Semester C, 2020

Course code: COSC2658

Course name: Data Structure & Algorithm

Lecturer: Dr. Yossi Nygate & Dr.Minh Dinh

Group Project: Guessing Game

Author: Group 10

Phu Vinh Luu - s3686970

Pham Nguyen Thanh Nhan - s3563953

Le Hong Thai - s3752577

SeungHee Lee - s3651505

Requirements from the rubric:
A small report was presented to describe:
1. The guessing strategy: clearly describe the steps in coming up with a guess
2. The special algorithm used (sorting, searching etc)
3. Any special data structure used in the strategy
4. Some examples run to demonstrate the efficiency of the guessing strategy.

Introduction:

In this guessing game, the computer will come up with a random number. The player must then continue to guess numbers until the player guesses the correct number. For every guess, the computer will either say:

(1) Strike -- a correct digit in the guess and in the right place.
(2) Hit -- a correct digit in the guess but out of order.
(3) Miss -- No digit of the guess is anywhere in the number.

From this rule of the game, we will have some algorithms to apply to our strategy to find the correct number fastest. We have the graph to show the time all system run

Guessing strategy:

Steps to come up with the guess:

1. Get all possible answers from 1000 to 9999 as required and put them into an array called PossibleAnswers.
2. Pick a random number to guess from the PossibleAnswers
3. Get the results of number of hits & strikes
4. Find and compare the result of hits & strikes of the guess, with all the elements in the PossibleAnswers array.
5. Keep all the elements with similar results of hits and strikes and eliminate elements with different results from the PossibleResult array.
6. Pick the next random number from the PossibleAnswers (After the PossibleAnswers got pruned by the previous guess) to guess.
7. Repeat step 2-5, until the guess is correct.

Example:

Target is 1111, to keep it simple.
An array list filled with possible answers from 1000-9999
Pick a random number from the array list (1000-9999), guess 2954 (result: Miss).
Keep all the elements with similar results of Miss, when compared to the guess 2954 and eliminate the rest elements that have different results. For example, 9542 4442 5429 etc. being eliminated with results of (4 hits, 1 strikes 1 hit and 4 hits consecutively) due to different results of hits & strikes compared to 2954.

Keep the elements with similar responses compared to the previous guess of 2954, i.e 1367, 8903, 8893, 8073 etc. They all returned Miss response when compared to the previous guess of 2954.

From the remaining elements in the array, pick one to guess (8073 Miss again). And keeps repeating the same step to eliminate all the possible answers with different results compared to the guess 8073 such as 3773, 8188, etc. (all the elements that are being eliminated either have one or more hits/strikes with both the previous guesses 8073 and 2954).

Keep the elements that have similar response Miss when compared to the previous guesses of 8073 (and 2954 for the first guess), such as 1116, 6611, 6116, 6661 etc.
The next guess is 6661 (1 strikes), keep all the elements with 1 strike compared to the 'internal target' of 6661 and randomly pick it to guess.

The next guess is 1111 (4 strikes). You finally win the game.

```
Guess   Response

2954
    Miss
8073
    Miss
6661
    1 strikes
1111
    4 strikes - Game over
Target: 1111 - Number of guesses: 4

Process finished with exit code 0
```

Figure 1: After 4 guesses, our program can win with the target of 1111.

Other example runs:

```
Guess   Response

1594
    1 strikes
6824
    2 strikes and 1 hits
6084
    1 strikes and 1 hits
6528
    1 strikes and 1 hits
7624
    3 strikes
2624
    4 strikes - Game over
Target: 2624 - Number of guesses: 6

Process finished with exit code 0
```

```
Guess   Response

3748
    2 hits
7326
    1 hits
2804
    1 strikes and 1 hits
8907
    4 hits
9870
    4 strikes - Game over
Target: 9870 - Number of guesses: 5

Process finished with exit code 0
```

```
Guess   Response

6300
    1 strikes
6254
    Miss
9810
    3 strikes
7810
    2 strikes and 1 hits
9870
    4 strikes - Game over
Target: 9870 - Number of guesses: 5

Process finished with exit code 0
```

Figure 2: Other example runs to demonstrate the efficiency of our program.

The efficiency of the strategy average case is about 6-7 guesses to get the result. Worst case witness so far is 10 steps. Best case witness so far is 4.

Explanation of algorithm:

The main key goal of the algorithm is to eliminate numbers that have different responses (hits & strikes) of those numbers when comparing with guesses and keep the numbers with similar responses (hits & strikes) compared to the guess.

By doing this way, after 4-7 guesses, the possible answers are being pruned or cut off down to 2-3 possible answers.

Our guessing strategy is inspired by Donald Knuth, who demonstrated the algorithm in 1977 to solve the Mastermind game with a 5 step algorithm (0-6 digits instead of 0-9).

Special data structure being used:

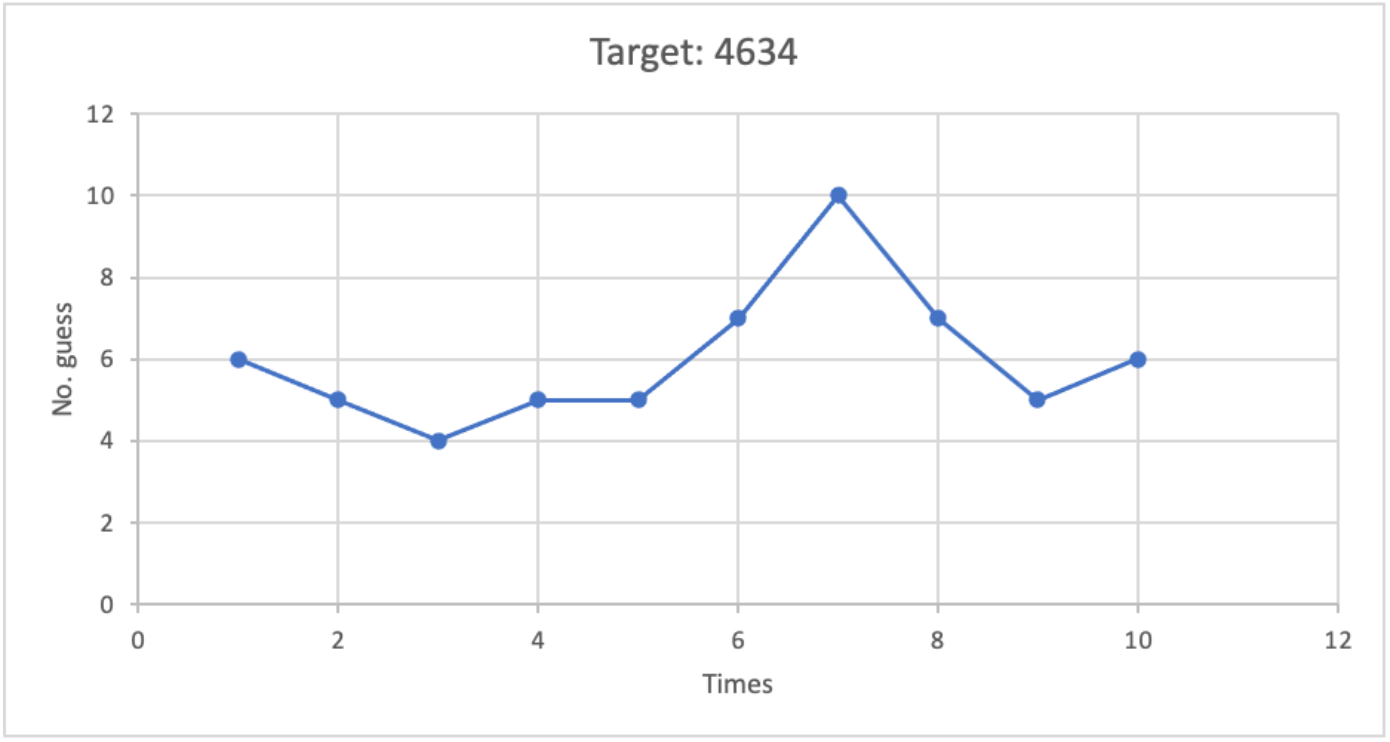
The main data for our guessing strategy relies on the pool of Possible Answers after each guess. Instead of using an Array List or 1D array, which is prohibited by the assignment requirements, we find that HashSet in Java is a more appropriate data structure for the Possible Answers variable.

We opt to use HashSet instead of ArrayList with the following reasons:

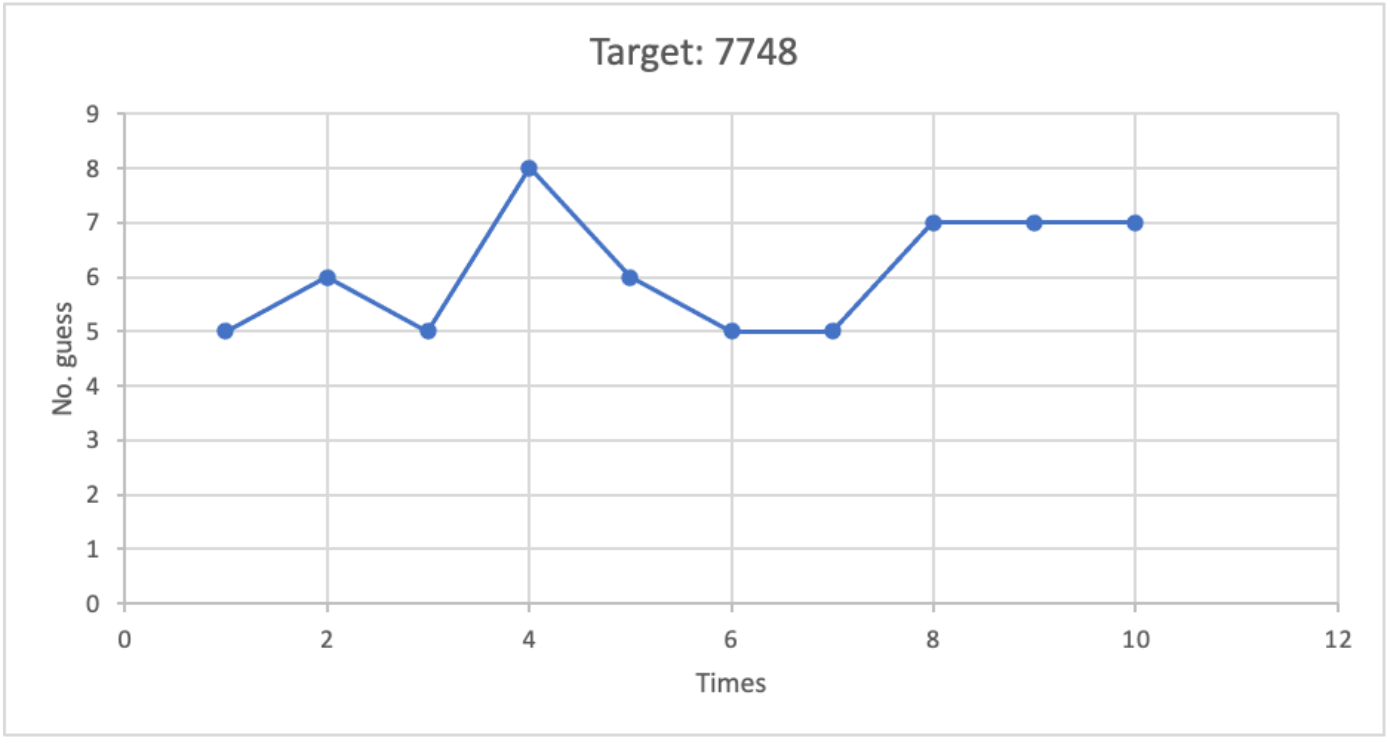
1. HashSet stores unique elements only

With the possible answers range from 1000-9999 we find it best to use HashSet so that we can make sure that there are no duplicate possible answers.

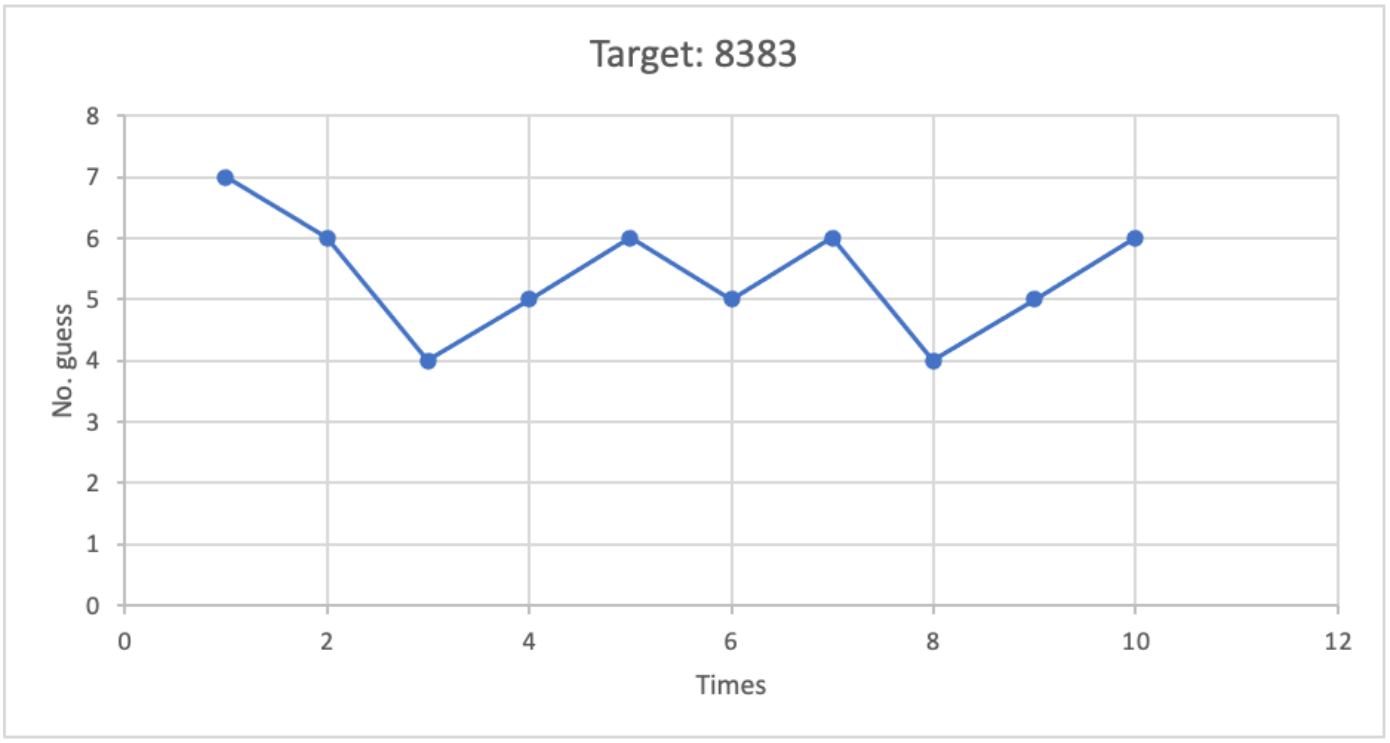
Testing case:



Average number of guess is 6



Average number of guess is 6.1



Average number of guess is 5.4

From some test cases, the average guess of my team is from 6 to 6.5.

References

Mastermind game:
https://en.wikipedia.org/wiki/Mastermind_(board_game)

Java HashSet:
https://www.javatpoint.com/java-hashset

Knuth algo:
https://www.cs.uni.edu/~wallingf/teaching/cs3530/resources/knuth-mastermind.pdf?fbclid=IwAR1KllnfKfpjTswS6iJNS_XD1Od4eAEyNsbxi0QgxmqeRDrAZV_VlToKKvw