Maths and Common Programming Operations

1. Write a program to print the k^{th} digit from last and begin. Ex: - input: 23617 and k = 4, then output: last=3, begin=1.
2. Write a program, which will find the sum of products of two consecutive digits. E.g. input: 23145, then output: 2x3 + 3x1 + 1x4 + 4x5 = 33.
3. Write a program, which reads two numbers (assume, both have the same number of digits) from the user. The program outputs the sum of products of corresponding digits. Ex: - input: 327 and 539, then output: 3x5 + 2x3 + 7x9 = 84.
4. Write a program to print positional values of digits. Ex: - Input: 21463 output: 3, 60, 400, 1000 and 20000.
5. Write a program to reverse a user-input number. Ignore leading zeros. Ex: - input: 4321, output: 1234. Input: 120, output: 21.
6. Write a program to find the sum of even digits and odd digits separately. Ex: - input: 23617 output: even sum = 2+6 = 8 and odd sum = 3 + 1 + 7 = 11.
7. Write a program to find the maximum and minimum digit from a number.
8. Write a program to print the second last even digit. Ex: - input: 23863, output: 8. Input: 325145761, output 4.
9. Write a program, which finds the sum of numbers formed by consecutive digits. Ex: - input: 2415, output: 24 + 41 + 15 = 80. Input: 24159, output: 24 + 41 + 15 + 59 = 139.
10. Write a program to find the sum of numbers formed by exchanging consecutive digits. Ex: - input: 2415, output: 42 + 14 + 51 = 107. Input: 24159, output: 42 + 14 + 51 + 95 = 202.
11. Write a program to find the number of times x digit occurs in a given input. Ex: - input: 948932 and x = 9, output: 2.
12. Write a program to find the largest prime factor of a number.
13. Write a program to express a user-input number as a sum of two prime numbers.
14. Write a program to add to fractions. Ex: - 1/3 + 3/9 = 2/3 (take numerators and denominators separately as input and give the simplified fraction of the sum as the output.

15. Write a program to calculate the permutation of “n” people who can occupy “x” seats in an empty room. Ex: - input: n=5 and x=4, output: 120 ways.

Arrays

1. Write a program to find the second largest element in an array.
2. Write a program to reverse an array.
3. Write a program to remove duplicates from:
    a. A sorted array
    b. An unsorted array
4. Write a program to check if an array is sorted (whether it be in increasing or decreasing order) or not.
5. Write a program to find the “Leaders” in an array. P.S.: Leaders in an array is that element, which is followed by all elements less than that, to its right. Ex: - arr[] = {7, 10, 4, 10, 6, 5, 2}, output: 10 6 5 2.
6. Write a program to list down the frequencies of all distinct elements from:
    a. A sorted array
    b. An unsorted array
7. Write a program to left rotate an array by:
    a. One space
    b. “k” spaces
8. Common operations on array: (for practice)
    a. Searching:
        i. Linear Search
        ii. Binary Search

b. Sorting:

    i. Bubble Sort
    ii. Insertion sort
    iii. Selection Sort
    iv. Quick Sort
    v. Merge Sort

9. Write a program to segregate/sort an array of 0’1, 1’s and 2’s without using any of the above sorting algorithms.
10. Write a program to find the maximum sum from a contiguous sub array of a one-dimensional array of numbers. Ex: - input: arr[] = {-2, -3, 4, -1, -2, 1, 5, -3}, output: 4 + (-1) + (-2) + 1 + 5 = 7.
11. Write a program to find maximum sum of “k” consecutive elements in the array. Ex: - input: a[] = {1, 4, 2, 10, 23, 3, 1, 0, 20} and k=4, output: 39 (we get maximum by adding the elements in the sub array {4, 2, 10, 23} of size “k” i.e. 4 here.
12. Write a program to search an element in a 2-D array (matrix), where the 2-D array must be sorted row-wise from left-to-right and first integer of each row is greater than last integer of previous row. Ex: - input: matrix = [

    [1, 3, 5, 7],

    [10, 11, 16, 20],

    [23, 30, 34, 50]

    ]; and target=16, output: True (False, if element not found).

13. Write a program to find the missing and a repeating number from an array. Below are the specifications: -

    a. Unordered array of size n

    b. Elements must be in the range 1 to n.

    Ex: - input: a[] = {4, 3, 1, 6, 2, 1}, output: missing = 5 & repeating = 1. Input: a[] = {3, 1, 3, 5, 4, 5}, output: missing = 2 & repeating = 3 and 5.

14. Write a program to find out the prefix sum array from a given array. Prefix sum array is another array of the same size, such that prefixsum[i] = arr[1] + arr[2] + …… + arr[n-1]. Ex: - input: arr[] = {10, 20, 10, 5, 15}, output: prefixSum[] = {10, 30, 40, 45, 60}.
15. Write a program to find:
    a. Maximum difference of two elements in an array. Ex: - input: a[] = {2, 3, 10, 6, 4, 8, 1}, output: 8 (2 and 10 as 10 – 2 = 8)
    b. Minimum difference of two elements in an array. Ex: - input: a[] = {2, 3, 10, 6, 4, 8, 1}, output: 1 (2 and 3 as 3 – 2 = 1)
    c. First repeating element in an array. Ex: - arr[] = {10, 5, 3, 4, 3, 5, 6}, output: 5.
    d. First non-repeating elements in an array. Ex: - arr[] = {10, 5 , 3, 4, 3, 5, 6}, output: 10.

Strings

1. Write a program to check if a user-input string is palindrome or not.
2. Write a program to toggle each character in a string. Ex: - input: “EduHub”, output: “eDUhUB”. Input: “int”, output: “INT”.
3. Write a program to calculate and print the frequencies of each distinct character in a string (all the characters in the string are lower case). Ex: - input: “eduhub”, output: b=1, d=1, e=1, h=1, u=2.
4. Write a program to check for anagram. Anagram of a word is another word formed by rearranging the letters. Ex: - input: s1 = “listen” and s2 = “silent”, output: “YES”. Input: s1 = “aab” and s2 = “bab”, output: “NO”.
5. Write a program to find all subsequences of a string. A string is a subsequence of a given string, which is formed by removing some characters of the given string. Ex: - input: “abc”, output: a, b, c, ab, bc, ac, abc. Input: “aaa”, output: a, aa, aaa.
6. Write a program to check if a string is a subsequence of another or not. Ex: - input: s1 = “ABCD” and s2 = “AD”, output: True. Input: s1 = “ABCDE” and s2 = “AED”, output: False.
7. Write a program to reverse words in a string. Corner Case: - If there is only one word in a string, then reverse that word, character-wise. Ex: - input: “Welcome to Eduhub Community”, output: “Community Eduhub to Welcome”. Input: “Hello”, output: “olleH”.

8. Write a program to check if the given pattern is available in the given string or not (Pattern Searching). Ex: - input: “Hello World” and pattern = “Wor”, output: “Found at index 6”. Input: “AABACDAABABAABA” and pattern = “AABA”, output: “Found at index 0, 6 and 11”.

9. Write a program to find:

    a. Leftmost repeating character in a string (assume all characters to be lowercase). Ex: - input: “attract”, output: “a”.

    b. Leftmost non-repeating character in a string (assume all characters to be lowercase). Ex: - input: “attract”, output: “r”.

10. Write a program to generate and count all permutations of a given string. Ex: - input: “ABC”, output: ABC ACB BAC BCA CBA CAB and count=6.

11. Write a program to find the sum of all numbers present in an alpha-numeric string. Ex: - input: “1wabc23z”, output: 1 + 2 + 3 = 6. The input should be alpha-numeric (containing both digits and characters), if the input contains only characters or only digits, then the output should be null or 0 respectively.

Linked Lists

1. Common operations on Linked Lists (for both singly and doubly) – for practice

    a. Creation

    b. Insertion

        i. At beginning

        ii. At end

        iii. At any position

            1. Before or after a given index

            2. Before or after a given value of node

    c. Deletion

        i. At beginning

ii. At end

iii. At any position

1. Before or after a given index
2. Before or after a given value of node

d. Count the number of nodes

e. Traverse and Display the whole linked list

f. Searching a particular node

g. Sorting – increasing and decreasing

h. Reversing

Note: Circular linked list and its operations are almost the same as singly linked list, just instead of pointing the last node to null, we point it to the head/start.

2. Write a program to find the n^{th} node from end and insert another node into a linked list.

3. Write a program to delete m nodes after n nodes in a linked list.

4. Write a program to remove duplicates from a sorted linked list.

5. Write a program to:

   a. Merge two linked lists at alternate positions. Ex: - input: l1 = 1->3->5->7 & l2 = 2->4->6->8->10, output: 1->2->3->4->5->6->7->8->10.

   b. Split a linked list into two linked lists, on the basis of the middle element. Ex: - input: 1->2->3->4->5->6->7->8->9->10, output: l1 = 1->2->3->4->5 & l2 = 6->7->8->9->10.

6. Write a program to check if the given linked list is palindrome or not. (The values can be either integers or characters).

Stacks & Queues

1. Write a program to implement a stack using an array.
2. Write a program to implement a stack using a linked list.
3. Write a program to check if an expression of brackets is valid or not using stacks. (Balanced parentheses in an expression) Consider there are 3 types of brackets: “{}, [], & ()”. Ex: - input: “[{({})}]”, output: “Balanced”. Input: “[(]}]”, output: “Not balanced”.
4. Write a program to find the next greater element present/not present in the stack. Ex: - input: 4, 5, 2, 25, 3, output: The next greater element for 4 is 5, 5 is 25, 2 is 25, 25 is -1 and 3 is -1. (Note: if there is no next greater element present for a particular element, then return -1).
5. Write a program to find the previous greater element present/not present in the stack. Ex: - input: 40, 30, 20, 10, output: The previous greater element for 40 is -1, 30 is 40, 20 is 30, 10 is 20. (Note: if there is no previous greater element present for a particular element, then return -1).
6. Write a program to sort (increasing or decreasing) the elements in a stack.
7. Write a program to implement:
    a. 2 stacks in a single array.
    b. “k” stacks in a single array.
8. Write a program to find:
    a. The minimum element from a stack.
    b. The maximum element from a stack.
9. Write a program to print the length of the valid substring in an expression of brackets. Ex: - input: “(()((”, output: valid substring = “()” and length = 2. Input: “()(())(“, output: valid substring = “()(())” and length=6.
10. Write a program to convert: (using stack)
    a. Infix to postfix. Ex: - infix: “a+b*(c^d-e)^(f+g*h)-i”, postfix: “abcd^e-fgh*+^*+i-“.
    b. Infix to prefix. Ex: infix: “a*b+c/d”, prefix: “+*ab/cd”.
11. Write a program to implement a queue using an array.

12. Write a program to implement a queue using a linked list.
13. Write a program to reverse a queue.
14. Write a program to implement stack using a queue.
15. Write a program to implement a queue using a stack.

Binary Trees and Binary Search Trees

1. Write a program to implement a Binary Tree.
2. Write a program to implement below mentioned traversals in a binary tree:
    a. In-order
    b. Pre-order
    c. Post-order
    d. Level-order
3. Write a program to calculate:
    a. Height of binary tree.
    b. Diameter of binary tree.
    c. Number of nodes in a binary tree.
    d. Number of leaf nodes in a binary tree.
    e. Distance between 2 nodes in a binary tree.
4. Write a program to mirror a binary tree, such that the left nodes will now become right nodes and vice-versa.
5. Write a program to print a binary tree as per below mentioned views:
    a. Top-view
    b. Bottom-view
    c. Left-view
    d. Right-view

6. Write a program to check if a binary tree is balanced or not.
7. Write a program to check whether a binary tree follows Children sum property or not. Children sum property is a property of a binary tree, in which the sum of the value of left child and right child is equal to their parent node. If only one child is present, then its value should be equal to its parent node.
8. Write a program to check if two trees are identical or not.
9. Write a program to find the Lowest Common Ancestor of two nodes.
10. Write a program to find the n^{th} ancestor of a node in a binary tree.
11. Write a program to
    a. Search a user-input node in a binary search tree.
    b. Insert a user-input node in a binary search tree.
    c. Delete a user-input node in a binary search tree.
12. Write a program to find the floor of a node in a binary search tree. (Here, floor of a node is the node that is immediately lesser than the entered node value.)
13. Write a program to find the ceil of a node in a binary search tree. (Here, the ceil of a node is the node that is immediately greater than the entered node value.)

Heap

1. Write a program to implement a binary heap:
    a. Min-heap
    b. Max-heap
    c. Insertion of elements
2. Write a program to perform below operations:
    a. Min-heapify
    b. Max-heapify

    c. Extract-min element
    d. Extract-max element
3. Write a program to implement Heap Sort.
4. Write a program to implement Priority Queue.

Graph

1. Write a program to represent a graph in the below given ways:
    a. Adjacency Matrix
    b. Adjacency List
2. Write a program to perform the following operations related to graph:
    a. Breadth-First Search (BFS)
    b. Depth-First Search (DFS)
3. Write a program to detect cycles in:
    a. Undirected graph
    b. Directed graph
4. Write a program to find the minimum spanning tree out from a given graph:
    a. Prim's Algorithm
    b. Kruskal's Algorithm
5. Write a program to find out the shortest path among nodes of a graph:
    a. Djistra's Algorithm
    b. Bellman Ford Algorithm
    c. Floyd-Warshall Algorithm

Greedy Problems

1. Activity selection problem.

2. Fractional knapsack problem.

3. Job sequencing problem.

4. Huffman Coding.

Backtracking Problems

1. Rat in a maze problem.
2. N Queens Problem.
3. Sudoku Problem.

Dynamic Programming

1. Longest Common Subsequence.
2. Edit Distance Problem.
3. Longest Increasing Subsequence.
4. Matrix chain multiplication.
5. Maximum Cuts.
6. 0-1 Knapsack Problem.
7. Egg dropping puzzle.
8. Subset Sum Problem.
9. Maximum Sum with no two consecutive.
10. Allocating minimum number of pages.