Arraylist Question

explain me problem statement with example in simple way and give java simple and optimizied code with time complexity

Basic-Level Questions:
Insert Element at a Specific Position in an ArrayList: Implement a function to insert an element at a given index.
Find the Minimum Element in an ArrayList: Write a function to find the minimum value in an ArrayList.
Remove Duplicates from an ArrayList: Given an ArrayList with duplicate entries, write a function to remove duplicates.
Check if an ArrayList is Sorted: Write a program to check if the elements in an ArrayList are sorted in ascending order.
Merge Two Sorted ArrayLists: Given two sorted ArrayLists, merge them into a single sorted ArrayList.
Shift ArrayList Elements by K Positions: Implement a function to rotate an ArrayList by K positions.

Memory updated

Let's go through each problem statement one by one with examples, simple explanations, and optimized Java code.

---

1. Insert Element at a Specific Position in an ArrayList

Problem Statement: Given an ArrayList and a specific index, insert a new element at that position.
Example:

- Input: `` `ArrayList = [1, 2, 4, 5]` ``, `` `element = 3` ``, `` `index = 2` ``
- Output: `` `[1, 2, 3, 4, 5]` ``

Explanation: Insert `` `3` `` at index `` `2` ``. The other elements are shifted to the right.
Java Code:

java

```java
import java.util.ArrayList;

public class ArrayListOperations {
    public static void insertElement(ArrayList<Integer> list, int element, int index) {
        list.add(index, element);
    }

    public static void main(String[] args) {
        ArrayList<Integer> list = new ArrayList<>();
        list.add(1);
        list.add(2);
        list.add(4);
        list.add(5);
        insertElement(list, 3, 2);
        System.out.println(list);  // Output: [1, 2, 3, 4, 5]
    }
}
```

Time Complexity: O(n) — Inserting an element at a specific index might require shifting elements.

---

2. Find the Minimum Element in an ArrayList

Problem Statement: Find the smallest element in an ArrayList.
Example:

- Input: `` `ArrayList = [5, 3, 8, 2, 7]` ``
- Output: `` `2` ``

Explanation: The smallest number in the list is `` `2` ``.
Java Code:

```java
import java.util.ArrayList;
import java.util.Collections;

public class ArrayListOperations {
    public static int findMinimum(ArrayList<Integer> list) {
        return Collections.min(list);  // Using built-in function
    }

    public static void main(String[] args) {
        ArrayList<Integer> list = new ArrayList<>();
        list.add(5);
        list.add(3);
        list.add(8);
        list.add(2);
        list.add(7);

        System.out.println(findMinimum(list));  // Output: 2
    }
}
```

Time Complexity: O(n) — The entire list is traversed to find the minimum.

---

3. Remove Duplicates from an ArrayList

Problem Statement: Remove duplicate entries from an ArrayList.
Example:

- Input: `` `ArrayList = [1, 2, 2, 3, 4, 4, 5]` ``
- Output: `` `[1, 2, 3, 4, 5]` ``

Explanation: Remove all duplicate entries so that each element appears only once.
Java Code:

java

```java
import java.util.ArrayList;
import java.util.LinkedHashSet;

public class ArrayListOperations {
    public static ArrayList<Integer> removeDuplicates(ArrayList<Integer> list) {
        LinkedHashSet<Integer> set = new LinkedHashSet<>(list);  // Maintain insertion
order
        return new ArrayList<>(set);
    }

    public static void main(String[] args) {
        ArrayList<Integer> list = new ArrayList<>();
        list.add(1);
        list.add(2);
        list.add(2);
        list.add(3);
        list.add(4);
        list.add(4);
        list.add(5);

        System.out.println(removeDuplicates(list));  // Output: [1, 2, 3, 4, 5]
    }
}
```

Time Complexity: O(n) — Uses a LinkedHashSet to eliminate duplicates while maintaining order.

---

4. Check if an ArrayList is Sorted

Problem Statement: Check if the ArrayList is sorted in ascending order.
Example:

- Input: `ArrayList = [1, 2, 3, 4, 5]`
- Output: `true`
- Input: `ArrayList = [1, 3, 2, 4, 5]`
- Output: `false`

Explanation: Compare each element with the next one and verify if it is greater or equal.
Java Code:

java

```java
import java.util.ArrayList;

public class ArrayListOperations {
    public static boolean isSorted(ArrayList<Integer> list) {
        for (int i = 0; i < list.size() - 1; i++) {
            if (list.get(i) > list.get(i + 1)) {
                return false;
            }
        }
        return true;
    }

    public static void main(String[] args) {
        ArrayList<Integer> list = new ArrayList<>();
        list.add(1);
        list.add(2);
        list.add(3);
        list.add(4);
        list.add(5);
        System.out.println(isSorted(list));  // Output: true
    }
}
```

Time Complexity: O(n) — The list is traversed once.

---

5. Merge Two Sorted ArrayLists

Problem Statement: Given two sorted ArrayLists, merge them into one sorted list.
Example:

- Input: `` `ArrayList1 = [1, 3, 5]` ``, `` `ArrayList2 = [2, 4, 6]` ``
- Output: `` `[1, 2, 3, 4, 5, 6]` ``

Explanation: Merging the two lists while maintaining order.
Java Code:

```java
import java.util.ArrayList;
import java.util.Collections;

public class ArrayListOperations {
    public static ArrayList<Integer> mergeSortedLists(ArrayList<Integer> list1,
ArrayList<Integer> list2) {
        list1.addAll(list2);  // Combine both lists
        Collections.sort(list1);  // Sort the merged list
        return list1;
    }

    public static void main(String[] args) {
        ArrayList<Integer> list1 = new ArrayList<>();
        list1.add(1);
        list1.add(3);
        list1.add(5);

        ArrayList<Integer> list2 = new ArrayList<>();
        list2.add(2);
        list2.add(4);
        list2.add(6);

        System.out.println(mergeSortedLists(list1, list2));  // Output: [1, 2, 3, 4, 5, 6]
    }
}
```

Time Complexity: O((n+m) log(n+m)) — Sorting is required after merging both lists.

---

6. Shift ArrayList Elements by K Positions

Problem Statement: Rotate the elements in an ArrayList by `` `K` `` positions.
Example:

- Input: `` `ArrayList = [1, 2, 3, 4, 5]` ``, `` `K = 2` ``
- Output: `` `[4, 5, 1, 2, 3]` ``

Explanation: Rotate the list such that elements shift by `` `K` `` positions to the right.

Java Code:

```java
import java.util.ArrayList;
import java.util.Collections;

public class ArrayListOperations {
    public static void rotate(ArrayList<Integer> list, int k) {
        int n = list.size();
        k = k % n;  // Handle cases where K > n
        Collections.rotate(list, k);  // Rotate by k positions
    }

    public static void main(String[] args) {
        ArrayList<Integer> list = new ArrayList<>();
        list.add(1);
        list.add(2);
        list.add(3);
        list.add(4);
        list.add(5);

        rotate(list, 2);
        System.out.println(list);  // Output: [4, 5, 1, 2, 3]
    }
}
```

Time Complexity: O(n) — Each element is moved once during rotation.

Let me know if you need any clarifications or additional optimizations!

Intermediate-Level Questions:
Find the Second Largest Element in an ArrayList: Write a program to find the second largest element in an ArrayList.
Find the Missing Number in an ArrayList (1 to N): Given an ArrayList with numbers from 1 to N with one missing, find the missing number.
Move Zeros to the End of the ArrayList: Write a function that moves all the zeros in an ArrayList to the end without changing the order of non-zero elements.
Intersection of Two ArrayLists: Write a program to find the intersection of two ArrayLists.
Find All Pairs with a Given Sum: Find all pairs in an ArrayList that sum up to a target value.
Find the Longest Consecutive Sequence in an ArrayList: Given an ArrayList, find the length of the longest consecutive elements sequence.

Let's tackle these intermediate-level questions with examples and Java code.

---

1. Find the Second Largest Element in an ArrayList

Problem Statement: Write a program to find the second largest element in an ArrayList.
Example:

- Input: `ArrayList = [5, 1, 9, 3, 8]`
- Output: `8`

Explanation: The largest element is `9`, and the second largest is `8`.
Java Code:

```java
import java.util.ArrayList;
import java.util.Collections;

public class ArrayListOperations {
    public static Integer findSecondLargest(ArrayList<Integer> list) {
        if (list.size() < 2) {
            return null;  // Not enough elements
        }
        Collections.sort(list, Collections.reverseOrder());  // Sort in descending order
        return list.get(1);  // Second largest
    }

    public static void main(String[] args) {
        ArrayList<Integer> list = new ArrayList<>();
        list.add(5);
        list.add(1);
        list.add(9);
        list.add(3);
        list.add(8);

        System.out.println(findSecondLargest(list));  // Output: 8
    }
}
```

Time Complexity: O(n log n) — Sorting the list is the most time-consuming operation.

---

2. Find the Missing Number in an ArrayList (1 to N)

Problem Statement: Given an ArrayList containing numbers from `1` to `N` with one missing, find the missing number.
Example:

- Input: `ArrayList = [1, 2, 4, 5]` with `N = 5`
- Output: `3`

Explanation: The list should contain numbers `1` to `5`. The missing number is `3`.
Java Code:

```java
import java.util.ArrayList;

public class ArrayListOperations {
    public static int findMissingNumber(ArrayList<Integer> list, int n) {
        int totalSum = n * (n + 1) / 2;  // Sum of first N natural numbers
        int currentSum = 0;
        for (int num : list) {
            currentSum += num;
        }
        return totalSum - currentSum;
    }

    public static void main(String[] args) {
        ArrayList<Integer> list = new ArrayList<>();
        list.add(1);
        list.add(2);
        list.add(4);
        list.add(5);

        System.out.println(findMissingNumber(list, 5));  // Output: 3
    }
}
```

Time Complexity: O(n) — Sum computation and iteration through the list.

---

3. Move Zeros to the End of the ArrayList

Problem Statement: Move all the zeros in an ArrayList to the end without changing the order of non-zero elements.
Example:

- Input: ``ArrayList = [0, 1, 2, 0, 3, 0, 4]``
- Output: ``[1, 2, 3, 4, 0, 0, 0]``

Explanation: All zeros should be moved to the end while maintaining the order of non-zero elements.
Java Code:

```java
import java.util.ArrayList;

public class ArrayListOperations {
    public static void moveZerosToEnd(ArrayList<Integer> list) {
        int count = 0;  // Count of non-zero elements

        // Traverse the list and move non-zero elements to the front
        for (int i = 0; i < list.size(); i++) {
            if (list.get(i) != 0) {
                list.set(count++, list.get(i));
            }
        }

        // Fill the rest of the list with zeros
        while (count < list.size()) {
            list.set(count++, 0);
        }
    }

    public static void main(String[] args) {
        ArrayList<Integer> list = new ArrayList<>();
        list.add(0);
        list.add(1);
        list.add(2);
        list.add(0);
        list.add(3);
        list.add(0);
        list.add(4);

        moveZerosToEnd(list);
        System.out.println(list);  // Output: [1, 2, 3, 4, 0, 0, 0]
    }
}
```

Time Complexity: O(n) — Each element is visited once.

---

4. Intersection of Two ArrayLists

Problem Statement: Find the intersection of two ArrayLists, i.e., common elements between them.
Example:

- Input: `ArrayList1 = [1, 2, 3, 4]`, `ArrayList2 = [3, 4, 5, 6]`
- Output: `[3, 4]`

Explanation: Common elements in both lists are `3` and `4`.
Java Code:

```java
import java.util.ArrayList;
import java.util.HashSet;

public class ArrayListOperations {
    public static ArrayList<Integer> intersection(ArrayList<Integer> list1,
ArrayList<Integer> list2) {
        HashSet<Integer> set = new HashSet<>(list1);
        ArrayList<Integer> result = new ArrayList<>();
        for (int num : list2) {
            if (set.contains(num)) {
                result.add(num);
                set.remove(num);  // To avoid duplicate entries in the result
            }
        }
        return result;
    }

    public static void main(String[] args) {
        ArrayList<Integer> list1 = new ArrayList<>();
        list1.add(1);
        list1.add(2);
        list1.add(3);
        list1.add(4);

        ArrayList<Integer> list2 = new ArrayList<>();
        list2.add(3);
        list2.add(4);
        list2.add(5);
        list2.add(6);

        System.out.println(intersection(list1, list2));  // Output: [3, 4]
    }
}
```

Time Complexity: O(n + m) — HashSet operations are O(1) on average, where `n` and `m` are sizes of the two lists.

---

5. Find All Pairs with a Given Sum

Problem Statement: Find all pairs in an ArrayList that sum up to a target value.
Example:

- Input: `ArrayList = [1, 2, 3, 4, 3]`, `target = 6`
- Output: `[(2, 4), (3, 3)]`

Explanation: The pairs that add up to `6` are `(2, 4)` and `(3, 3)`.
Java Code:

```java
import java.util.ArrayList;
import java.util.HashSet;

public class ArrayListOperations {
    public static ArrayList<ArrayList<Integer>> findPairsWithSum(ArrayList<Integer> list,
int target) {
        HashSet<Integer> seen = new HashSet<>();
        ArrayList<ArrayList<Integer>> result = new ArrayList<>();
        for (int num : list) {
            int complement = target - num;
            if (seen.contains(complement)) {
                ArrayList<Integer> pair = new ArrayList<>();
                pair.add(complement);
                pair.add(num);
                result.add(pair);
            }
            seen.add(num);
        }
        return result;
    }

    public static void main(String[] args) {
        ArrayList<Integer> list = new ArrayList<>();
        list.add(1);
        list.add(2);
        list.add(3);
        list.add(4);
        list.add(3);

        System.out.println(findPairsWithSum(list, 6));  // Output: [[2, 4], [3, 3]]
    }
}
```

Time Complexity: O(n) — Each element is processed once, with set operations being O(1) on average.

---

6. Find the Longest Consecutive Sequence in an ArrayList

Problem Statement: Given an ArrayList, find the length of the longest consecutive elements sequence.
Example:

- Input: `ArrayList = [100, 4, 200, 1, 3, 2]`
- Output: `4` (Sequence is `[1, 2, 3, 4]`)

Explanation: The longest consecutive sequence is `[1, 2, 3, 4]`, which has length `4`.
Java Code:

```java
import java.util.ArrayList;
import java.util.HashSet;

public class ArrayListOperations {
    public static int longestConsecutiveSequence(ArrayList<Integer> list) {
        HashSet<Integer> set = new HashSet<>(list);
        int longestStreak = 0;

        for (int num : set) {
            if (!set.contains(num - 1)) {  // Start of a sequence
                int currentNum = num;
                int currentStreak = 1;
                while (set.contains(currentNum + 1)) {
                    currentNum++;
                    currentStreak++;
                }

                longestStreak = Math.max(longestStreak, currentStreak);
            }
        }

        return longestStreak;
    }

    public static void main(String[] args) {
        ArrayList<Integer> list = new ArrayList<>();
        list.add(100);
        list.add(4);
        list.add(200);
        list.add(1);
        list.add(3);
        list.add(2);

        System.out.println(longestConsecutiveSequence(list));  // Output: 4
    }
}
```

Time Complexity: O(n) — Each element is processed once with set operations being O(1) on average.

Feel free to ask if you have any further questions or need more details!

Advanced-Level Questions:
Find Triplets in an ArrayList That Sum to a Given Value: Given an ArrayList of integers, find all triplets that sum up to a specific value.
Subarray with Given Sum in an ArrayList: Given an ArrayList, find a contiguous subarray whose sum equals a given number.
Find the Leader Elements in an ArrayList: A leader element is greater than all the elements to its right. Find all the leader elements in the ArrayList.
Find the Smallest Subarray with a Given Sum: Given an ArrayList, find the smallest contiguous subarray whose sum is greater than or equal to a given value.
ArrayList of ArrayLists: Find the Maximum Sum Subarray: Given a 2D ArrayList (ArrayList of ArrayLists), find the subarray with the maximum sum.
Find All Combinations of ArrayList Elements: Given an ArrayList, find all possible combinations of elements (similar to powerset generation).
Find the First Repeating Element in an ArrayList: Write a program to find the first repeating element in an ArrayList.
Find the Maximum Product of Two Elements in an ArrayList: Given an ArrayList of integers, find the maximum product of any two elements.

Let's delve into these advanced-level questions with examples, explanations, and Java code.

---

1. Find Triplets in an ArrayList That Sum to a Given Value

Problem Statement: Given an ArrayList of integers, find all unique triplets that sum up to a specific value.
Example:

- Input: `ArrayList = [1, 2, -1, -2, 3, 0]`, `target = 0`
- Output: `[[-2, -1, 3], [-2, 0, 2], [-1, 0, 1]]`

Explanation: Triplets in the list that sum to `0` are `[[-2, -1, 3], [-2, 0, 2], [-1, 0, 1]]`.
Java Code:

```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

public class ArrayListOperations {
    public static List<List<Integer>> findTriplets(ArrayList<Integer> list, int target) {
        List<List<Integer>> triplets = new ArrayList<>();
        int n = list.size();
        Collections.sort(list);

        for (int i = 0; i < n - 2; i++) {
            if (i > 0 && list.get(i).equals(list.get(i - 1))) continue;  // Skip duplicates

            int left = i + 1;
            int right = n - 1;
            while (left < right) {
                int sum = list.get(i) + list.get(left) + list.get(right);
                if (sum == target) {
                    triplets.add(Arrays.asList(list.get(i), list.get(left), list.get(right)));
                    while (left < right && list.get(left).equals(list.get(left + 1))) left++;  // Skip duplicates
                    while (left < right && list.get(right).equals(list.get(right - 1))) right--;  // Skip duplicates
                    left++;
                    right--;
                } else if (sum < target) {
                    left++;
                } else {
                    right--;
                }
            }
        }
        return triplets;
    }

    public static void main(String[] args) {
        ArrayList<Integer> list = new ArrayList<>(Arrays.asList(1, 2, -1, -2, 3, 0));
        System.out.println(findTriplets(list, 0));  // Output: [[-2, -1, 3], [-2, 0, 2], [-1, 0, 1]]
    }
}
```

Time Complexity: O(n^2) — Sorting takes O(n log n) and finding triplets takes O(n^2).

---

2. Subarray with Given Sum in an ArrayList

Problem Statement: Given an ArrayList, find a contiguous subarray whose sum equals a given number.
Example:

- Input: `ArrayList = [1, 2, 3, 7, 5]`, `target = 12`
- Output: `[2, 3, 7]` (starting from index `1` to `3`)

Explanation: The subarray `[2, 3, 7]` sums up to `12`.
Java Code:

```java
import java.util.ArrayList;

public class ArrayListOperations {
    public static ArrayList<Integer> findSubarrayWithSum(ArrayList<Integer> list, int
target) {
        int start = 0;
        int sum = 0;

        for (int end = 0; end < list.size(); end++) {
            sum += list.get(end);

            while (sum > target && start <= end) {
                sum -= list.get(start++);
            }

            if (sum == target) {
                return new ArrayList<>(list.subList(start, end + 1));
            }
        }
        return new ArrayList<>();  // No such subarray found
    }

    public static void main(String[] args) {
        ArrayList<Integer> list = new ArrayList<>(Arrays.asList(1, 2, 3, 7, 5));
        System.out.println(findSubarrayWithSum(list, 12));  // Output: [2, 3, 7]
    }
}
```

Time Complexity: O(n) — Each element is processed once.

---

3. Find the Leader Elements in an ArrayList

Problem Statement: A leader element is greater than all the elements to its right. Find all leader elements in the ArrayList.
Example:

- Input: `ArrayList = [16, 17, 4, 3, 5, 2]`
- Output: `[17, 5, 2]`

Explanation: The leader elements are `17`, `5`, and `2`.
Java Code:

```java
import java.util.ArrayList;
import java.util.Collections;
```

```
public class ArrayListOperations {
    public static ArrayList<Integer> findLeaders(ArrayList<Integer> list) {
        ArrayList<Integer> leaders = new ArrayList<>();
        int n = list.size();
        int maxFromRight = list.get(n - 1);
        leaders.add(maxFromRight);

        for (int i = n - 2; i >= 0; i--) {
            if (list.get(i) > maxFromRight) {
                maxFromRight = list.get(i);
                leaders.add(maxFromRight);
            }
        }

        Collections.reverse(leaders);  // Reverse to maintain original order
        return leaders;
    }

    public static void main(String[] args) {
        ArrayList<Integer> list = new ArrayList<>(Arrays.asList(16, 17, 4, 3, 5, 2));
        System.out.println(findLeaders(list));  // Output: [17, 5, 2]
    }
}
```

Time Complexity: O(n) — Each element is processed once.

---

4. Find the Smallest Subarray with a Given Sum

Problem Statement: Given an ArrayList, find the smallest contiguous subarray whose sum is greater than or equal to a given value.
Example:

- Input: `ArrayList = [1, 4, 4]`, `target = 8`
- Output: `[4, 4]` (length `2`)

Explanation: The smallest subarray `[4, 4]` sums to `8`.
Java Code:

```java
import java.util.ArrayList;

public class ArrayListOperations {
    public static ArrayList<Integer> findSmallestSubarrayWithSum(ArrayList<Integer> list,
int target) {
        int minLength = Integer.MAX_VALUE;
        int start = 0;
        int sum = 0;
        ArrayList<Integer> smallestSubarray = new ArrayList<>();

        for (int end = 0; end < list.size(); end++) {
            sum += list.get(end);

            while (sum >= target) {
                if (end - start + 1 < minLength) {
                    minLength = end - start + 1;
                    smallestSubarray = new ArrayList<>(list.subList(start, end + 1));
                }
                sum -= list.get(start++);
            }
        }

        return smallestSubarray;
    }
    public static void main(String[] args) {
        ArrayList<Integer> list = new ArrayList<>(Arrays.asList(1, 4, 4));
        System.out.println(findSmallestSubarrayWithSum(list, 8));  // Output: [4, 4]
    }
}
```

Time Complexity: O(n) — Each element is processed once.

---

5. Find the Maximum Sum Subarray in a 2D ArrayList

Problem Statement: Given a 2D ArrayList, find the subarray (submatrix) with the maximum sum.
Example:

- Input: `` `2D ArrayList = [[1, 2, -1, -4, -20], [-8, -3, 4, 2, 1], [3, 8, 10, 1, 3], [-4, -1, 1, 7, -6]]` ``
- Output: The maximum sum submatrix will be printed (actual values depend on the array).

Explanation: Use Kadane's algorithm on rows and columns to find the maximum sum submatrix.
Java Code:

```java
import java.util.ArrayList;

public class ArrayListOperations {
    public static int maxSumSubmatrix(ArrayList<ArrayList<Integer>> matrix) {
        int rows = matrix.size();
        int cols = matrix.get(0).size();
        int maxSum = Integer.MIN_VALUE;

        for (int left = 0; left < cols; left++) {
            int[] temp = new int[rows];
            for (int right = left; right < cols; right++) {
                for (int i = 0; i < rows; i++) {
                    temp[i] += matrix.get(i).get(right);
                }

                maxSum = Math.max(maxSum, kadane(temp));
            }
        }

        return maxSum;
    }

    private static int kadane(int[] arr) {
        int maxSoFar = Integer.MIN_VALUE;
        int maxEndingHere = 0;

        for (int num : arr) {
            maxEndingHere += num;
            maxSoFar = Math.max(maxSoFar, maxEndingHere);
            if (maxEndingHere < 0) {
                maxEndingHere = 0;
            }
        }

        return maxSoFar;
    }

    public static void main(String[] args) {
        ArrayList<ArrayList<Integer>> matrix = new ArrayList<>();
        matrix.add(new ArrayList<>(Arrays.asList(1, 2, -1, -4, -20)));
        matrix.add(new ArrayList<>(Arrays.asList(-8, -3, 4, 2, 1)));
        matrix.add(new ArrayList<>(Arrays.asList(3, 8, 10, 1, 3)));
        matrix.add(new ArrayList<>(Arrays.asList(-4, -1, 1, 7, -6)));
        System.out.println(maxSumSubmatrix(matrix));  // Output: The maximum sum submatrix
value
    }
}
```

Time Complexity: O(n^3) — Due to the combination of Kadane's algorithm and nested loops for submatrices.

---

6. Find All Combinations of ArrayList Elements

Problem Statement: Given an ArrayList, find all possible combinations of elements (similar to powerset generation).
Example:

- Input: `` `ArrayList = [1, 2]` ``
- Output: `` `[[], [1], [2], [1, 2]]` ``

Explanation: All possible subsets of the list are generated.
Java Code:

```java
import java.util.ArrayList;

public class ArrayListOperations {
    public static ArrayList<ArrayList<Integer>> findAllCombinations(ArrayList<Integer>
list) {
        ArrayList<ArrayList<Integer>> result = new ArrayList<>();
        generateCombinations(list, 0, new ArrayList<>(), result);
        return result;
    }

    private static void generateCombinations(ArrayList<Integer> list, int start,
ArrayList<Integer> current, ArrayList<ArrayList<Integer>> result) {
        result.add(new ArrayList<>(current));
        for (int i = start; i < list.size(); i++) {
            current.add(list.get(i));
            generateCombinations(list, i + 1, current, result);
            current.remove(current.size() - 1);
        }
    }

    public static void main(String[] args) {
        ArrayList<Integer> list = new ArrayList<>(Arrays.asList(1, 2));
        System.out.println(findAllCombinations(list));  // Output: [[], [1], [2], [1, 2]]
    }
}
```

Time Complexity: O(2^n) — There are 2^n subsets for a list of size n.

---

7. Find the First Repeating Element in an ArrayList

Problem Statement: Write a program to find the first repeating element in an ArrayList.
Example:

- Input: `` `ArrayList = [1, 2, 3, 4, 2, 5, 3]` ``
- Output: `` `2` ``

Explanation: The first element that repeats is `2`.
Java Code:

```java
import java.util.ArrayList;
import java.util.HashSet;

public class ArrayListOperations {
    public static Integer findFirstRepeatingElement(ArrayList<Integer> list) {
        HashSet<Integer> seen = new HashSet<>();
        for (int num : list) {
            if (seen.contains(num)) {
                return num;
            }
            seen.add(num);
        }
        return null;  // No repeating element
    }

    public static void main(String[] args) {
        ArrayList<Integer> list = new ArrayList<>(Arrays.asList(1, 2, 3, 4, 2, 5, 3));
        System.out.println(findFirstRepeatingElement(list));  // Output: 2
    }
}
```

Time Complexity: O(n) — Each element is processed once.

---

8. Find the Maximum Product of Two Elements in an ArrayList

Problem Statement: Given an ArrayList of integers, find the maximum product of any two elements.
Example:

- Input: `ArrayList = [1, 10, 3, 7, 9]`
- Output: `90` (Product of `10` and `9`)

Explanation: The maximum product of any two elements is `90` (from `10` and `9`).
Java Code:

```java
import java.util.ArrayList;

public class ArrayListOperations {
    public static int findMaximumProduct(ArrayList<Integer> list) {
        int max1 = Integer.MIN_VALUE;
        int max2 = Integer.MIN_VALUE;

        for (int num : list) {
            if (num > max1) {
                max2 = max1;
                max1 = num;
            } else if (num > max2) {
                max2 = num;
            }
        }

        return max1 * max2;
    }

    public static void main(String[] args) {
        ArrayList<Integer> list = new ArrayList<>(Arrays.asList(1, 10, 3, 7, 9));
        System.out.println(findMaximumProduct(list));  // Output: 90
    }
}
```

Time Complexity: O(n) — Each element is processed once.

Feel free to ask if you need any more details or additional questions!

Is this conversation helpful so far?

ChatGPT can make mistakes. Check important info.