

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Medium problems for Average Learners

Student Name: Om Ankur Prajapati

UID: 22BCS15926

Branch: CSE

Section/Group: NTPP_602-A

Semester: 6

Date of Performance: 10-04-25

Subject Name: Advanced Programming Lab-2

Subject Code: 22CSH-359

Problem-1

1. **Title:** Palindrome Number

2. **Objective:** To determine whether a given integer is a palindrome (reads the same forward and backward).

3. **Algorithm:**

- Input: An integer x
- Edge Case:
 - If $x < 0$, it cannot be a palindrome.
- Convert to String (or use digit reverse logic):
 - Convert number to string and compare it with its reverse.
- Output:
 - Return true if both match, else false.

4. **Implementation/Code:**

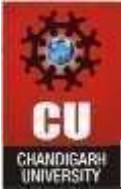
```
class Solution {
    public boolean isPalindrome(int x) {
        if (x < 0) return false;
        String s = Integer.toString(x);
        StringBuilder rev = new StringBuilder(s).reverse();
        return s.equals(rev.toString());
    }
}
```

5. **Output:**

- Input: 121 → Output: true
- Input: -121 → Output: false

6. **Time Complexity:** $O(n)$

7. **Space Complexity:** $O(n)$



Problem-2

1. Title: Two Sum

2. Objective: To find the indices of two numbers in an array that sum up to a specific target value.

3. Algorithm:

- Input: Array nums, Integer target
- Use Hash Map:
 - Store each number and its index as you iterate.
 - Check if (target - current number) exists in the map.
- Output: Return the pair of indices.

4. Implementation/Code:

```
class Solution {  
    public int[] twoSum(int[] nums, int target) {  
        Map<Integer, Integer> map = new HashMap<>();  
        for (int i = 0; i < nums.length; i++) {  
            int diff = target - nums[i];  
            if (map.containsKey(diff)) {  
                return new int[]{map.get(diff), i};  
            }  
            map.put(nums[i], i);  
        }  
        return new int[]{}; // No solution found  
    }  
}
```

5. Output:

- Input: `nums = [2, 7, 11, 15]`, `target = 9` → Output: `[0, 1]`

6. Time Complexity: $O(n)$

7. Space Complexity: $O(n)$



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Problem-3

1. Title: Longest Substring Without Repeating Characters

2. Objective: To find the length of the longest substring with all unique characters.

3. Algorithm:

- Input: A string s
- Sliding Window + Set:
 - Use two pointers (left, right) to maintain a window of non-repeating characters.
 - Move window when duplicate is found.
- Output: Return max length found.

4. Implementation/Code:

```
class Solution {
    public int lengthOfLongestSubstring(String s) {
        Set<Character> set = new HashSet<>();
        int left = 0, maxlen = 0;

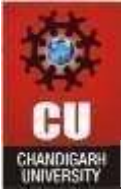
        for (int right = 0; right < s.length(); right++) {
            while (set.contains(s.charAt(right))) {
                set.remove(s.charAt(left++));
            }
            set.add(s.charAt(right));
            maxlen = Math.max(maxlen, right - left + 1);
        }
        return maxlen;
    }
}
```

5. Output:

• Input: "abcabcbb" → Output: 3

8. Time Complexity: $O(n)$

9. Space Complexity: $O(\min(n,m))$



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

10. Learning Outcomes:

- Understood how to handle edge cases like negatives.
- Practiced string operations to simplify integer problems.
- Applied hash map for constant-time lookup.
- Strengthened logic for pair-sum problems.
- Mastered sliding window technique.
- Learned to track seen characters efficiently with sets.