Question Bank

Set 1: Data Structures and Algorithms

Easy (2 x 15 = 30 Marks)

Q1. Array – Second Largest Element

- **Problem Description:** The task is to find the second-largest unique element in an array of integers. This requires sorting or iterating through the array to identify the two largest values.
- **Input/Output Format:**
  - **Input:** The first line contains an integer N representing the size of the array. The second line contains N space-separated integers.
  - **Output:** A single integer representing the second-largest element.
- **Constraints:** 2leNle1000
- **Sample Test Cases:**
  - **Input:**
    5
    10 20 4 45 99

  - **Output:**
    45

- **Solution Outline:**
  - **Method 1 (Sorting):** Sort the array in descending order. The second element will be the second largest. Handle duplicates by finding the first element that is not equal to the largest.
  - **Method 2 (Iteration):** Initialize two variables, first_max and second_max, to a very small number (or the first two unique elements of the array). Iterate through the array: if an element is greater than first_max, update second_max to first_max and first_max to the new element. If an element is greater than second_max but not equal to first_max, update second_max to the new element.

Q2. String – Palindrome Check

- **Problem Description:** Check if a given string is a palindrome, ignoring case sensitivity. A palindrome is a sequence that reads the same forwards and backward.
- **Input/Output Format:**
  - **Input:** A single string without spaces.
  - **Output:** YES if the string is a palindrome, NO otherwise.
- **Sample Test Cases:**
  - **Input:**
    Level

  - **Output:**
    YES

- **Solution Outline:**
  - Convert the entire string to a single case (e.g., lowercase) to handle case-insensitivity.
  - Use two pointers: one at the beginning and one at the end of the string. Move them towards the center, comparing the characters at each position. If at any point the characters don't match, it's not a palindrome. If the pointers meet or cross without any mismatches, it is a palindrome.

Medium (2 x 20 = 40 Marks)

Q3. Stack – Array Implementation

- **Problem Description:** Implement a stack using a static array. The implementation should support PUSH, POP, and DISPLAY operations.
- **Input/Output Format:**
  - **Input:** A series of commands, each on a new line (PUSH x, POP, DISPLAY), terminated by the "EXIT" command.
  - **Output:** The program should print the result of each operation. For PUSH, print "PUSHED [value]". For POP, print "POPPED [value]". For DISPLAY, print the current elements of the stack.
- **Sample Test Cases:**
  - **Input:**
    PUSH 10
    PUSH 20

POP
DISPLAY
EXIT

- **Output:**
  PUSHED 10
  PUSHED 20
  POPPED 20
  Stack: 10

- **Solution Outline:**
  - Define a global or static array to store the stack elements and an integer variable, top, to keep track of the top of the stack. Initialize top to -1.
  - **PUSH:** Check for stack overflow (top < max_size - 1). If there's space, increment top and store the new element at array[top].
  - **POP:** Check for stack underflow (top >= 0). If the stack isn't empty, retrieve the element at array[top] and decrement top.
  - **DISPLAY:** Iterate from top down to 0 and print each element.

Q4. Recursion – GCD

- **Problem Description:** Find the greatest common divisor (GCD) of two integers using the Euclidean algorithm implemented with recursion.
- **Input/Output Format:**
  - **Input:** Two integers a and b on a single line.
  - **Output:** A single integer representing the GCD of a and b.
- **Sample Test Cases:**
  - **Input:**
    24 36

  - **Output:**
    12

- **Solution Outline:**
  - The Euclidean algorithm is based on the principle that the GCD of two numbers also divides their difference. The recursive base case is when the second number (b) is 0; the GCD is the first number (a).
  - The recursive step is GCD(a, b) = GCD(b, a % b). The function repeatedly replaces the larger number with the remainder of the division of the two numbers until the remainder becomes zero.

Hard (1 x 30 = 30 Marks)

Q5. Linked List – Merge Sort

- **Problem Description:** Implement merge sort to sort a singly linked list. The program should first create the linked list from user input and then sort it.
- **Input/Output Format:**
  - **Input:** The first line contains an integer N. The second line contains N space-separated integers to form the linked list.
  - **Output:** The sorted linked list elements, printed in ascending order on a single line.
- **Sample Test Cases:**
  - **Input:**
    5
    50 20 40 10 30

  - **Output:**
    10 20 30 40 50

- **Solution Outline:**
  - **Merge Sort Algorithm:**
    1. **Base Case:** If the list is empty or has only one node, it's already sorted. Return the head.
    2. **Split:** Find the middle of the linked list using the fast and slow pointer technique. Split the list into two halves.
    3. **Recurse:** Recursively call merge sort on each half.
    4. **Merge:** Merge the two sorted halves back together into a single sorted list. This involves comparing the head nodes of the two lists and iteratively building the new sorted list.

Set 2: Data Structures and Algorithms

Easy (2 x 15 = 30 Marks)

Q1. Mathematical – Armstrong Number

- **Problem Description:** Check if a number is an Armstrong number. An Armstrong number is a number that is equal to the sum of its own digits raised to the power of the number of digits.
- **Input/Output Format:**
  - **Input:** A single integer.
  - **Output:** YES if the number is an Armstrong number, NO otherwise.
- **Sample Test Cases:**
  - **Input:**
    153
  - **Output:**
    YES
- **Solution Outline:**
  - Count the number of digits in the given integer.
  - Iterate through the digits of the number. For each digit, calculate the digit raised to the power of the total number of digits and add it to a running sum.
  - Compare the final sum with the original number. If they are equal, it's an Armstrong number.

Q2. Array – Reverse

- **Problem Description:** Reverse an array of integers in-place without using an additional array.
- **Input/Output Format:**
  - **Input:** The first line contains an integer N. The second line contains N space-separated integers.
  - **Output:** The elements of the reversed array, space-separated on a single line.
- **Sample Test Cases:**
  - **Input:**

5
1 2 3 4 5

- **Output:**
  5 4 3 2 1

- **Solution Outline:**
  - Use two pointers, one at the beginning (start) and one at the end (end) of the array.
  - Loop until start is less than end. In each iteration, swap the elements at the start and end indices, then increment start and decrement end.

Medium (2 x 20 = 40 Marks)

Q3. Queue – Circular Queue

- **Problem Description:** Implement a circular queue using an array. The implementation should support ENQUEUE, DEQUEUE, and DISPLAY operations, handling both empty and full conditions.
- **Input/Output Format:** Similar to the Stack problem (Q3, Set 1).
- **Solution Outline:**
  - Define an array for the queue and integer variables for the front and rear pointers. Initialize both to -1.
  - **ENQUEUE:** Check for queue full condition: (rear + 1) % size == front. If not full, increment rear circularly (rear = (rear + 1) % size) and add the element. If it's the first element, set front = 0.
  - **DEQUEUE:** Check for queue empty condition: front == -1. If not empty, retrieve the element at front. If front equals rear, reset both to -1 (queue is now empty). Otherwise, increment front circularly (front = (front + 1) % size).
  - **DISPLAY:** Iterate from front to rear (handling the wrap-around for circular queues) and print the elements.

Q4. String – Longest Substring Without Repeating Characters

- **Problem Description:** Given a string, find the length of the longest substring that contains no repeating characters.
- **Input/Output Format:**
  - **Input:** A single string.

    - **Output:** An integer representing the length of the longest substring.
- **Sample Test Cases:**
    - **Input:**
      abcabcbb

    - **Output:**
      3

- **Solution Outline:**
    - Use a sliding window approach with a hash map (or an array for a limited character set) to keep track of characters seen within the current window.
    - Use two pointers, left and right. Expand the window by moving the right pointer. If a character at right is already in the hash map, shrink the window by moving the left pointer until the repeating character is no longer in the window.
    - Keep track of the maximum window size encountered.

Hard (1 x 30 = 30 Marks)

Q5. Recursion – N-Queens

- **Problem Description:** Solve the N-Queens problem using recursion and backtracking. The goal is to place N queens on an N x N chessboard such that no two queens attack each other. The program should print one valid solution.
- **Input/Output Format:**
    - **Input:** A single integer N, the size of the board.
    - **Output:** An array or list of integers representing the column position of the queen for each row. For example, [2, 4, 1, 3] means the queen in row 1 is at column 2, row 2 at column 4, and so on.
- **Sample Test Cases:**
    - **Input:**
      4

    - **Output:**
      [2, 4, 1, 3]

- **Solution Outline:**
    - The problem can be solved by placing one queen at a time, starting from the first row.
    - Define a recursive function, solveNQueens(row, board).

- **Base Case:** If row == N, a valid solution has been found. Print the solution and return.
- **Recursive Step:** For the current row, iterate through all columns from 1 to N. For each column, check if placing a queen there is safe (i.e., no other queen is in the same column, or on the same diagonal).
    - If it's safe, place the queen (mark the position on the board), then recursively call solveNQueens(row + 1, board).
    - If the recursive call doesn't lead to a solution, backtrack: remove the queen from the current position and try the next column.

Set 3: Data Structures and Algorithms

Easy (2 x 15 = 30 Marks)

Q1. Array – Frequency of Elements

- **Problem Description:** Count the frequency of each distinct element in an array of integers.
- **Solution Outline:**
  - **Method 1 (Hash Map):** Use a hash map (or an array acting as a hash map if the numbers are within a limited range) to store the count of each element. Iterate through the input array and for each element, increment its count in the hash map. Finally, iterate through the hash map and print each element and its frequency.
  - **Method 2 (Sorting):** Sort the array. Then, iterate through the sorted array, counting consecutive occurrences of the same number. Print the number and its count whenever a new number is encountered.

Q2. String – Title Case Conversion

- **Problem Description:** Convert a given string into title case. Title case means the first letter of each word is capitalized, and the rest of the letters are lowercase.
- **Solution Outline:**
  - Iterate through the string, character by character. A character should be capitalized if it's the first character of the string or if it's preceded by a space. All other letters should be converted to lowercase.

Medium (2 x 20 = 40 Marks)

Q3. Linked List – Insert Operations

- **Problem Description:** Implement a singly linked list and provide functions to insert a

new node at the beginning, at the end, and at a specified position.

- **Solution Outline:**
  - **Insert at Beginning:** Create a new node, point its next pointer to the current head, and update the head to the new node.
  - **Insert at End:** Traverse the list to the last node. Create a new node and make the last node's next pointer point to the new node.
  - **Insert at Position:** Traverse the list to the node just before the desired position. Create a new node and link it by adjusting the pointers of the preceding and succeeding nodes. Handle edge cases like an invalid position or an empty list.

Q4. Sorting – Quick Sort

- **Problem Description:** Implement the quick sort algorithm to sort an array of integers in ascending order.
- **Solution Outline:**
  - **Quick Sort Algorithm:**
    1. **Pivot Selection:** Choose a pivot element from the array (e.g., the last element).
    2. **Partition:** Rearrange the array so that all elements smaller than the pivot come before it, and all elements greater than the pivot come after it. The final position of the pivot is its correct sorted position.
    3. **Recurse:** Recursively apply the above steps to the sub-array of elements with smaller values and the sub-array of elements with greater values. The recursion ends when a sub-array has one or zero elements.

Hard (1 x 30 = 30 Marks)

Q5. Stack – Postfix Evaluation

- **Problem Description:** Evaluate a postfix expression (Reverse Polish Notation) using a stack.
- **Solution Outline:**
  - Iterate through the postfix expression from left to right.
  - If an operand (number) is encountered, push it onto the stack.
  - If an operator (+, -, *, /) is encountered, pop the top two operands from the stack, perform the operation, and push the result back onto the stack.
  - After the entire expression has been traversed, the final result will be the only element left on the stack.

Set 4: Data Structures and Algorithms

Easy (2 x 15 = 30 Marks)

Q1. Mathematical – Prime Numbers

- **Problem Description:** Print all prime numbers up to a given integer N.
- **Solution Outline:**
    - **Sieve of Eratosthenes:** Create a boolean array isPrime of size N+1 and initialize all entries to true. Mark 0 and 1 as false. Iterate from p=2 up to sqrtN. For each p that is marked as true, mark all multiples of p (from $p*p$) as false. Finally, iterate through the isPrime array from 2 to N and print all indices i where isPrime[i] is true.

Q2. String – Vowel/Consonant Count

- **Problem Description:** Count the total number of vowels and consonants in a given string.
- **Solution Outline:**
    - Initialize two counters, one for vowels and one for consonants.
    - Convert the string to lowercase to simplify comparisons.
    - Iterate through the string. For each character, check if it's one of the vowels (a, e, i, o, u). If it is, increment the vowel counter. If it's a lowercase letter but not a vowel, increment the consonant counter. Ignore other characters (numbers, symbols, spaces).

Medium (2 x 20 = 40 Marks)

Q3. Stack – Balanced Parentheses

- **Problem Description:** Check if a string containing parentheses ( ), brackets [ ], and curly braces { } has balanced and correctly nested pairs.
- **Solution Outline:**

- Use a stack. Iterate through the string character by character.
- If an opening bracket ((, [, {) is found, push it onto the stack.
- If a closing bracket (),], }) is found, check if the stack is empty or if the top of the stack is the corresponding opening bracket. If it is, pop the stack. If not, the parentheses are unbalanced.
- After iterating through the entire string, the stack should be empty for the expression to be balanced.

Q4. Recursion – Fibonacci

- **Problem Description:** Generate the Fibonacci series up to a given number of terms using recursion.
- **Solution Outline:**
  - Define a recursive function fib(n) that returns the n-th Fibonacci number.
  - **Base Cases:**
    - fib(0) = 0
    - fib(1) = 1
  - **Recursive Step:** fib(n) = fib(n-1) + fib(n-2).
  - To generate the series, call this function in a loop from 0 to the desired number of terms.

Hard (1 x 30 = 30 Marks)

Q5. Linked List – Cycle Detection & Removal

- **Problem Description:** Given a singly linked list, detect if a cycle exists. If a cycle is found, remove it.
- **Solution Outline:**
  - **Cycle Detection (Floyd's Tortoise and Hare Algorithm):** Use two pointers, slow and fast, starting at the head. slow moves one step at a time, and fast moves two steps at a time. If a cycle exists, fast will eventually catch up to slow. If fast reaches NULL or its next is NULL, there is no cycle.
  - **Cycle Removal:** Once a cycle is detected (when slow and fast meet), move slow back to the head of the list. Keep fast where it is. Now, move both slow and fast one step at a time. The point where they meet again is the start of the loop. To break the cycle, keep a pointer to the node before the meeting point and set its next to NULL.