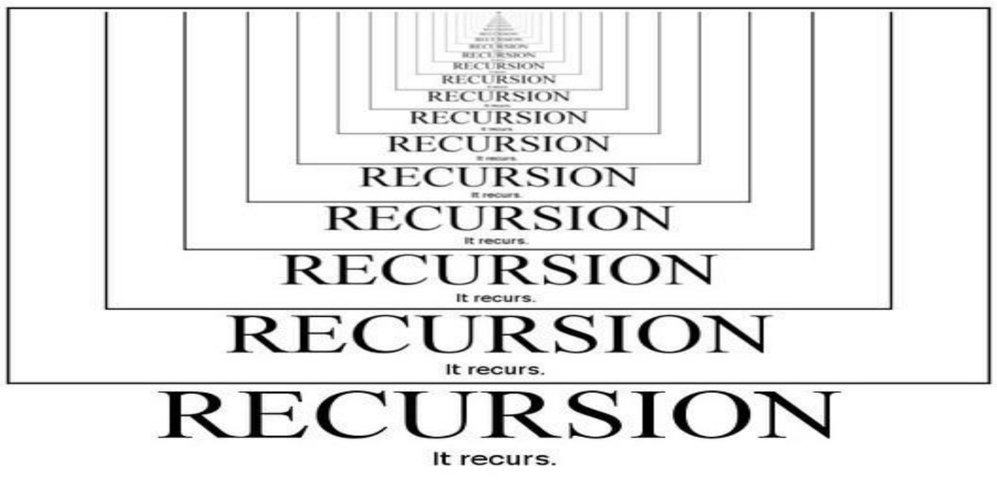
Data Structures And Algorithms

HOMEWORK - 1

Name	ID
İSAK ÇOBAN	1242***0116
OĞUZHAN ERÇELİK	1083***0016
SEVDA ERGÜN	2860***0092



REPORT

	Good	Bad
Computing Fibonacci Sequence		✓
Towers of Hanoi (Search for it if you don't know the problem)	✓	
Computing factorial of a given number		✓
An autonomous robot looking for a way out of an unmapped maze	✓	

THE PURPOSE OF THIS PROJECT:

The purpose of the homework is to create a solvable sudoku and solve it. The user can press ***C*** to create a sudoku and ***S*** to solve it. It doesn't matter if the key words are lower or upper case.

HOW DID YOU SOLVE IT?

There are two ways to create a sudoku. First one is creating a completed sudoku and then erasing some of cells randomly until there are enough hints to solve it. Other one is creating a sudoku with enough clues. We choose the second option to solve this problem. There were several problems about this algorithm. Understanding ***enough clues with unique solution*** is the most important part. To create a unique sudoku we needed at least 17 clues. Gary McGuire and his friends proved that number many years ago and we still accept it as a fact. So we decided to set the number of clues as 18 to know that our sudoku is solvable and unique.

We have 4 classes in this homework. ***SudokuCreater*** class can generate a puzzle with enough hints. We have a 9x9 array and it is full with zeros. To fill this array with enough hints according to the sudoku rules we need some method. ***Controller*** class is an abstract class because I need it just to use its methods not to create an object of ***Controller***. The methods in ***Controller*** class check the array constantly and tells the method named as ***generate()*** in ***SudokuCreater*** class it is safe to put the hint or not. If it is safe, program continues. After the array has 18 hints, we have to know it is solvable to print it. To check this, ***isSolvable()*** method helps the ***SudokuCreater*** class. ***SudokuSolver*** class exists to solve a sudoku. ***solveSudoku()*** method in ***SudokuSolver*** does this job. Solving a sudoku means using ***Controller*** class one more time so while it is trying to solve it, the method uses ***Controller*** class's method.

Solving occurs as recursively. After the creating operation, ***solveSudoku()*** method tries all possible keys according to the sudoku rules. If it can't put any keys to a cell this means the method couldn't solve the sudoku so method comes back and try other key combinations until it can solve the sudoku or decide there is no solution. If the ***isSolvable()*** control says ***generate()*** method there is a solution for the created sudoku, now it is safe to print it to screen. If there is no solution ***generate()*** method resets the array and creates a new sudoku. This loop will continue until there is only one solvable sudoku on the screen.

Sudoku class provides the user an interface. This class exists just for the user so he/she can use other classes. When the user presses ***C***, the sudoku with hints will appear on the screen if it is solvable. After this, if the user press ***S***, the solution of the sudoku will appear on the screen. There is another option, ***Stop*** . If the user writes the screen ***Stop*** the program will terminate.

NOTE 1:

While we are creating the sudoku we checked if it is solvable or not and while doing this we get the fully solved sudoku. When the user asked for the solution, we printed to the screen that solved puzzle because trying to solve the created puzzle takes time and doing this twice will make our program slower.

NOTE 2:

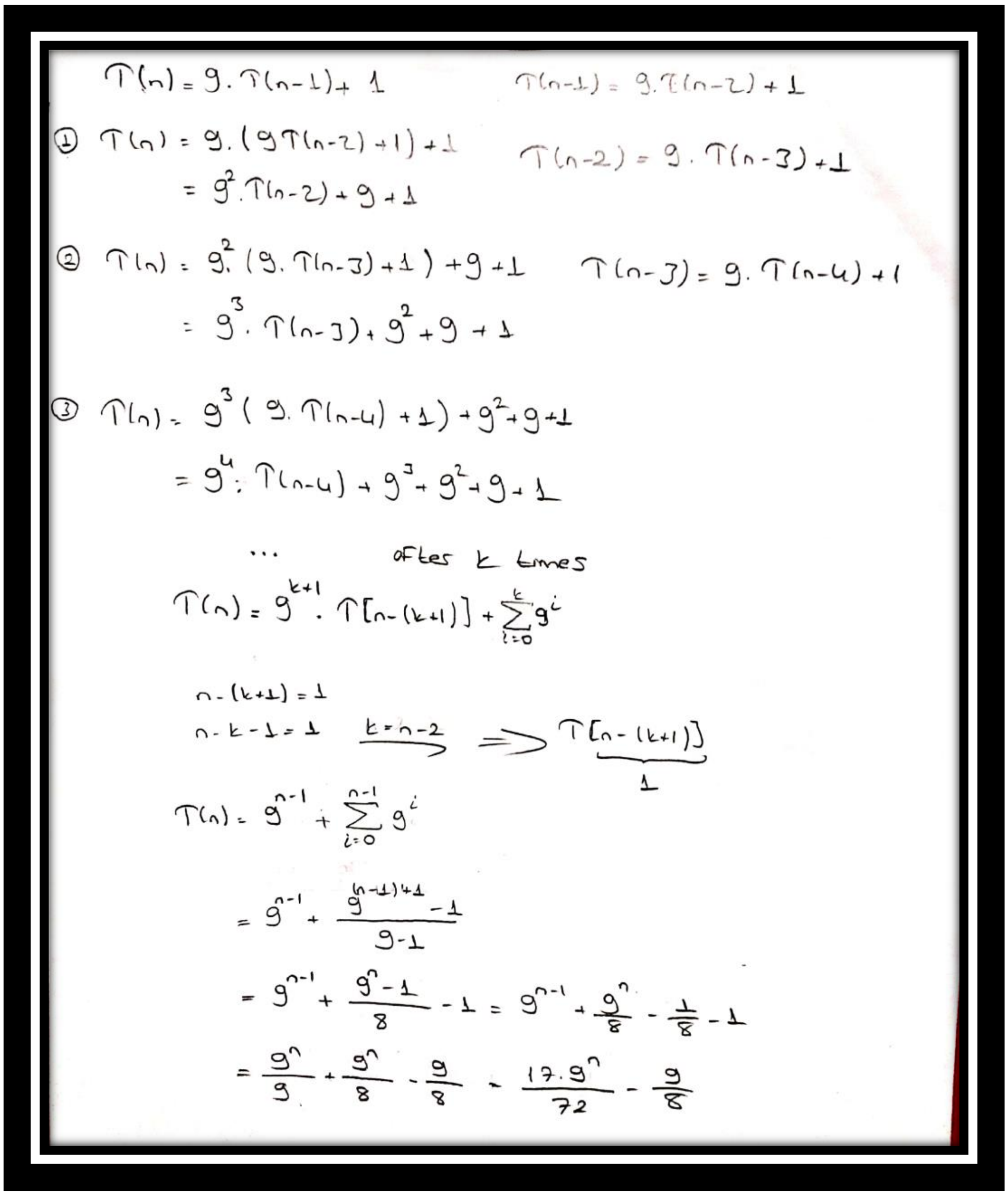
Creating a sudoku as randomly sometimes takes a lot of time. The program comes back if it can't find a solvable sudoku so when this happens repeatedly, the user can wait for a long time.

We use ***Intellij*** as IDE to code all classes. Moreover, we use our private repositories in ***Git*** to share our codes with our teammates.

TIME COMPLEXITY:

We did some researches and examined our source codes. We conclude that running time of ***solveSudoku()*** method is $\boldsymbol{T(n) = 9 \times T(n-1) + O(1)}$**.** Our function is recursive so to understand its ***big-O*** we used backward substitution.

Lets assume that we are trying the solve a sudoku and we have only one cell to fill. In the worst case scenerio we have **n** possible tries. Lets go backward. We have two empty cells now. For the first cell we have n possible tries again and after we filled it for the second cell we have n possible tries one more time. This gives us $\boldsymbol{n^2}$tries. This goes until it reaches all cells are filled. Finally we get **O**$(\boldsymbol{n^m})$ which n will be 9. So in the end **O**$(\boldsymbol{9^m})$ which m is the number of spaces that are blank. With this point of view and the solution below time complexity will be **O**$(\boldsymbol{9^m})$**.**



$T(n) = 9 . T(n-1) + 1$ $T(n-1) = 9 . T(n-2) + 1$

① $T(n) = 9 . (9T(n-2) + 1) + 1$ $T(n-2) = 9 . T(n-3) + 1$

$= 9^2 . T(n-2) + 9 + 1$

② $T(n) = 9^2 (9 . T(n-3) + 1) + 9 + 1$ $T(n-3) = 9 . T(n-4) + 1$

$= 9^3 . T(n-3) + 9^2 + 9 + 1$

③ $T(n) = 9^3 (9 . T(n-4) + 1) + 9^2 + 9 + 1$

$= 9^4 . T(n-4) + 9^3 + 9^2 + 9 + 1$

... after k times

$$T(n) = 9^{k+1} . T[n-(k+1)] + \sum_{i=0}^{k} 9^i$$

$n-(k+1) = 1$

$n-k-1 = 1$ $\underset{\longrightarrow}{k=n-2}$ $\Rightarrow \underbrace{T[n-(k+1)]}_{1}$

$$T(n) = 9^{n-1} + \sum_{i=0}^{n-1} 9^i$$

$$= 9^{n-1} + \frac{9^{(n-1)+1} - 1}{9-1}$$

$$= 9^{n-1} + \frac{9^n - 1}{8} - 1 = 9^{n-1} + \frac{9^n}{8} - \frac{1}{8} - 1$$

$$= \frac{9^n}{9} + \frac{9^n}{8} - \frac{9}{8} = \frac{17 . 9^n}{72} - \frac{9}{8}$$

Contributions as a percent of each member of a group:

İsak	Oğuzhan	Sevda
33%	***33%***	***34%***