Problem 1:

Problem Description:

We are tasked with developing a program to use the Luhn check to verify credit card numbers in this scenario. Financial transactions rely primarily on credit card numbers, and verifying their validity is vital for preventing mishaps and fraud. An algorithm for confirming the validity of credit card numbers is a must and is done by this check.

The problem is important because accurate credit card number verification is necessary for businesses, consumers, and financial institutions. Implementing and understanding the Luhn check method not only aids in the resolution of this issue but also throws light on the practical applications of mathematical algorithms. We can improve our problem-solving abilities by using the knowledge we learn from this challenge to complete other validation jobs that are comparable.

Analysis:

We employ the Luhn check algorithm, which includes the following steps, to resolve this issue:

1. If the answer is a two-digit number, double each subsequent digit from right to left and add the digits.
2. Add up all of the single digit figures you got in step one.
3. From right to left, add all the digits that are in the odd positions.
4. Add the outcomes from steps 2 and 3.
5. Verify that step 4's total is divisible by 10. The credit card number is legitimate if it is; else, it is not.

Difficulties encountered in this problem:

1. Applying the doubling and summing logic to digits in even positions.
2. Handling special circumstances, such as credit card numbers with varied first digits for different card issuers.

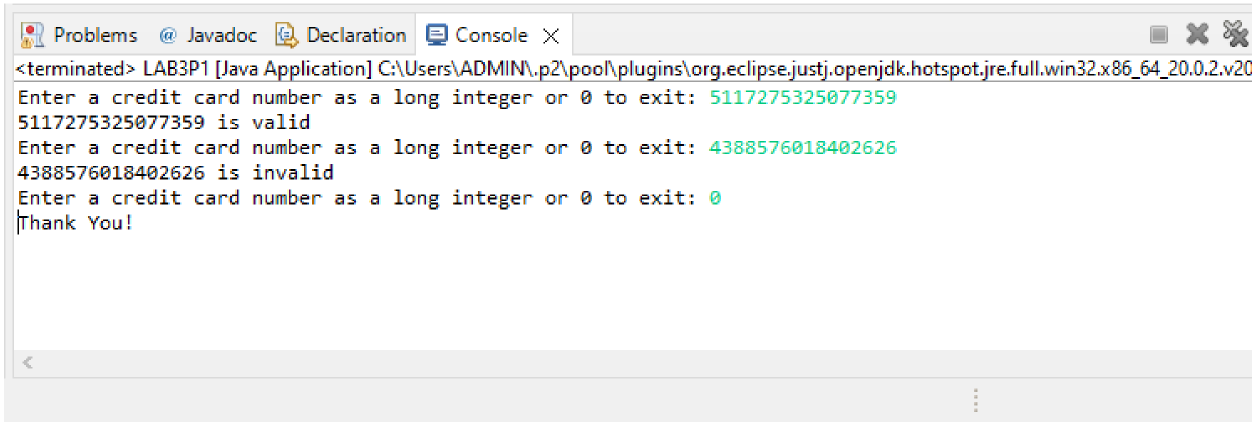
A more effective solution would be speeding up the code or giving more thorough error messages to explain which step of the validation process failed. However, the fundamental Luhn check technique is the cornerstone of any solution and is essential for validating credit card numbers.

Source Code:

```
package edu.northeastern.csye6200;
import java.util.Scanner;

public class LAB3P1 {
	public static void main(String[] args) {
		Scanner input = new Scanner(System.in);
	  while (true) {
            System.out.print("Enter a credit card number as a long integer or 0 to 
exit: ");
            long cardDetails = input.nextLong();
            if (cardDetails == 0) {
                System.out.println("Thank You!");
                break;
            }
            if (isValid(cardDetails)) {
                System.out.println(cardDetails + " is valid");
            } else {
                System.out.println(cardDetails + " is invalid");
            }
	  }
	  input.close();
	}

	public static boolean isValid(long number) {
		return (getSize(number) >= 13 && getSize(number) <= 16) &&
				(prefixMatched(number, 4) || prefixMatched(number, 5) ||
						prefixMatched(number, 37) ||
prefixMatched(number, 6)) &&
				((sumOfDoubleEvenPlace(number) + sumOfOddPlace(number)) % 
10 == 0);
	}

	public static int sumOfDoubleEvenPlace(long number) {
		int sum = 0;
	  String num = Long.toString(number);
	  for (int i = num.length() - 2; i >= 0; i -= 2) {
	      sum = sum + getDigit(Integer.parseInt(num.charAt(i) + "") * 2);
	  }
	  return sum;
	}

	public static int getDigit(int number) {
		if (number < 9)
	      return number;
	  return number / 10 + number % 10;
	}

	public static int sumOfOddPlace(long number) {
		int sum = 0;
	  String num = Long.toString(number);
	  for (int i = num.length() - 1; i >= 0; i -= 2) {
```

```
            sum = sum + Integer.parseInt(num.charAt(i) + "");
        }
        return sum;
    }

    public static boolean prefixMatched(long number, int d) {
            return getPrefix(number, getSize(d)) == d;
    }

    public static int getSize(long d) {
            return Long.toString(d).length();
    }

    public static long getPrefix(long number, int k) {
            String x = Long.toString(number);
        if (x.length() > k) {
            return Long.parseLong(x.substring(0, k));
        }
        return number;
    }
}
```

Screenshots of Sample Runs:



```
<terminated> LAB3P1 [Java Application] C:\Users\ADMIN\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_20.0.2.v20
Enter a credit card number as a long integer or 0 to exit: 5117275325077359
5117275325077359 is valid
Enter a credit card number as a long integer or 0 to exit: 4388576018402626
4388576018402626 is invalid
Enter a credit card number as a long integer or 0 to exit: 0
Thank You!
```

Problem 2:

Problem Description:

In this problem, our aim is to write a program that determines whether an array has four numbers that are consecutively the same value. This problem is relevant in a number of applications, including data analysis, where it is crucial to recognize patterns and sequences.

This issue is significant because it tests our capacity to create a function that can find patterns in data. Understanding array traversal and pattern recognition, which are abilities that may be applied to other programming tasks, is a prerequisite for the answer. This topic also challenges us to consider the effectiveness of our method, particularly when working with enormous datasets.

Analysis:

Implementing the 'isConsecutiveFour' method, which determines whether an array has four consecutive numbers with the same value, can help us fix this issue. The crucial phases involve iterating through the array and comparing neighboring elements to find instances of the same value that occur repeatedly.

Difficulties encountered in this problem:

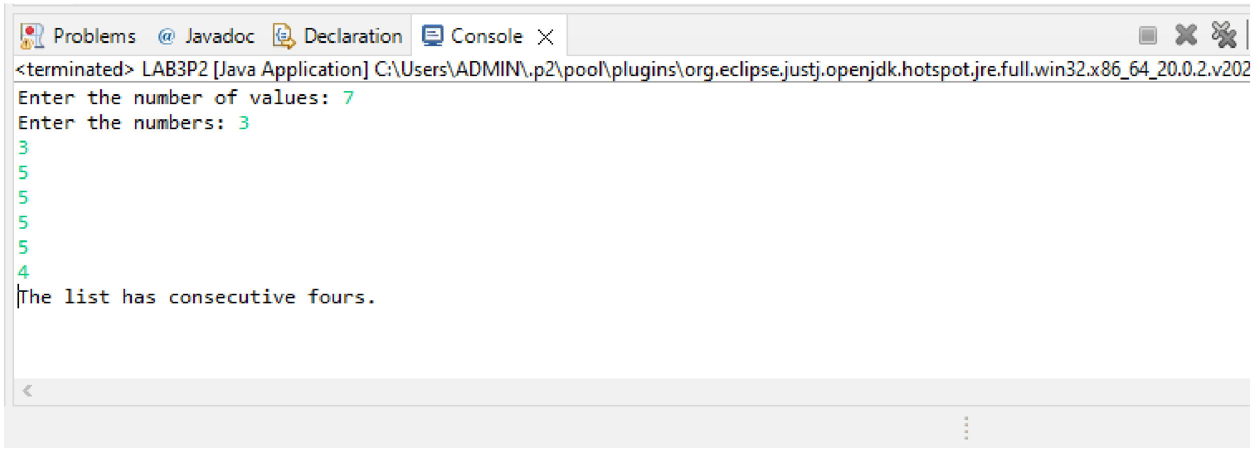
1. Selecting the best loop structure to efficiently traverse the array.
2. Taking care of situations where the array might have fewer than four members or if the string of fours is either at the start or end of the array.

A potential optimization: If four consecutive values are detected, it may be more efficient to end the loop early rather than inspecting the full array.

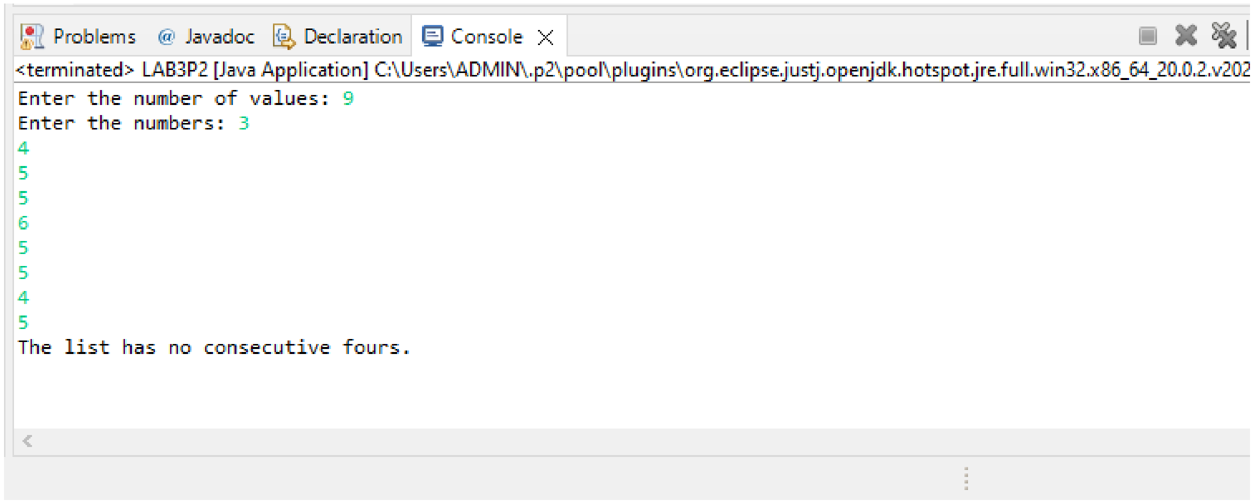
Source Code:

```
package edu.northeastern.csye6200;
import java.util.Scanner;

public class LAB3P2 {
	public static void main(String[] args) {
		 Scanner input = new Scanner(System.in);
	  System.out.print("Enter the number of values: ");
	  int x = input.nextInt();
	  int[] y = new int[x];
	  System.out.print("Enter the numbers: ");
	  for (int i = 0; i < x; i++) {
	      y[i] = input.nextInt();
	  }
	  if (isConsecutiveFour(y)) {
	      System.out.println("The list has consecutive fours.");
	  } else {
	      System.out.println("The list has no consecutive fours.");
	  }
	  input.close();
    }

	public static boolean isConsecutiveFour(int[] values) {
		 if (values.length < 4) {
	      return false;
	  }
	  for (int i = 0; i <= values.length - 4; i++) {
	      if (values[i] == values[i + 1] && values[i] == values[i + 2] && values[i]
== values[i + 3]) {
	          return true;
	      }
	  }
	  return false;
    }

}
```

Screenshots of Sample Runs:



```
<terminated> LAB3P2 [Java Application] C:\Users\ADMIN\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_20.0.2.v202
Enter the number of values: 7
Enter the numbers: 3
3
5
5
5
5
4
The list has consecutive fours.
```



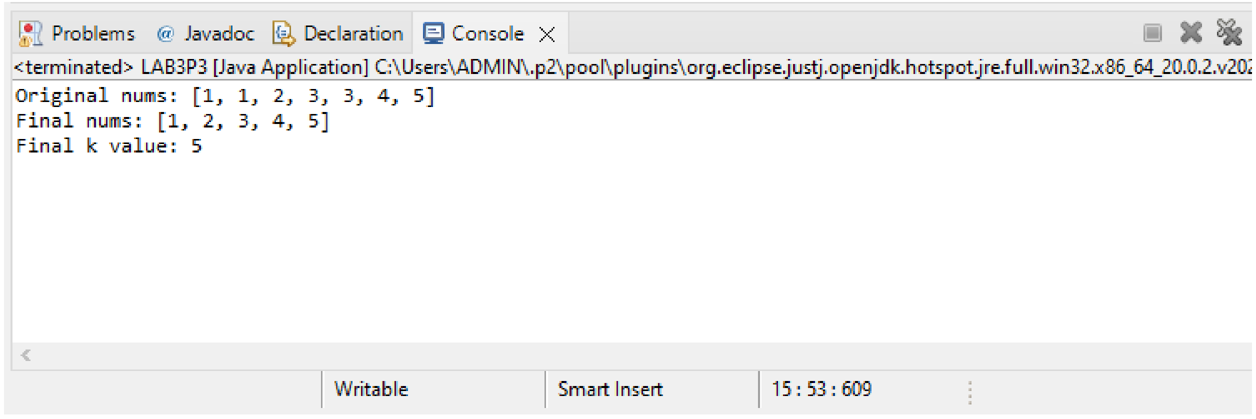
```
<terminated> LAB3P2 [Java Application] C:\Users\ADMIN\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_20.0.2.v202
Enter the number of values: 9
Enter the numbers: 3
4
5
5
6
5
5
4
5
The list has no consecutive fours.
```

Problem 3:

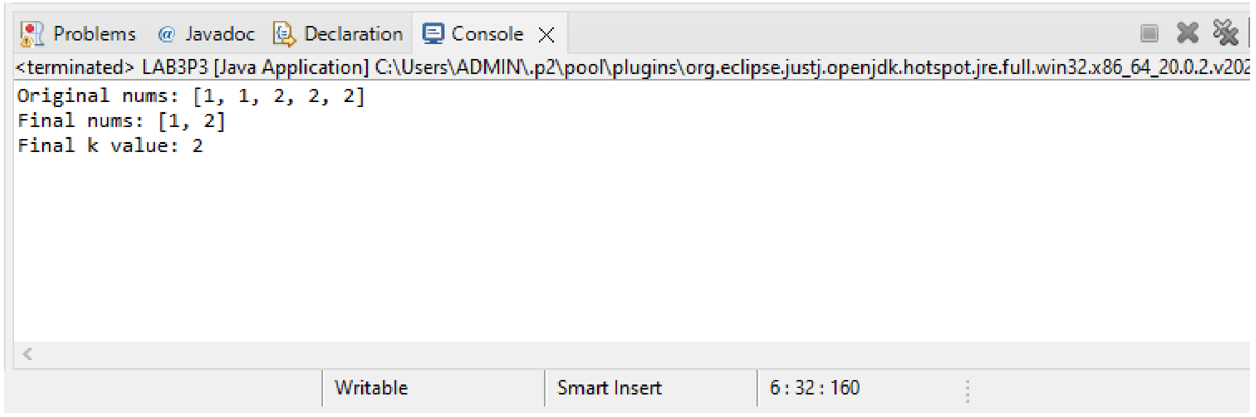
Source Code:

```java
package edu.northeastern.csye6200;
import java.util.Arrays;

public class LAB3P3 {
    public static void main(String[] args) {
        int[] nums = {1, 1, 2, 3, 3, 4, 5};
        int[] originalNums = Arrays.copyOf(nums, nums.length);
        int k = removeDuplicates(nums);
        System.out.println("Original nums: " +
java.util.Arrays.toString(originalNums));
        int[] finalNums = Arrays.copyOf(nums, k);
        System.out.println("Final nums: " + java.util.Arrays.toString(finalNums));
        System.out.println("Final k value: " + k);
    }

    public static int removeDuplicates(int[] nums) {
        if (nums.length <= 1) {
            return nums.length;
        }
        int i=0;
        for (int j=1; j<nums.length; j++) {
            if ( nums[j] != nums[i] ) {
                i++;
                nums[i] = nums[j];
            }
        }
        return i+1;
    }
}
```

Screenshots of Sample Runs:



```
<terminated> LAB3P3 [Java Application] C:\Users\ADMIN\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_20.0.2.v202
Original nums: [1, 1, 2, 3, 3, 4, 5]
Final nums: [1, 2, 3, 4, 5]
Final k value: 5
```



```
<terminated> LAB3P3 [Java Application] C:\Users\ADMIN\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_20.0.2.v202
Original nums: [1, 1, 2, 2, 2]
Final nums: [1, 2]
Final k value: 2
```