Java Most Asked Coding Questions for Interviews

Q #1) Write a Java Program to reverse a string without using String inbuilt function.

Answer:

```
public String reverseString(String input) {

    char[] chars = input.toCharArray();
    int left = 0, right = chars.length - 1;
    while (left < right) {
        char temp = chars[left];
        chars[left] = chars[right];
        chars[right] = temp;
        left++;
        right--;
    }
    return new String(chars);
}
```

Explanation: This solution manually swaps the characters of the string from the start and end, moving towards the center, effectively reversing the string without using any built-in functions.

Q #2) Write a Java Program to swap two numbers without using the third variable.

Answer:

```
public void swapNumbers(int a, int b) {
    a = a + b;
    b = a - b;
    a = a - b;
    System.out.println("After swap: a = " + a + ", b = " + b);
}
```

Explanation: This method uses arithmetic operations to swap two numbers without a temporary variable. It first adds the two numbers and stores the result in `a`, then subtracts `b` from the new `a` to recover the original `a` and assigns it to `b`, and finally subtracts the new `b` from the new `a` to recover the original `b`.

Q #3) Write a Java Program to count the number of words in a string using HashMap.

Answer:

```
public Map<String, Integer> countWords(String input) {
    Map<String, Integer> wordCount = new HashMap<>();
    String[] words = input.split("\\s+");
    for (String word : words) {
        wordCount.put(word, wordCount.getOrDefault(word, 0) + 1);
    }
    return wordCount;
}
```

Explanation: This solution splits the input string into words using a space delimiter, then uses a `HashMap` to count the occurrences of each word. The `getOrDefault` method is used to simplify the counting logic.

Q #4) Write a Java Program to iterate HashMap using While and advance for loop.

Answer:

```
public void iterateHashMap(Map<String, String> map) {
    // Using advanced for-loop
    for (Map.Entry<String, String> entry : map.entrySet()) {
        System.out.println(entry.getKey() + " -> " + entry.getValue());
    }

    // Using while-loop with iterator
    Iterator<Map.Entry<String, String>> iterator =
map.entrySet().iterator();
    while (iterator.hasNext()) {
        Map.Entry<String, String> entry = iterator.next();
        System.out.println(entry.getKey() + " -> " + entry.getValue());
    }
}
```

Explanation: This method shows two ways to iterate over a `HashMap`: using an enhanced for-loop to traverse the entry set, and using an iterator in a while-loop to perform the same task.

Q #5) Write a Java Program to find whether a number is prime or not in the most efficient way?

Answer:

```
public boolean isPrime(int num) {
    if (num <= 1) return false;
    if (num <= 3) return true;
    if (num % 2 == 0 || num % 3 == 0) return false;
    for (int i = 5; i * i <= num; i += 6) {
        if (num % i == 0 || num % (i + 2) == 0) return false;
    }
    return true;
}
```

Explanation: This function checks for divisibility using small primes and then iterates through potential factors up to the square root of the number, checking divisibility at 6k ± 1 intervals to efficiently determine if a number is prime.

Q #6) Write a Java Program to find whether a string or number is palindrome or not.

Answer:

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

Explanation: This method checks if a string is a palindrome by comparing characters from both ends moving toward the center. If all characters match, it's a palindrome.

Q #7) Write a Java Program for the Fibonacci series in recursion.

Answer:

```
public int fibonacci(int n) {
    if (n <= 1) return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```

Explanation: This recursive function computes Fibonacci numbers. The base cases return the number itself for `n = 0` or `1`, and the recursive case returns the sum of the two preceding numbers in the sequence.

Q #8) Write a Java Program to iterate ArrayList using for-loop, while-loop, and advance for-loop.

Answer:

```
public void iterateList(List<Integer> list) {
    // Using for-loop
    for (int i = 0; i < list.size(); i++) {
        System.out.println(list.get(i));
    }

    // Using while-loop
    int j = 0;
    while (j < list.size()) {
        System.out.println(list.get(j));
        j++;
    }

    // Using advanced for-loop
    for (int item : list) {
        System.out.println(item);
    }
}
```

Explanation: This method demonstrates three different ways to iterate through an `ArrayList`: using a traditional for-loop, a while-loop, and an enhanced for-loop.

Q #9) Write a Java Program to find the duplicate characters in a string.

Answer:

```
public void findDuplicates(String input) {
    HashMap<Character, Integer> charCount = new HashMap<>();
    for (char c : input.toCharArray()) {
        charCount.put(c, charCount.getOrDefault(c, 0) + 1);
    }
    for (Map.Entry<Character, Integer> entry : charCount.entrySet()) {
        if (entry.getValue() > 1) {
            System.out.println(entry.getKey() + " appears " +
entry.getValue() + " times");
        }
    }
}
```

Explanation: This solution uses a `HashMap` to count the occurrences of each character in the string. It then checks which characters have a count greater than one to identify duplicates.

Q #10) Write a Java Program to find the second-highest number in an array.

Answer:

```
public int secondHighest(int[] nums) {
    int highest = Integer.MIN_VALUE, secondHighest = Integer.MIN_VALUE;
    for (int num : nums) {
        if (num > highest) {
            secondHighest = highest;
            highest = num;
        } else if (num > secondHighest && num != highest) {
            secondHighest = num;
        }
    }
    return secondHighest;
}
```

Explanation: This method maintains two variables to track the highest and second-highest numbers. It iterates through the array once, updating these values appropriately to find the second-highest number.

Q #11) Write a Java Program to check Armstrong number.

Answer:

```
public boolean isArmstrong(int number) {
    int original = number, sum = 0;
    int digits = String.valueOf(number).length();
    while (number > 0) {
        int digit = number % 10;
        sum += Math.pow(digit, digits);
        number /= 10;
    }
    return sum == original;
}
```

Explanation: An Armstrong number is a number that is equal to the sum of its own digits each raised to the power of the number of digits. This function checks if the given number is an Armstrong number.

Q #12) Write a Java Program to remove all white spaces from a string without using replace().

Answer:

```
public String removeWhitespaces(String input) {
    StringBuilder result = new StringBuilder();
    for (int i = 0; i < input.length(); i++) {
        if (input.charAt(i) != ' ') {
            result.append(input.charAt(i));
        }
    }
    return result.toString();
}
```

Explanation: This method iterates through the string, appending only non-space characters to a `StringBuilder` to create the final string without spaces.

Q #13) Given an array of integers nums and an integer target, return indices of the two numbers such that they add up to target.

Solution:

```
public int[] twoSum(int[] nums, int target) {
    Map<Integer, Integer> numMap = new HashMap<>();
    for (int i = 0; i < nums.length; i++) {
        int complement = target - nums[i];
        if (numMap.containsKey(complement)) {
            return new int[] { numMap.get(complement), i };
        }
        numMap.put(nums[i], i);
    }
    throw new IllegalArgumentException("No two sum solution");
}
```

Explanation: This solution uses a hash map to track each element's complement (i.e., `target - nums[i]`). If a complement is found in the map, the indices of the current element and its complement are returned.

Q #14) Write a program that accepts comma-separated strings, sorts the strings in ascending order, and outputs the concatenated string of sorted strings.

Solution:

```
public String sortAndConcatenate(String input) {
    String[] parts = input.split(",");
    Arrays.sort(parts);
    return String.join("", parts);
}
```

Explanation: This method splits the input string into an array using commas as delimiters, sorts the array, and then concatenates the sorted strings into a single string.

Q #15) Given a string s, return true if s is a "good" string, or false otherwise. A string s is good if all characters that appear in s have the same number of occurrences (i.e., the same frequency).

Solution:

```
public boolean areOccurrencesEqual(String s) {
    int[] count = new int[26];  // There are 26 lowercase English letters
    for (char c : s.toCharArray()) {
        count[c - 'a']++;
    }

    int frequency = 0;
    for (int i = 0; i < 26; i++) {
        if (count[i] != 0) {
            if (frequency == 0) {
                frequency = count[i];  // Set the first non-zero frequency
found
            } else if (frequency != count[i]) {
                return false;  // Return false if any frequency doesn't
match the first found
            }
        }
    }
    return true;
}
```

Explanation: This solution creates an array to count occurrences of each letter in the string. It then checks if all non-zero counts are the same.

Q #16) Given an array nums and a value val, remove all instances of that value in-place and return the new length of the array. Do not allocate extra space for another array. You must modify the input array in-place with O(1) extra memory.

Solution:

```
public int removeElement(int[] nums, int val) {
    int i = 0;
    for (int j = 0; j < nums.length; j++) {
        if (nums[j] != val) {
            nums[i] = nums[j];
            i++;
        }
    }
    return i;
}
```

Explanation: This solution uses two pointers. The fast pointer `j` scans through the array, and the slow pointer `i` tracks the position of the next element to be replaced. If the current element is not equal to `val`, it is assigned to `nums[i]` and `i` is incremented.

Q #17) You are given an integer array nums and an array of queries queries where queries[i] = [val, index]. For each query, add val to nums[index]. Then, return the sum of all even numbers in nums.

Solution:

```
public int[] sumEvenAfterQueries(int[] nums, int[][] queries) {
    int sumEven = 0;
    for (int num : nums) {
        if (num % 2 == 0) sumEven += num;  // Calculate initial sum of even
numbers
    }

    int[] result = new int[queries.length];
    for (int i = 0; i < queries.length; i++) {
        int val = queries[i][0], index = queries[i][1];
        if (nums[index] % 2 == 0) sumEven -= nums[index];  // Remove old
value if it was even
        nums[index] += val;
        if (nums[index] % 2 == 0) sumEven += nums[index];  // Add new value
if it is even

        result[i] = sumEven;
    }

    return result;
}
```

Explanation: This solution first computes the sum of all even numbers in the original array. For each query, it adjusts the `sumEven` based on the old value at `nums[index]` (subtracting it if it was even) and the new value (adding it if it becomes even after modification). Each result is stored in the `result` array.

Q #18) Given two strings s and p, find all the start indices of p's anagrams in s.

Solution:

```
public List<Integer> findAnagrams(String s, String p) {
    List<Integer> result = new ArrayList<>();
    if (s.length() == 0 || p.length() > s.length()) return result;
    int[] charCount = new int[26];
    for (char c : p.toCharArray()) {
        charCount[c - 'a']++;
    }

    int start = 0, end = 0, count = p.length();
    while (end < s.length()) {
        if (charCount[s.charAt(end++) - 'a']-- >= 1) count--;

        if (count == 0) result.add(start);

        if (end - start == p.length() && charCount[s.charAt(start++) -
'a']++ >= 0) count++;
    }
    return result;
}
```

Explanation: This solution uses a sliding window approach with a character count array for `p`. As we expand the window, we decrease the count of characters. When the window size matches `p`'s length, if the count is zero, we know the current window is an anagram.

Q #19) Given a string s, find the length of the longest substring without repeating characters.

Solution:

```
public int lengthOfLongestSubstring(String s) {
    int[] chars = new int[128];  // There are 128 ASCII characters
    int left = 0, right = 0;
    int res = 0;
    while (right < s.length()) {
        char r = s.charAt(right);
        chars[r]++;

        while (chars[r] > 1) {
            char l = s.charAt(left);
            chars[l]--;
            left++;
        }

        res = Math.max(res, right - left + 1);
        right++;
    }
    return res;
}
```

Explanation: Using a sliding window approach, we expand the right boundary of our window until we encounter a repeating character. Then, we contract the left boundary until there are no duplicates in the window.

Q #20) Merge two sorted linked lists and return it as a new sorted list.

Solution:

```
public ListNode mergeTwoLists(ListNode l1, ListNode l2) {
    ListNode dummy = new ListNode(0);
    ListNode current = dummy;

    while (l1 != null && l2 != null) {
        if (l1.val < l2.val) {
            current.next = l1;
            l1 = l1.next;
        } else {
            current.next = l2;
            l2 = l2.next;
        }
        current = current.next;
    }

    current.next = (l1 != null) ? l1 : l2;
    return dummy.next;
}
```

Explanation: This solution creates a dummy node to facilitate the merge process. It iterates through both lists, appending the smaller node to the merged list, and finally attaches any remaining elements.

Q #21) You are given an n x n 2D matrix representing an image, rotate the image by 90 degrees (clockwise).

Solution:

```
public void rotate(int[][] matrix) {
    int n = matrix.length;

    // Transpose the matrix
    for (int i = 0; i < n; i++) {
        for (int j = i; j < n; j++) {
            int temp = matrix[j][i];
            matrix[j][i] = matrix[i][j];
            matrix[i][j] = temp;
        }
    }

    // Reverse each row
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n / 2; j++) {
            int temp = matrix[i][j];
            matrix[i][j] = matrix[i][n - 1 - j];
            matrix[i][n - 1 - j] = temp;
        }
    }
}
```

Explanation: The matrix is rotated by first transposing it (swapping rows with columns) and then reversing each row.

Q #22) Given a non-negative integer num, repeatedly add all its digits until the result has only one digit.

Solution:

```
public int addDigits(int num) {
    while (num >= 10) {
        int sum = 0;
        while (num > 0) {
            sum += num % 10;
            num /= 10;
        }
        num = sum;
    }
    return num;
}
```

Explanation: This solution repeatedly extracts and sums the digits of `num` until `num` becomes a single-digit number.

Q #23) Given an integer, write a function to determine if it is a power of two.

Solution:

```
public boolean isPowerOfTwo(int n) {
    return (n > 0) && ((n & (n - 1)) == 0);
}
```

Explanation: This solution uses a bit manipulation trick: a number `n` is a power of two if it has exactly one bit set to `1` in its binary representation. Using the expression `n & (n - 1)`, we can zero out the lowest set bit; if the result is `0`, then `n` was a power of two.

Q #24) Given an array nums, write a function to move all 0's to the end of it while maintaining the relative order of the non-zero elements.

Solution:

```
public void moveZeroes(int[] nums) {
    int insertPos = 0;
    for (int num : nums) {
        if (num != 0) nums[insertPos++] = num;
    }
    while (insertPos < nums.length) {
        nums[insertPos++] = 0;
    }
}
```

Explanation: This solution scans through the array with a two-pointer approach. It uses `insertPos` to store the next position for a non-zero element, effectively shifting non-zero values forward in the array. After moving all non-zero elements, it fills the rest of the array with zeros.

Q #25) Given an array nums of n integers where nums[i] is in the range [1, n], return an array of all the integers in the range [1, n] that do not appear in nums.

Solution:

```
public List<Integer> findDisappearedNumbers(int[] nums) {
    List<Integer> result = new ArrayList<>();
    for (int i = 0; i < nums.length; i++) {
        int val = Math.abs(nums[i]) - 1;
        if (nums[val] > 0) {
            nums[val] = -nums[val];
        }
    }
    for (int i = 0; i < nums.length; i++) {
        if (nums[i] > 0) {
            result.add(i + 1);
        }
    }
    return result;
}
```

Explanation: This solution marks each number that appears in the array by negating the value at its corresponding index (considering `1`-based indexing). In the second pass, it identifies the indices that contain positive numbers, indicating the numbers that didn't appear in the original array.