***DATA STRUCTURE HOMEWORK 1***

1242\*\*\*0116 İSAK ÇOBAN

2860\*\*\*0092 SEVDA ERGÜN

1083\*\*\*0016 OĞUZHAN ERÇELİK

***REPORT***



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.

There are two ways to create a sudoku. First one is creating a completed sudoku and then erase 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. “solve” 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 trys 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 alppear on the screen. There is another option, “Stop” . If the user writes the screen “Stop” the program will terminate.

NOTE:

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.

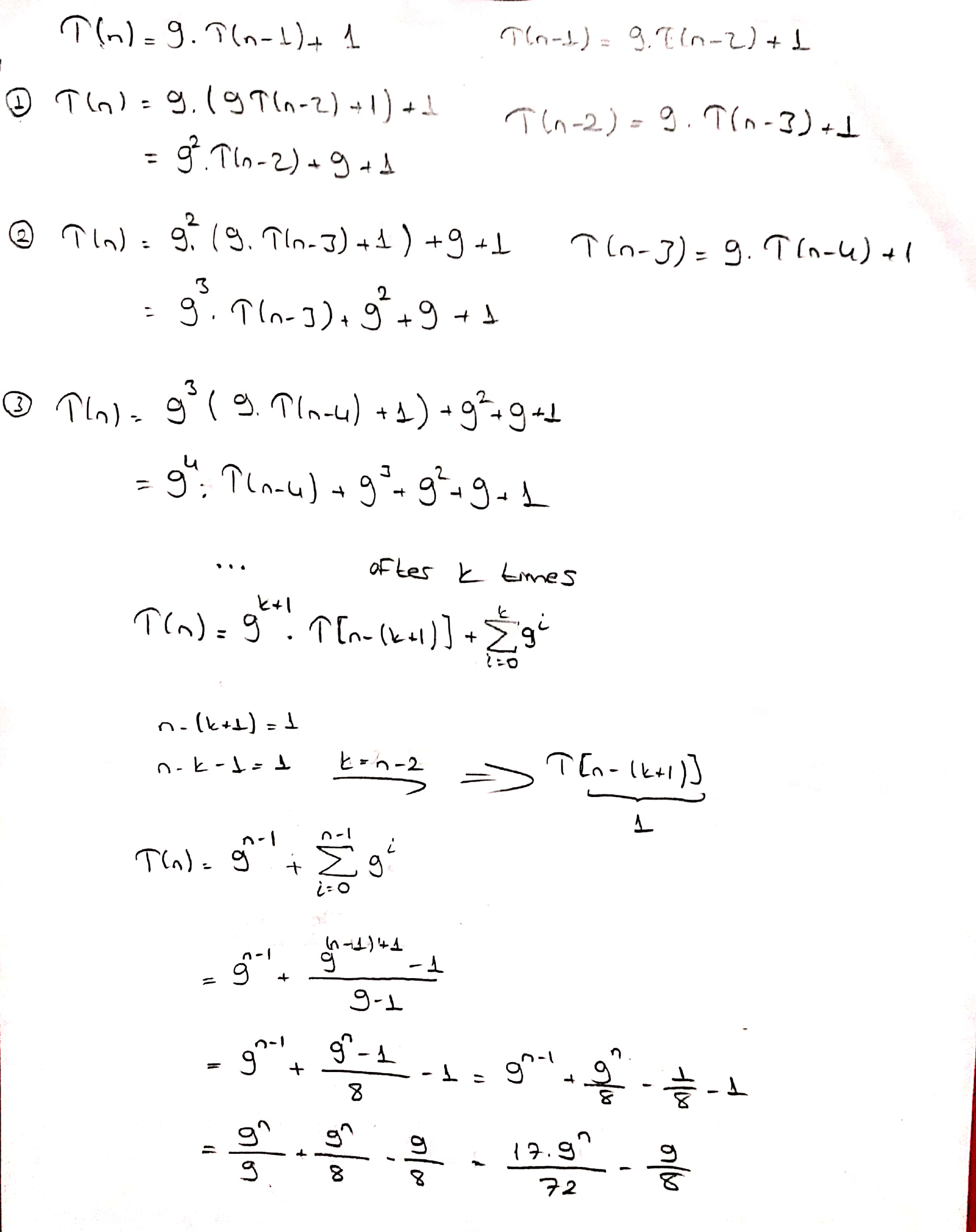
NOTE:

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 wait 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 resarches and examine our source codes. We conclude that running time of solveSudoku method is . 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 cell 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 tries. This goes until it reaches all cells are filled. Finally we get O() which n will be 9. So in the end O() which m is the number of spaces that are blank. With this point of view and the solution above time complexity will be O().

|  |  |  |
| --- | --- | --- |
| İsak | Oğuzhan | Sevda |
| ***33%*** | ***33%*** | ***34%*** |