RMIT UNIVERSITY

RMIT University Vietnam

Assignment Cover Page

Subject Code:	COSC2658
Subject Name:	Data Structures & Algorithms
Location and Campus:	SGS Campus
Title of Assignment:	Group Project Report
Group Name:	Group 13
Student Name and Number:	Hong SeongJoon (S3726123) Lee Seungmin (S3864235) Nguyen Anh Duy (S3878141) Kang Junsik (S3916884)
Lecturer Name:	Tri Dang
Assignment Due Date:	May 9 by 23:59
Date of Submission:	May 5, 2023
Number of Pages:	13 pages

I declare that in submitting all work for this assessment I have read, understood and agreed to the content and expectations of the Assessment declaration.

1. Overview

This report describes a program that finds a *String correctKey* of 16 letters consisting of R, M, I, and T in the least number of attempts as small as possible.

The program will return -1 if an invalid argument of the *String guessKey* in the *public class SecretKeyGuesser* is provided. If the argument is valid, then the program will find out the *String correctKey* by running the algorithm. The program automatically stops whenever it receives 16 from a call to the *key.guess() method* which means all letters are correct, and then will display the correct secret key.

Since only one class exists, this program does not reference any software design patterns. Also, we created our own data structure and algorithm for the program, not using existing data structures and algorithms.

2. Data Structures & Algorithms

The following figures explain the basic steps of the algorithm. They focus on flow rather than the methods or variables of the code. All steps are assumed that *String guessKey* is 16 letters consisting of R, M, I, and T. And the next letter of R is M, the next letter of M is I, the next letter of I is T, and the next letter of T is R. As mentioned earlier, if the *String guessKey* is an invalid argument, -1 will be returned and the program is immediately stopped.

If the *String guessKey* is the exactly same as the *String correctKey* and gets a return value of '16' on the first attempt, the algorithm in the while loop will not run and print out the found secret key with a counter 1 as below.

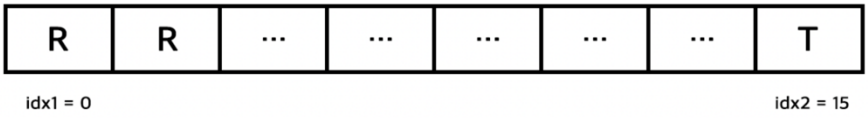
```
Number of guesses: 1
Guessing... RRRRRRRRRRRRRRR
I found the secret key. It is RRRRRRRRRRRRRRR
```

But if the return value is the unknown number 'n' which is smaller than 16, the algorithm inside the while loop will be run and put each letter in a *char[] curr* from the *String guessKey*. The *String guessKey* is 16 letters, but some parts are just omitted in the figure for a clear explanation.

There are two types of pivots in the algorithm: *int idx1* and *int idx2*. The *int idx1* moves

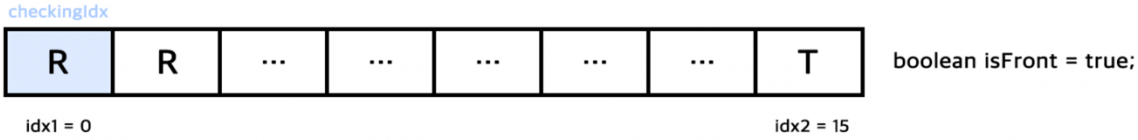
left to right from 0 which is the beginning of the *char[] curr* array, and *int idx2* moves right to left from 15 which is the end of the *char[] curr* array.

```
* guessKey = "RR.....T"          * int match = n
* correctKey = "MT.....T"        * int newMatch
```



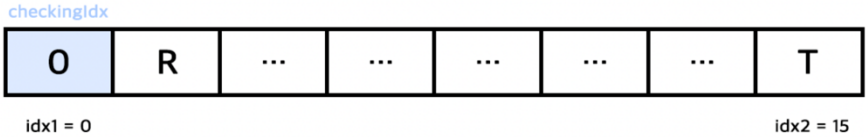
At first, *boolean isFront* is true which means the algorithm will check from the beginning index of the array. So, *int checkingIdx* becomes *int idx1*.

```
* guessKey = "RR.....T"          * int match = n
* correctKey = "MT.....T"        * int newMatch
```

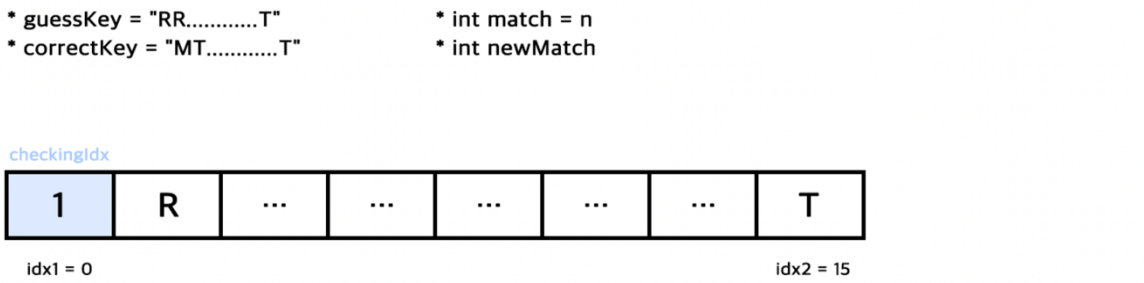


A *static void toNextChar(char[] curr, int idx)* is used to change the letter located in *int checkingIdx* to the next letter, change it to a number first. At this time, a *static int order(char c)* method that returns 0 for R, 1 for M, 2 for I and 3 for T is used. In this case, 0 corresponding to R is returned.

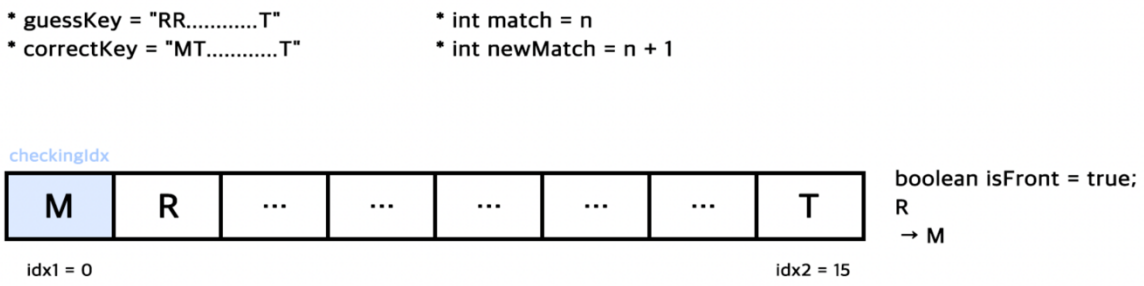
```
* guessKey = "RR.....T"          * int match = n
* correctKey = "MT.....T"        * int newMatch
```



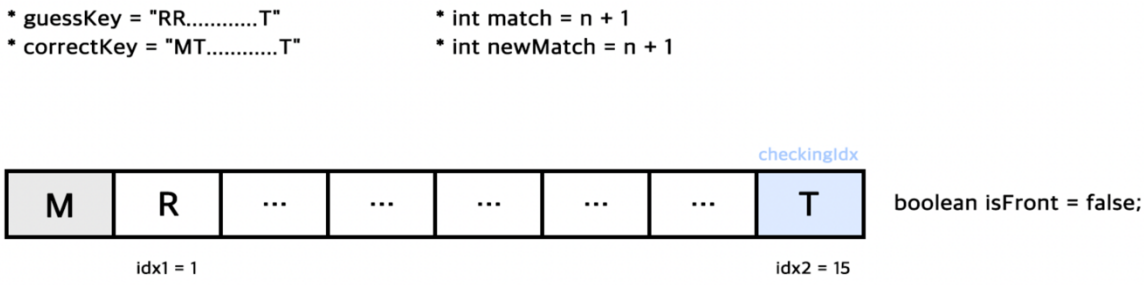
If the returned value is less than 3, then add 1 to the returned value to change to the next letter. In this case, 0 + 1 = 1.



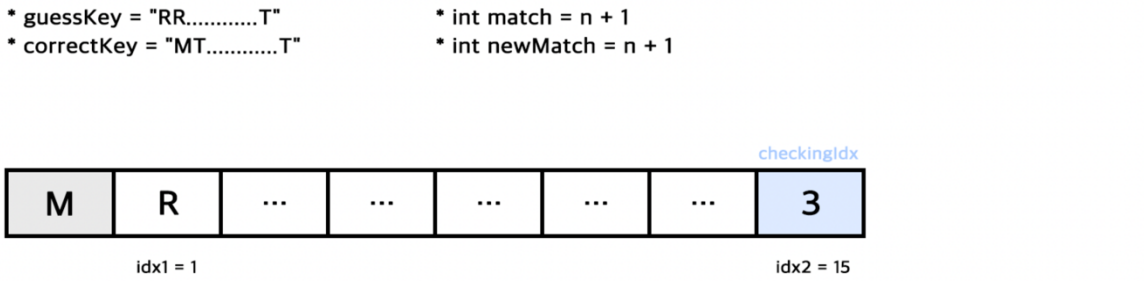
Returns the letter corresponding to 1 using a *static char charOf(int order)* method. In this case, M is returned. After changing the letter, execute the *key.guess()* method to see how many letters are correct. If *int newMatch* is increased than *int match* like the example, add 1 to *int idx1* pivot to move on to the next index.



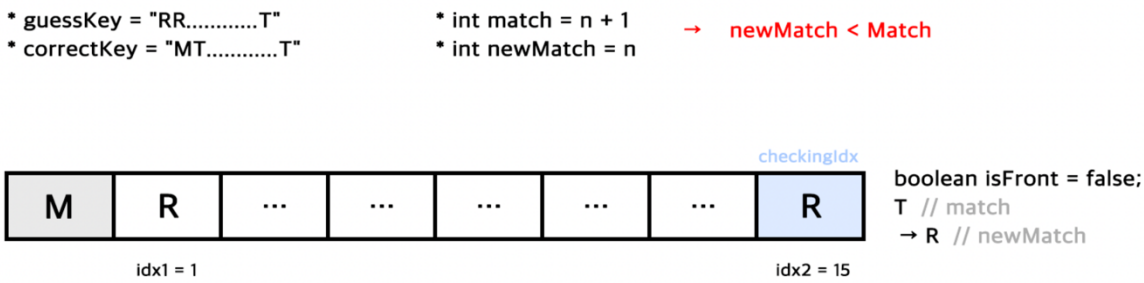
The *boolean isFront* changes from true to false and then *int checkingIdx* moves to *int idx2* pivot's position.



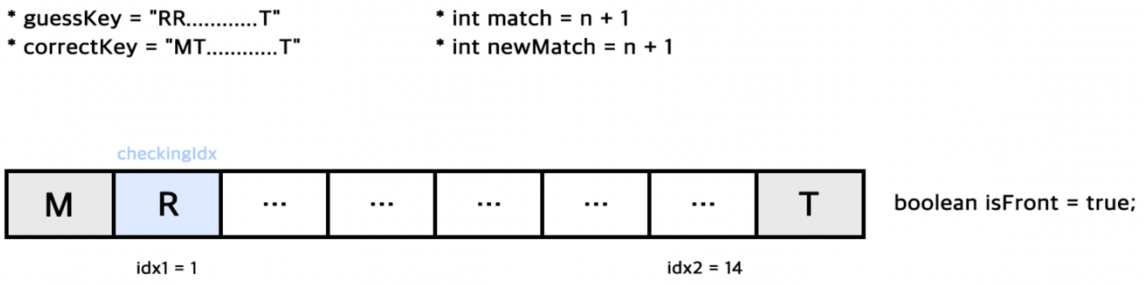
As before, to change the letter located in *int checkingIdx* to the next letter, the *static void toNextChar(char[] curr, int idx)* method is used first. Then inside the method, the *static int order(char c)* method is used to change it to a number. In this case, the *static int order(char c)* method returns 3 which is corresponding to the letter T.



If the returned number is less than 3, then 1 is added to the returned number to change to the next letter, but if 3 is returned, as in the example, R is unconditionally returned without an operation. After changing the letter, the *key.guess()* method will be executed to see how many letters are correct.



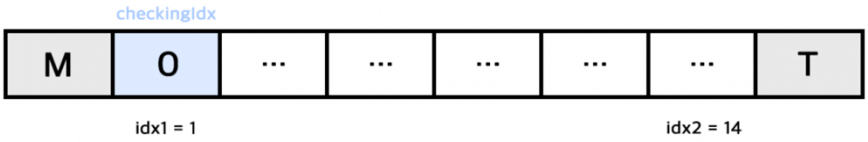
As shown in the example, if the *int newMatch* is less than the match which means the previous letter is the correct letter, it is returned to the previous letter. In this case, R is returned to T. And subtract 1 from *int idx2* pivot and move to the previous position. The *boolean isFront* changes from false to true and then *int checkingIdx* moves to *int idx1*'s position.



Repeat the same steps. Return the number corresponding to R, add 1 and change to the next letter.

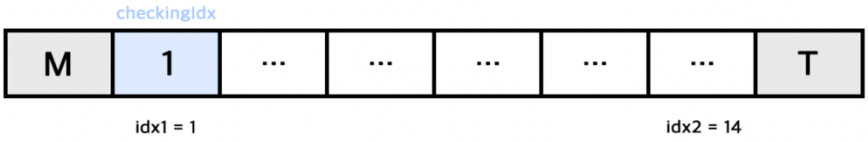
* guessKey = "RR.....T"
* correctKey = "MT.....T"

* int match = n + 1
* int newMatch = n + 1



* guessKey = "RR.....T"
* correctKey = "MT.....T"

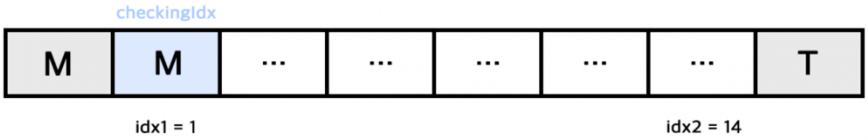
* int match = n + 1
* int newMatch = n + 1



Then, run the *key.guess()* method again to check if the *int newMatch* is greater than the *int match*. Since it is still the same as ‘n + 1’ in this case, repeat the steps of changing it to the next letter.

* guessKey = "RR.....T"
* correctKey = "MT.....T"

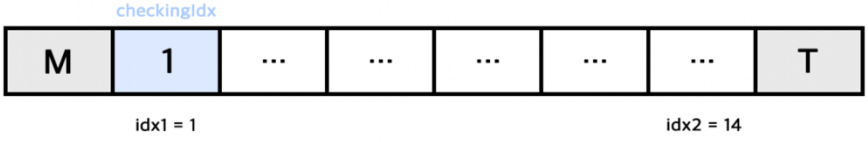
* int match = n + 1
* int newMatch = n + 1




```
boolean isFront = true;
R  // match
→ M  // newMatch
```

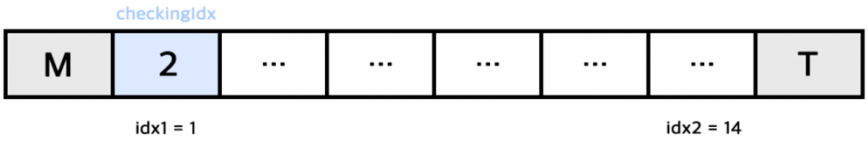
* guessKey = "RR.....T"
* correctKey = "MT.....T"

* int match = n + 1
* int newMatch = n + 1



* guessKey = "RR.....T"
* correctKey = "MT.....T"

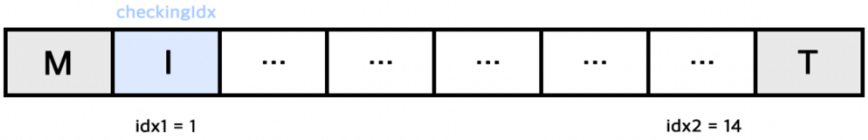
* int match = n + 1
* int newMatch = n + 1



Then, run the *key.guess()* method once again to check if the *int newMatch* is greater than the *int match*. Since the *int newMatch* is still the same as the *int match* in this case, repeat the steps of changing it to the next letter.

* guessKey = "RR.....T"
* correctKey = "MT.....T"

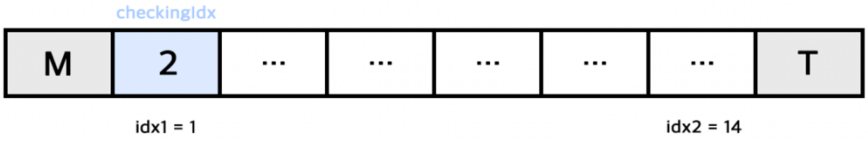
* int match = n + 1
* int newMatch = n + 1




```
boolean isFront = true;
R (x)
→ M (x)  // match
→ I  // newMatch
```

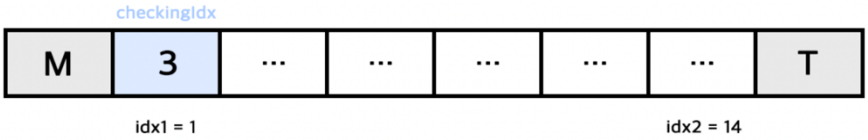
* guessKey = "RR.....T"
* correctKey = "MT.....T"

* int match = n + 1
* int newMatch = n + 1



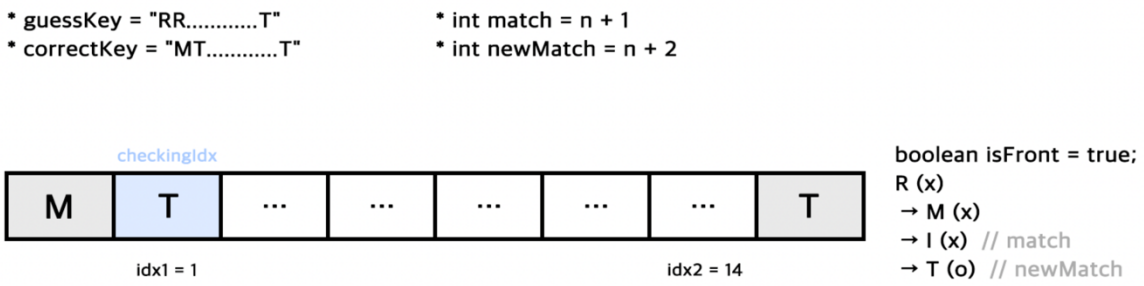
* guessKey = "RR.....T"
* correctKey = "MT.....T"

* int match = n + 1
* int newMatch = n + 1

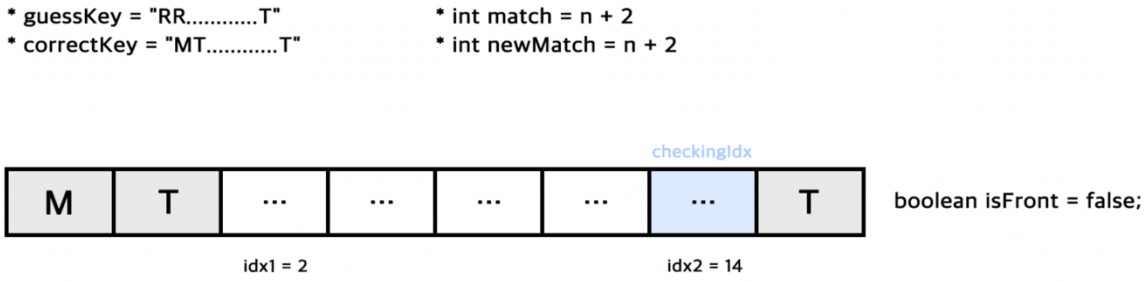


Since the algorithm has already changed from R to M, from M to I, and from I to T three times, T becomes the correct letter automatically. In this case, the algorithm does not run

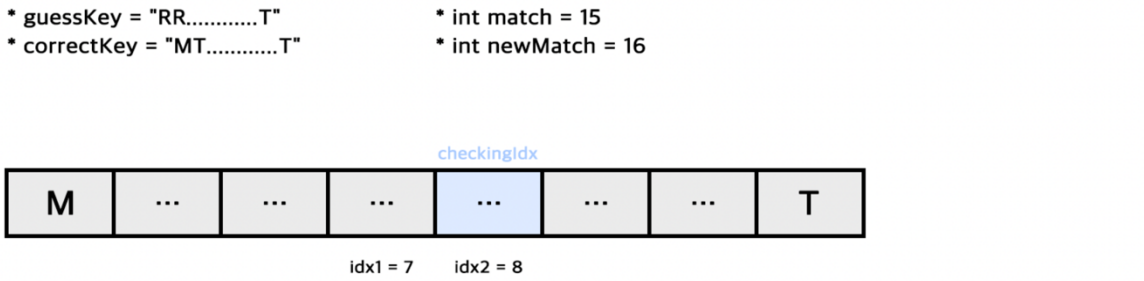
the *key.guess()* method, but add 1 to *int newMatch* to make it ‘n+2’.



Since the *int newMatch* is increased than the *int match*, add 1 to the *int idx1* and move on to the next letter. The *boolean isFront* changes from true to false and then the *int checkingIdx* moves to the *int idx2*’s position. Then, after changing the letter in the same process, the *key.guess()* method is executed to compare the values of *int newMatch* and *int match* until the *int match* becomes 16.



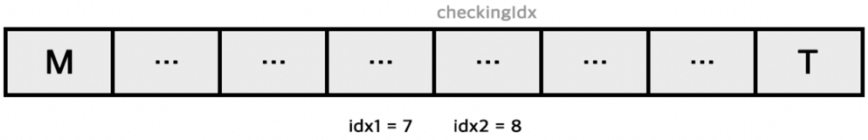
In the worst case, if all 16 letters of *String guessKey* were incorrect, the algorithm runs until *int idx2* becomes 8.



If *int match == 16*, which means all 16 letters match with the *String correctKey*, exit the while loop and print out the *String correctKey* with the counter of the total.

* guessKey = "RR.....T"
* correctKey = "MT.....T"

* int match = 16
* int newMatch = 16



3. Complexity Analysis

- Time Complexity

```
public void start() {
        SecretKey key = new SecretKey();   // { 1 }
        String guessKey = "RRRRRRRRRRRRRRT";   // { 1 }
        int idx1 = 0;   // { 1 }
        int idx2 = str.length() - 1;   // { 1 }
        boolean isFront = true;   // { 1 }
        int checkingIdx;   // { 1 }
        int match = key.guess(str);   // { 1 }

        if(match == -1){   // { 1 }
            System.out.println("Invalid! Please enter a valid key!!");   // { 1 }
        } else {   // { 1 }
            System.out.println("Guessing... " + str);   // { 1 }
            while (match != 16) {   // Since there are 16 letters to guess, the worst occurrences is { N + 1 }
                if (isFront) {   // { N }
                    checkingIdx = idx1;   // { N }
                } else {   // { N }
                    checkingIdx = idx2;   // { N }
                }

                char[] curr = str.toCharArray();   // { N }
                int newMatch;   // { N }
                toNextChar(curr, checkingIdx);   // { N }
                String s;   // { N }
                s = String.valueOf(curr);   // { N }
                System.out.println("Guessing... " + s);   // { N }
                newMatch = key.guess(s);   // { N }

                if (newMatch > match) {   // { N }
                    match = newMatch;   // { N }
                    str = s;   // { N }
                }
```

```
            else if (newMatch == match) {   // { N }
                toNextChar(curr, checkingIdx);   // { N }
                s = String.valueOf(curr);   // { N }
                System.out.println("Guessing... " + s);   // { N }
                newMatch = key.guess(s);   // { N }
                if (newMatch == match) {   // { N }
                    toNextChar(curr, checkingIdx);   // { N }
                    s = String.valueOf(curr);   // { N }
                    System.out.println("Guessing... " + s);   // { N }
                    newMatch++;   // { N }

                    if (newMatch == 16) {   // { N }
                        key.guess(s);   // { N }
                    }
                }
                str = s;   // { N }
                match = newMatch;   // { N }
            }
            if (isFront) {   // { N }
                idx1++;   // { N }
                isFront = false;   // { N }
            } else {   // { N }
                idx2--;   // { N }
                isFront = true;   // { N }
            }
        }
        System.out.println("I found the secret key. It is " + str);   // { 1 }
    }
}
```

Before the while loop, the occurrences of each line are 1. Since there are there are 16 letters to guess when while loop runs, the worst case of occurrences is N + 1. Inside the while loop, the occurrences of each line become N.

$$T(n) = 1 * 12 + 1 * (N + 1) + 34 * N = 35N + 13$$

So, the time complexity for this algorithm is O(n).

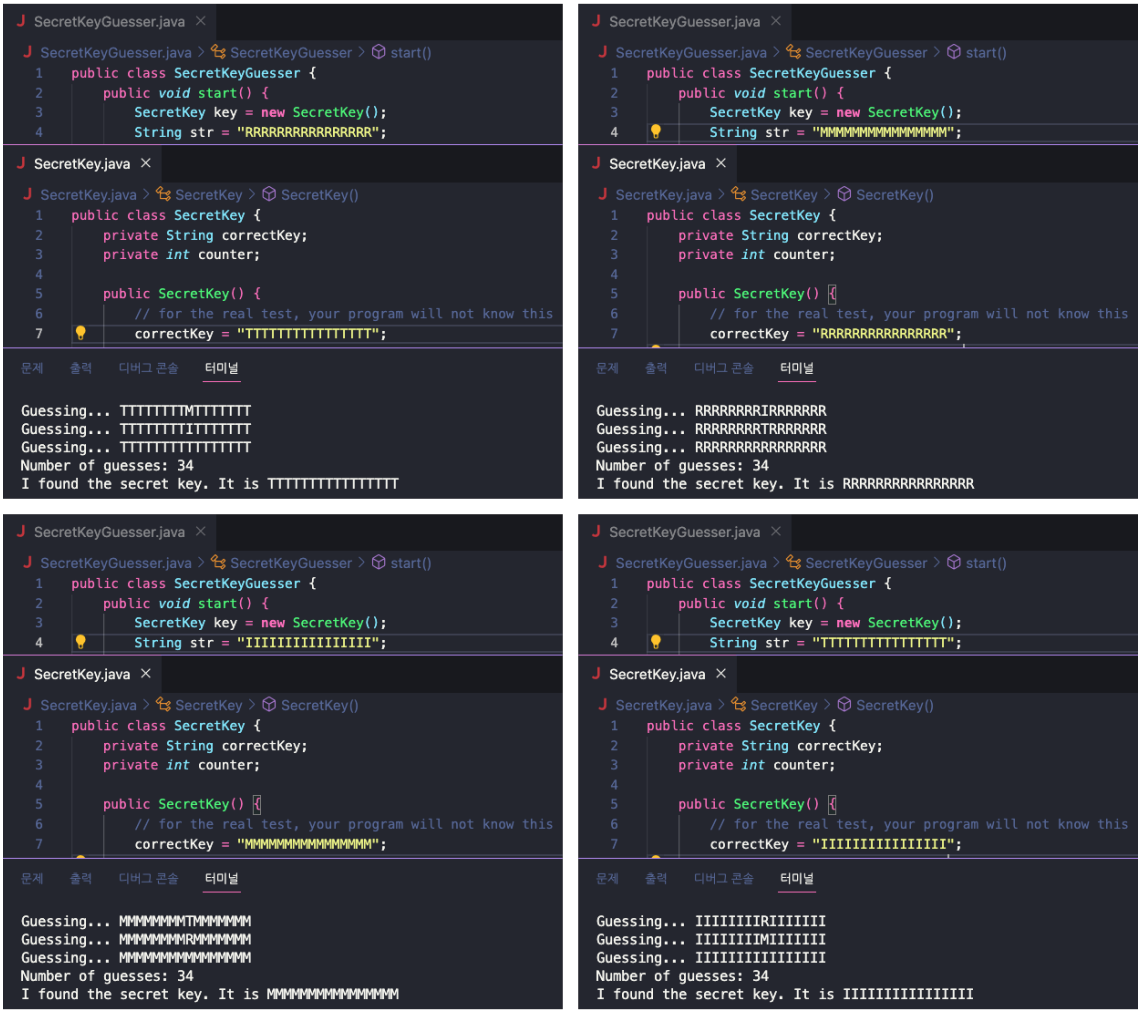
- **Space Complexity**

```
19                  checkingIdx = idx2;
20              }
21
22              char[] curr = str.toCharArray();
23              int newMatch;
24              toNextChar(curr, checkingIdx);
25              String s;
```

A one-dimensional array is declared inside the while loop, so N of space is needed. Memory grows linearly with the input value, so the space complexity for this algorithm becomes O(n).

4. Evaluation

- **Case 1**



Because the *String correctKey* and *String guessKey* consists of four alphabets, R, M, I, and T, in the worst, they can be replaced with the next letters up to three times for each index. Then, the *key.guess()* method runs two times for each index, the total counter will be the 1*(First check)* + 2 * 16 + 1*(Last check)*. If so, the counter will be 34 for the worst case.

- **Case 2**

SecretKey.java

```java
public SecretKey() {
    // for the real test, your program will not know this
    correctKey = "MIMIMMMMIIIIRMMM";
    counter = 0;
}
```

SecretKeyGuesser.java

```java
public void start() {
    SecretKey key = new SecretKey();
    String guessKey = "RRRRRRRRRRRRRRR";
```

문제 출력 디버그 콘솔 터미널

```
Guessing... MIMIMMMRRMIIRMMM
Guessing... MIMIMMMRRIIIRMMM
Guessing... MIMIMMMMRIIIRMMM
Guessing... MIMIMMMMMIIIRMMM
Guessing... MIMIMMMMIIIIRMMM
Number of guesses: 23
I found the secret key. It is MIMIMMMMIIIIRMMM
```

String correctKey = "RMIMIMMMMIIIIRMMM"

String guessKey = "RRRRRRRRRRRRRRR"

➡ *15 letters different*

As mentioned before, there is an order of letters. The next letter of R is M, the next letter of M is I, the next letter of I is T, and the next letter of T is R. So, in this case, even if a lot of letters are wrong, most of the counter results are in the mid-20s because the order of the letter has a greater impact.

- **Case 3**

SecretKey.java

```
public SecretKey() {
    // for the real test, your program will not know this
    correctKey = "RMRRMTMIRMIRRTRR";
    counter = 0;
}
```

SecretKeyGuesser.java

```
public void start() {
    SecretKey key = new SecretKey();
    String guessKey = "RRRRRRRRRRRRRRR";
```

문제 출력 디버그 콘솔 터미널

```
Guessing... RMRRMTRRRRIRRTRR
Guessing... RMRRMTMRRRIRRTRR
Guessing... RMRRMTMRRMIRRTRR
Guessing... RMRRMTMMRMIRRTRR
Guessing... RMRRMTMIRMIRRTRR
Number of guesses: 20
I found the secret key. It is RMRRMTMIRMIRRTRR
```

String correctKey = "RMRRMTMIRMIRRTRR"

String guessKey = "RRRRRRRRRRRRRRR"

➡ *8 letters different*

Compared to Case 2, there are fewer unmatching letters, but the counter does not show much difference due to an affection by the order of letters. Changing the letter of each index once, twice, or three times affects the counter a lot.

- **Case 4**

SecretKey.java

```
5       public SecretKey() {
6           // for the real test, your program will not know this
7           correctKey = "RTRRTTTIRTIRRTRR";
8           counter = 0;
9       }
```

SecretKeyGuesser.java

```
2       public void start() {
3           SecretKey key = new SecretKey();
4           String guessKey = "RRRRRRRRRRRRRRR";
```

```
문제  출력  디버그 콘솔  터미널
Guessing... RTRRTTTRRMIRRTRR
Guessing... RTRRTTTRRIIRRTRR
Guessing... RTRRTTTRRTIRRTRR
Guessing... RTRRTTTMRTIRRTRR
Guessing... RTRRTTTIRTIRRTRR
Number of guesses: 24
I found the secret key. It is RTRRTTTIRTIRRTRR
```

String correctKey = "RTRRTTTIRTIRRTRR"

String guessKey = "RRRRRRRRRRRRRRR"

➡ *8 letters different*

Compared to Case 3, the number of incorrect letters is the same, but we can see that the number of counters increases because they replaced much more letters in order.