Computer Engineering
CSE 222/505 - Spring 2023
Homework # Report

Name: Ahmet Furkan Ekinci
Student Number: 200104004063

1. Problem Solution Approach

First of all, I read the pdf and understood it properly. Then I created classes for username and passwords in accordance with the document. Now i will tell you how i thought while creating the methods we were asked to create in this assignment.

(checkIfValidUsername) is a recursive method. I created some conditions as the logic of recursive methods at the beginning of the method. Then I called method again, decreasing size by 1 each time, until any of these conditions are met.

(containsUserNameSpirit) is a stack method. Since we were asked to use stack logic, I first converted password1 and username to stack. Then I checked whether any of the letters in the password are in username in the while loop until the stack where I hold the password is empty. While providing this control, I used the method of the stack class, search().

(isBalancedPassword) is another stack method. I created a satck to add the parenthesis which in the password1 string. Then I created a loop that goes from the first index of the password1 string to the last index. If it is in this loop, when I came across an open parenthesis, I had it added to the stack. If I come across a closed parenthesis, I first check if the stack is empty. If the stack is empty it means no parenthesis has been opened and if it is trying to close without opening the parenthesis then this is a problem so i return false. If the stack is not empty i have one more check. If the topmost element of the stack does not have an open parenthesis of the same type as the closed parenthesis, I return false again. While looking for the same type of open parenthesis, I also delete the last element added to the stack with the help of the pop method. And when the loop ends, I finally check if there are any elements left in the stack. The reason for this check is that any open parenthesis were added to the stack but could not be deleted because no closed parenthesis were found. When such a situation occurs, I return false again, in other cases the method returns true.

(isPalindromePossible) is a recursive method. This method could be written in a much more efficient way, but I was able to do it this way. I created an 'arr' array of length 26 holding the letters. I wrote a method that deletes the parenthesis of the password and converts the uppercase letters to lowercase letters, and by calling this method, I get a string containing only the letters of the password. I wrote a helper method that takes a string containing only letters and an array named arr as parameters. This helper method increases the value of that index of the arr array by one, according to the alphabetical order of the letter in the string. From this method, the arr array is returned as filled, and in our main method, it is checked whether there is an odd or even number of letters in the desired index. If there is an odd number, the method returns false because there cannot be a palindrome, but if it is even, the method will continue to call itself until the index is less than 0. If index is less than 0 the method will return true.

(isExactDivision) is a recursive method. The purpose of this method is to find out whether the sum of the numbers in an array can be equal to the password. I did not use any helper method while writing this recursive method. According to the logic of the algorithm I wrote, we start from the first index of denominations and move towards the last index and try all possibilities. If the password cannot be reset during these operations, then the method returns true. If the password cannot be reset, then the method will return false. With recursive logic, the method starts by first trying the probabilities in the last index of the denominations, and the probabilities come to the beginning, try by try.

2. Inputs and Outputs

```
System.out.print("1- ");
TestClass.helperMethod("furkanEkinci", "[rac()ecar]", 125, new int[] {4,7,9});

System.out.print("\n2- ");
TestClass.helperMethod("", "[rac()ecar]", 125, new int[] {4,7,9});

System.out.print("\n3- ");
TestClass.helperMethod("furkanEkinci", "pass[]", 125, new int[] {4,7,9});

System.out.print("\n4- ");
TestClass.helperMethod("furkanEkinci1", "pass[]", 125, new int[] {4,7,9});

System.out.print("\n5- ");
TestClass.helperMethod("furkanEkinci", "abcddbac", 125, new int[] {4,7,9});

System.out.print("\n6- ");
TestClass.helperMethod("furkanEkinci", "[[[[]]]]", 125, new int[] {4,7,9});

System.out.print("\n7- ");
TestClass.helperMethod("furkanEkinci", "([[[asd]]]]", 125, new int[] {4,7,9});

System.out.print("\n8- ");
TestClass.helperMethod("furkanEkinci", "race(c)ars", 125, new int[] {4,7,9});

System.out.print("\n9- ");
TestClass.helperMethod("furkanEkinci", "race(c)ar", 12125, new int[] {4,7,9});

System.out.print("\n9- ");
TestClass.helperMethod("furkanEkinci", "race(c)ar", 254, new int[] {44,71,98});
```

```
1- The username and passwords are valid. The door is opening, please wait...

2- The username is invalid. It should have at least 1 character. Try again...

3- The password1 is invalid. It should have at least 8 characters..

4- The username is invalid. It should have letters only. Try again...

5- The password1 is invalid. It should have at least 2 brackets.

6- The password1 is invalid. It should have at least 1 character from the username.

7- The password1 is invalid. It should be balanced.

8- The password1 is invalid. It should be possible to obtain a palindrome from the password1.

9- The password2 is invalid. It should be between 10 and 10,000.

9- The password2 is invalid. It is not compatible with the denominations.
```

3. Complexity Analysis

`checkIfValidUsername => O(n)`

The method works as recursive as the size of the string.

`containsUserNameSpirit => O(n)`

Made n rotations using 3 different loops. complexity is 3 * n.

`isBalancedPassword => O(n)`

In this method, completely stack logic is used and the result is reached by rotating a single loop n times.

`isPalindromePossible => O(n^2)`

In this method, two different methods with time complexity of O(n) are called, in addition, no loop is used. But since the method is recursive, the method will return itself n times. So time complexity is considered O(n^2).

`isExactDivision => O(2^n)`

Even if this method starts from the beginning in the for loop, it goes recursively to the end and starts the control with the number in the last index. When it comes to the last index, we check whether it can be created with the numbers at the end, and then we come to the first index by checking it as recursive logic.