DSA Programming Practice – 9 – KAVYA U – CSBS(III yr) – 2026 BATCH – 22CB024

1. Valid Palindrome: O(N)

```
class Solution {
    public boolean isPalindrome(String s) {
        String validChars = "abcdefghijklmnopqrstuvwxyz1234567890";
        StringBuilder filtered = new StringBuilder();
        String lowerCase = s.toLowerCase();

        for (char c : lowerCase.toCharArray()) {
            if (validChars.indexOf(c) != -1) {
                filtered.append(c);
            }
        }

        int left = 0, right = filtered.length() - 1;
        while (left < right) {
            if (filtered.charAt(left) != filtered.charAt(right)) {
                return false;
            }
            left++;
            right--;
        }
        return true;
    }
}
```

2. Is Subsequence: O(N)

```
class Solution {
    public boolean isSubsequence(String s, String t) {
        int i = 0, j = 0;
        while (i < s.length() && j < t.length()) {
            if (s.charAt(i) == t.charAt(j)) {
                i++;
            }
            j++;
        }
        return i == s.length();
    }
}
```

3. Two Sum II: O(N)

```
class Solution {
    public int[] twoSum(int[] numbers, int target) {
        int left = 0, right = numbers.length - 1;
        while (left < right) {
            int sum = numbers[left] + numbers[right];
            if (sum == target) {
                return new int[]{left + 1, right + 1};
            } else if (sum < target) {
                left++;
            } else {
                right--;
            }
        }
        return new int[]{};
    }
}
```

4. 3Sum: O(N^2)

```
class Solution {
    public List<List<Integer>> threeSum(int[] nums) {
        List<List<Integer>> result = new ArrayList<>();
        Arrays.sort(nums);
        int n = nums.length;

        for (int i = 0; i < n - 2; i++) {
            if (i > 0 && nums[i] == nums[i - 1]) {
                continue;
            }

            int left = i + 1, right = n - 1;
            while (left < right) {
                int sum = nums[i] + nums[left] + nums[right];
                if (sum == 0) {
                    result.add(Arrays.asList(nums[i], nums[left], nums[right]));
                    while (left < right && nums[left] == nums[left + 1]) left++;
                    while (left < right && nums[right] == nums[right - 1]) right--;

                    left++;
                    right--;
                } else if (sum < 0) {
                    left++;
                } else {
                    right--;
                }
            }
        }
        return result;
    }
}
```

5. Minimise Size Subarray: O(N)

```java
class Solution {
    public int minSubArrayLen(int target, int[] nums) {
        int left = 0, result = Integer.MAX_VALUE, total = 0;
        for (int right = 0; right < nums.length; right++) {
            total += nums[right];
            while (total >= target) {
                result = Math.min(result, right - left + 1);
                total -= nums[left];
                left++;
            }
        }
        return result == Integer.MAX_VALUE ? 0 : result;
    }
}
```

6. Substring with concatenation of all words: O(N*M)

```java
class Solution {
    public List<Integer> findSubstring(String s, String[] words) {
        List<Integer> result = new ArrayList<>();
        if (s == null || words == null || words.length == 0) {
            return result;
        }
        Map<String, Integer> wordCount = new HashMap<>();
        for (String word : words) {
            wordCount.put(word, wordCount.getOrDefault(word, 0) + 1);
        }
        int wordLen = words[0].length();
        int substringLen = wordLen * words.length;

        for (int i = 0; i <= s.length() - substringLen; i++) {
            Map<String, Integer> seen = new HashMap<>();
            int j = i;

            while (j < i + substringLen) {
                String word = s.substring(j, j + wordLen);
                if (wordCount.containsKey(word)) {
                    seen.put(word, seen.getOrDefault(word, 0) + 1);
                    if (seen.get(word) > wordCount.get(word)) {
                        break;
                    }
                } else {
                    break;
                }
                j += wordLen;
            }

            if (j == i + substringLen) {
                result.add(i);
            }
        }
        return result;
    }
}
```

7. Minimum window substring: O(N)

```java
import java.util.*;

class Solution {
    public String minWindow(String s, String t) {
        if (s == null || t == null || s.length() < t.length()) {
            return "";
        }

        Map<Character, Integer> mpt = new HashMap<>();
        Map<Character, Integer> mps = new HashMap<>();
        for (char c : t.toCharArray()) {
            mpt.put(c, mpt.getOrDefault(c, 0) + 1);
        }

        int cur = 0, total = mpt.size();
        int left = 0, right = 0;
        int minLen = Integer.MAX_VALUE;
        int start = -1;

        while (right < s.length()) {
            char c = s.charAt(right);
            if (mpt.containsKey(c)) {
                mps.put(c, mps.getOrDefault(c, 0) + 1);
                if (mps.get(c).equals(mpt.get(c))) {
                    cur++;
                }
            }

            while (cur == total) {
                if (right - left + 1 < minLen) {
                    minLen = right - left + 1;
                    start = left;
                }

                char leftChar = s.charAt(left);
                if (mpt.containsKey(leftChar)) {
                    mps.put(leftChar, mps.get(leftChar) - 1);
                    if (mps.get(leftChar) < mpt.get(leftChar)) {
                        cur--;
                    }
                }
                left++;
            }
            right++;
        }

        return start == -1 ? "" : s.substring(start, start + minLen);
    }
}
```

8. Valid Parenthesis: O(N)

```java
import java.util.*;

class Solution {
    public boolean isValid(String s) {
        Map<Character, Character> pairs = new HashMap<>();
        pairs.put('(', ')');
        pairs.put('{', '}');
        pairs.put('[', ']');

        Stack<Character> stack = new Stack<>();
        for (char c : s.toCharArray()) {
            if (pairs.containsKey(c)) {
                stack.push(c);
            } else if (stack.isEmpty() || c != pairs.get(stack.pop())) {
                return false;
            }
        }

        return stack.isEmpty();
    }
}
```

9. Simplify path: O(N)

```java
import java.util.*;

class Solution {
    public String simplifyPath(String path) {
        Stack<String> stack = new Stack<>();
        String[] dirs = path.split("/");

        for (String dir : dirs) {
            if (dir.equals("..")) {
                if (!stack.isEmpty()) {
                    stack.pop();
                }
            } else if (!dir.equals(".") && !dir.equals("") && !dir.equals("..")) {
                stack.push(dir);
            }
        }

        StringBuilder result = new StringBuilder();
        for (String dir : stack) {
            result.append("/").append(dir);
        }

        return result.length() == 0 ? "/" : result.toString();
    }
}
```

10. Min stack: O(1)

```
class MinStack {
    private Stack<int[]> stack;

    public MinStack() {
        stack = new Stack<>();
    }

    public void push(int val) {
        int minVal = (stack.isEmpty()) ? val : Math.min(val, stack.peek()[1]);
        stack.push(new int[]{val, minVal});
    }

    public void pop() {
        stack.pop();
    }

    public int top() {
        return stack.peek()[0];
    }

    public int getMin() {
        return stack.peek()[1];
    }
}
```

11. Evaluate reverse nail polish notation: O(N)

```
class Solution {
    public int evalRPN(String[] tokens) {
        Stack<Integer> stack = new Stack<>();

        for (String token : tokens) {
            if (token.equals("+")) {
                int second = stack.pop();
                int first = stack.pop();
                stack.push(first + second);
            } else if (token.equals("-")) {
                int second = stack.pop();
                int first = stack.pop();
                stack.push(first - second);
            } else if (token.equals("*")) {
                int second = stack.pop();
                int first = stack.pop();
                stack.push(first * second);
            } else if (token.equals("/")) {
                int second = stack.pop();
                int first = stack.pop();
                stack.push(first / second); // Integer division truncates towards zero
            } else {
                stack.push(Integer.parseInt(token));
            }
        }

        return stack.pop();
    }
}
```

12. Basic Calculator: O(N)

```java
import java.util.*;

class Solution {
    public int calculate(String s) {
        int number = 0;
        int signValue = 1;
        int result = 0;
        Stack<Integer> operationsStack = new Stack<>();

        for (char c : s.toCharArray()) {
            if (Character.isDigit(c)) {
                number = number * 10 + (c - '0');
            } else if (c == '+' || c == '-') {
                result += number * signValue;
                signValue = (c == '-') ? -1 : 1;
                number = 0;
            } else if (c == '(') {
                operationsStack.push(result);
                operationsStack.push(signValue);
                result = 0;
                signValue = 1;
            } else if (c == ')') {
                result += signValue * number;
                result *= operationsStack.pop();
                result += operationsStack.pop();
                number = 0;
            }
        }

        return result + number * signValue;
    }
}
```

13. Search insert position: O(N)

```java
class Solution {
    public int searchInsert(int[] nums, int target) {
        int l = 0, r = nums.length - 1;

        while (l <= r) {
            int mid = (l + r) / 2;
            if (nums[mid] < target) {
                l = mid + 1;
            } else if (nums[mid] > target) {
                r = mid - 1;
            } else {
                return mid;
            }
        }

        return l;
    }
}
```

14. Search 2D matrix: O(M*N)

```
class Solution {
    public boolean searchMatrix(int[][] matrix, int target) {
        for (int[] row : matrix) {
            for (int num : row) {
                if (num == target) {
                    return true;
                }
            }
        }
        return false;
    }
}
```

15. Find peak element: O(N)

```
class Solution {
    public int findPeakElement(int[] nums) {
        int l = 0, r = nums.length - 1;

        while (l < r) {
            int mid = (l + r) / 2;
            if (nums[mid] < nums[mid + 1]) {
                l = mid + 1;
            } else {
                r = mid;
            }
        }

        return l;
    }
}
```

16. First and last position in sorted array: O(N)

```
import java.util.*;

class Solution {
    public int[] searchRange(int[] nums, int target) {
        int[] result = {-1, -1};
        if (nums == null || nums.length == 0) {
            return result;
        }
        int left = -1, right = -1;

        for (int i = 0; i < nums.length; i++) {
            if (nums[i] == target) {
                left = i;
                break;
            }
        }

        for (int i = nums.length - 1; i >= 0; i--) {
            if (nums[i] == target) {
                right = i;
                break;
```

```
            }
        }
        if (left != -1 && right != -1) {
            result[0] = left;
            result[1] = right;
        }
        return result;
    }
}
```

17. Find minimum in rotated sorted array: O((N + M) log(N + M))

```
class Solution {
    public int findMin(int[] nums) {
        int left = 0, right = nums.length - 1;

        while (left < right) {
            int mid = left + (right - left) / 2;

            if (nums[mid] < nums[right]) {
                right = mid;
            } else {
                left = mid + 1;
            }
        }

        return nums[left];
    }
}
```

18. Median of two sorted array: O(logN)

```
import java.util.*;

class Solution {
    public double findMedianSortedArrays(int[] nums1, int[] nums2) {
        int[] merged = new int[nums1.length + nums2.length];
        System.arraycopy(nums1, 0, merged, 0, nums1.length);
        System.arraycopy(nums2, 0, merged, nums1.length, nums2.length);

        Arrays.sort(merged);

        int n = merged.length;
        if (n % 2 == 1) {
            return merged[n / 2];
        } else {
            return (merged[n / 2] + merged[n / 2 - 1]) / 2.0;
        }
    }
}
```