Worst-time Complexity in Big O notation

-Genetic Algorithm:

We are supposing that n is the population size, and we-ll take in account only the methods which are not constant and iterate over sets.

The main method of the Genetic algorithm is the solve() method , to analyze the worst time complexity of solve() we'll first analyze the worst time cases of the methods called in solve() because it depends on the time complexity of each step.

For now, as in our test cases for the genetic algorithm , let's assume that the population size is 100, codeLength is 4 and maximum number of steps is 10.

-generateInitialPopulation() method is called only once at the start of solve(). It iterates over population_size elements and does constant operations(population.add).It generates populationSize random codes, each of codeLength length. The time complexity is O(populationSize * codeLength). In our case that would be O(100 * 4). So the cost would be O(n) in the worst case.

-getBestGuess() method is called in each iteration, which has a time complexity of O(n**maxSteps*) because it iterates through the population(size n) to find the guess with the lowest fitness, and it's repeated *maxSteps* times. *maxSteps* is constant so we'll assume that its cost is O(n). In our case, It calculates the fitness for 100 individuals and finds the one with the lowest fitness value. So the cost would be O(n) in the worst case.

-performSelection() method sorts all the population based on fitness (iterates all over n) , and then selects half of the population with the lowest fitness(iterates again over n/2). So the time complexity would be O(populationSize * log(populationSize)). In the worst case that would be O(n*log(n)) which is because of sorting the population.

-performCrossover() method selects two random parents and performs a crossover with a fixed crossover rate. It iterates over the whole population and the time complexity for iterating is O(populationSize * codeLength). So, the time complexity is linear with respect to the population size. So in the worst case the time complexity is O(n).

-performMutation() method involves randomly selecting a gene by iterating through the entire population and changing its value to generate a new solution, which has a time complexity of O(n). In the worst case scenario all codes are mutated. So in the worst case it takes O(n) , a linear time with respect to population size and the code length.

As we know, the solve() method runs for a certain number of generations. Let's say generation **M.** And in each generation it does selection,crossover and mutation. So overall we can say that the worst time complexity for our algorithm depends on solve() method itself and is **O(M*(N+N * (Log (N)))).** Where M is the number of generations, and N is the size of population.

Also we come to conclude that the most time consuming function in our algorithm is to perform selection because it involves sorting the population of size N and takes O(N*log(N)) time.

-Five Guess:

For the five guess algorithm, we will be assuming that the codeLength is 4, number of pegs (**P**) is 6, and maximum number of attempts (**M**) is 10, as in a classical Mastermind game. The codeLength is the length of the secret code that the algorithm has to solve, number of pegs is the number of possible pegs/colors that can be used in the game and maximum number of attempts is for the algorithm to find a solution.

The main method of the five guess algorithm is the solve() method , to analyze the worst time complexity of solve() we'll first analyze the worst time cases of the methods called in solve() because it depends on the time complexity of each step.

First we have a method generateAllPossibleCodes() which is called at the start to generate all possible combinations of length codeLength and **P** pegs.We call this methods recursively for codeLength times and in each recursive call, it iterates over the numberOfPegs (**P**). So the time complexity is **O(P ^ n)** where n is the number of code Length.

GenerateAllPossibleCodesHelper() is a helper method we use in generateAllPossibleCodes. The time complexity of this method is already pointed out above.

We have a reinitialize() method which is called after we find a solution, and it sets the remaining codes and the current guesses (which is an array) back to the initial value. This method creates an array list of all possible codes so the time complexity will be O(p^n).

Now comes the main method of the algorithm, the solve() method. As we know we have to solve the method in the maxSteps value. Solve() method tries to solve the game in maxSteps steps. In each step, it generates a guess, calculates the feedback for that guess, removes the codes from the remainingCodes which won't give the same feedback, and finds the best next guess using the minmax method. So, the time complexity of this method is **O(maxSteps * (P^n + n * P^n + n^2 * p^n))** because it involves iterating through all remaining codes in each step,copying remaining codes to another list and also finding the best guess using the minmax method. So, in our case the cost in the worst case scenario is **O(10 * (P^4 + 4 * P^4 + 4^2 * P^4)).**

As mentioned above the minmax method tries to find the best guess from the remainingCodes. It calculates the scores for all possible guesses and chooses the guess that minimizes the maximum number of possible remaining codes in the worst case so the time complexity of this method is **O(n^2 * P^n)** as it involves two nested loops over remainingCodes.

Overall, the worst case time complexity of the five guess algorithm can be said to be **O(maxSteps * P ^ n * n ^2)**, where maxSteps is the maximum number of steps allowed, P is the number of pegs(6) and n is the codeLength(4).

Principal functionality of the game such as maintaining of records, rankings, difficulty levels, providing help, and the management of loading and saving of game is as follows:

1)Ranking & Records

Let's start with the Ranking class which handles the score functionality between two players and also maintains a record of all the players.

First we have a function called **UpdateRecords** that updates the records of ranking with the player's name and score. Well for now we assume we will just be storing the records for 10 players so in our case the time complexity is not that big but as it uses a sort operation, in the worst case scenario the time complexity of this function would be **O(n log n).**

Now we have a method to **update the scores of two players**,which calls the method addScore which creates a new ScoreEntry and adds it to the list of scores then sorts the list. So, the time complexity will be **O(n log n).**

We have several other functions which call the **addScore** function to be complete so they also have a time complexity of **O (n log n).**

There are several methods that calculate the score, take the top scores from the list of scoreEntry, etc which has a constant time.

Overall, the most time consuming operations in the **Ranking class** are those which require sorting the scores. So, we can say that the time complexity will be **O(n log n).**

2)Assistance, Difficulty ,Feedback class

This is the class that handles the assistance and feedback mechanism in the game, i.e you can ask for assistance and it will provide you assistance in several ways and the feedback after each guess. Below is the analysis of the time complexity of this class :

getRightPegsInRightPosition(boolean forPlayer1) : This method provides feedback about the number of correctly positioned pehs in the last guess made. It accesses the element which is in the MastermindBoard class through a getter so the time complexity is **O(1)**.

Also, there are two other functions called getRightPegsInWrongPosition and getRandomCorrectPegAndPosition which also access the element of the mastermind board

class and have the time complexity of **O (1).**

Now to get the time complexity of how the feedback is provided we will analyze the MastermindBoard class:

As we know **Mastermind Board** class is literally the board of the game that is responsible for handling mainly guesses and feedback. First of all we have a **isCorrectGuess** to analyze which checks if the given guess matches the secret code for either of the two players. This method has a time complexity of **O (n)**, where n is the length of the array i.e the codeLength.

Another we have a **getFeedback** method that provides the feedback which has a time complexity of O (n). This method calls another method to get the feedback for a guess which has a time complexity of O (n) and then adds the guess and the feedback to a list which has a time complexity of O (1). So, overall we can say that the **time complexity is O (n)**.

Now, how the **difficulty level** is handled in a game is another topic. Our class that represents the difficulty level is very simple. Difficulty level has 3 fields: name of the difficulty level, number of maximum attempts which depend on the difficulty level and according to the difficulty level we decide if assistance is provided or not. So, as our difficulty level class just has getters and setters the time complexity remains constant. And while configuring the game we set the difficulty level to a string which can be easy, medium and hard. And then we do other things like setting maxAttempts to 8, 10 or 12 which have a constant time Also, we set if the assistance is provided or not which has a constant time too.

3) Persistence layer

Now for last, let's talk about how the game is being saved and loaded. In the domain layer we have a game save class that we serialize. The gameSave class contains all the necessary attributes and values that are needed to load the state of the game later. Then we pass this Serialize class as a parameter to the load and save functions in the persistence layer.

a) **save** : This method writes the gameSave object to a file, so the time complexity will be the time taken to do the saving operation which is writing to the disk. And writing to the disk depends on the size of the object. For example, let's consider our object has n elements. Then we need to save the n elements which takes linear time, i.e. **O (n**).

b) **load** : This method reads the object from a file, and the time complexity depends upon the size of the file being read. And after loading we need to set the game board which has a constant time. So, overall the time complexity can be said to be **O (n)**,

c) **SaveRecordsToFile and LoadRecordsFromFile** : This method is used to create a leaderboard of the game. The scores are taken after each game and compared to the scores that are in a file which is saved by this operation. It also has linear time complexity as it depends upon the size of file being saved and loaded. So, overall the time complexity can be said to be **O (n)**.