Without Repeating Characters, Palindrome Number, Cycle Detection, Maximum Subarray, Longest Increasing Subsequence, Validate BST, Word Break, and Trapping Rain Water.** Applied **hashing, sliding window, recursion, DP, Kadane's algorithm, and two-pointer techniques** to optimize solutions in Java.

## Learning Outcomes:

a) Improved understanding of **arrays, linked lists, trees, and hash maps**.
b) Learned **time complexity trade-offs and optimization strategies**.
c) Strengthened problem-solving with **efficient DSA techniques**