Data Structures and Algorithms

HOMEWORK 4 REPORT

RUNNING COMMANDS AND RESULTS

test 1

```
String username = "sibelgulmez";
String password1 = "rac()ecar";
int password2 = 75;
int[] denominations = {4, 17, 29};
```

```
enesaysu@Enes-Aysu-MacBook-Air 1901042671_hw4 % java TestClass.java
The username and passwords are valid. The door is opening, please wait...
```

test 2

```
String username = "";
String password1 = "rac()ecar";
int password2 = 75;
int[] denominations = {4, 17, 29};
```

```
enesaysu@Enes-Aysu-MacBook-Air 1901042671_hw4 % java TestClass.java
The username is invalid. It should have at least 1 letter.
```

test 3

```java
String username = "sibel1";
String password1 = "rac()ecar";
int password2 = 75;
int[] denominations = {4, 17, 29};
```

```
enesaysu@Enes-Aysu-MacBook-Air 1901042671_hw4 % java TestClass.java
The username is invalid. It should have letters only.
```

test 4

```java
String username = "sibel";
String password1 = "pass[]";
int password2 = 75;
int[] denominations = {4, 17, 29};
```

```
enesaysu@Enes-Aysu-MacBook-Air 1901042671_hw4 % java TestClass.java
The password is invalid. It should have at least 8 characters
```

test 5

```java
String username = "sibel";
String password1 = "(racecar))";
int password2 = 75;
int[] denominations = {4, 17, 29};
```

```
enesaysu@Enes-Aysu-MacBook-Air 1901042671_hw4 % java TestClass.java
The password is invalid. It should be balanced.
```

test 6

```
String username = "sibel";
String password1 = "(no)(no)";
int password2 = 75;
int[] denominations = {4, 17, 29};
```

```
enesaysu@Enes-Aysu-MacBook-Air 1901042671_hw4 % java TestClass.java
The password is invalid. It should contain at least one letter from username
```

test 7

```
String username = "sibel";
String password1 = "(rac()ecars)";
int password2 = 75;
int[] denominations = {4, 17, 29};
```

```
enesaysu@Enes-Aysu-MacBook-Air 1901042671_hw4 % java TestClass.java
The password is invalid. It should be palindrome.
```

test 8

```
String username = "sibel";
String password1 = "(rac()ecar)";
int password2 = 5;
int[] denominations = {4, 17, 29};
```

```
enesaysu@Enes-Aysu-MacBook-Air 1901042671_hw4 % java TestClass.java
The password is invalid. It should be between 10 and 10000
```

test 9

```
String username = "sibel";
String password1 = "(rac()ecar)";
int password2 = 35;
int[] denominations = {4, 17, 29};
```

```
enesaysu@Enes-Aysu-MacBook-Air 1901042671_hw4 % java TestClass.java
The password is invalid. It should be obtained by summation of denominations along with arbitrary coefficients.
```

TIME COMPLEXITY ANALYSIS

checkIfValidUsername: The time complexity of this function is O(n), where n is the length of the input string 'username'. This is because the function recursively calls itself on a substring of 'username' with each recursive call reducing the length of the string by 1.

containsUserNameSpirit: The time complexity of this function is O(n), where n is the length of the input string 'username'. Inside the loop, the function pushes each character of 'username' onto a stack, and this operation takes constant time per character. Once the loop completes, the function pops each character from the stack and checks if it exists in the input password using the indexOf method, which takes O(m) time, where m is the length of the password.

isBalancedPassword: The time complexity of this function is O(n), where n is the length of the input string 'password1'. The pop and push operations on the stack take constant time. Therefore, the time complexity of the entire loop is O(n). The function returns a boolean value indicating whether the stack is empty or not. Checking the stack's emptiness takes constant time, so the overall time complexity of the function is O(n).

isPalindromePossible: The time complexity of this function is O(n^2), where n is the length of the input string 'password1'. First, the function uses the replaceAll method to remove any bracket characters from the input string 'password1', which takes O(n) time, where n is the length of the input string. Next, the function checks if the length of the resulting string is less than or equal to 1, which takes constant time. If this condition is true, the function immediately returns true. The overall time complexity of this function is O(n^2), where n is the length of the input string 'password1'.

isExactDivision: The time complexity of the isExactDivision function is O(n * m), where n is the length of the denominations array, and m is the value of the password2. If the password2

value is within the valid range, the function calls the sumHelper function with the denominations array, password2 value, and current index set to 0. This initial call takes constant time. The sumHelper function recursively tries all possible combinations of the denominations array to see if they can add up to the password2 value. This is done using a nested loop that iterates over all possible coefficients for each denomination, starting from 0 up to the maximum number of times that denomination can be used to obtain the password2 value. The recursion continues until either the password2 value is equal to 0, in which case the function returns true, or the current index reaches the end of the denominations array, in which case the function returns false. The worst-case time complexity of the sumHelper function is O(m), as it tries all possible combinations of coefficients for each denomination, starting from 0 up to the maximum number of times that denomination can be used to obtain the password2 value. Therefore, the overall time complexity of the isExactDivision function is O(n * m).