Name: Han Chien Leow
Student Id: s3778722

Concept:

The Killer Advanced Solver implemented uses smart representation and heuristics of the exact cover problem to solve the killer sudoku.

In constructing the binary matrix, the solver first computes for each cage's all possible sets of symbols that add to the cage's total value, each set of length equal to the number of cells occupied by the cage. Unlike the Algorithm X Solver class, each row of the binary matrix in the Killer Advanced Sudoku solver is a set of cell placements that satisfies a cage. Each cage will have a row for every order of every combination of symbols that adds to the cage's total. Like the Algorithm X Solver class, the columns of the binary matrix are the constraints of standard sudoku, but for the Killer Advanced Sudoku solver comes with the addition of a set of columns that represent a cage being satisfied. When constructing a row, for each placement the row represents, the standard sudoku constraint columns satisfied by the placement, are given a 1 in the row. Similarly, the column of the Killer Advanced Sudoku solver that represents the cage that the row satisfies is given a 1 in the row. Algorithm X is closely referred to for the Killer Advanced Sudoku to be implemented to find an exact cover solution, which is a grid that satisfies all constraints of the sudoku and satisfies every cage.

Definitions / Explanations:

- A cage is an object containing a total and a list of cell indices that are occupied by the cage.
- A placement of a symbol in a cell is represented as an object containing a row, a column, and a symbol.
- The rows of the binary matrix are objects containing a list of placements and a reference to the cage it satisfies.
- The columns of the binary matrix are objects containing a list of references to rows that satisfy the column.

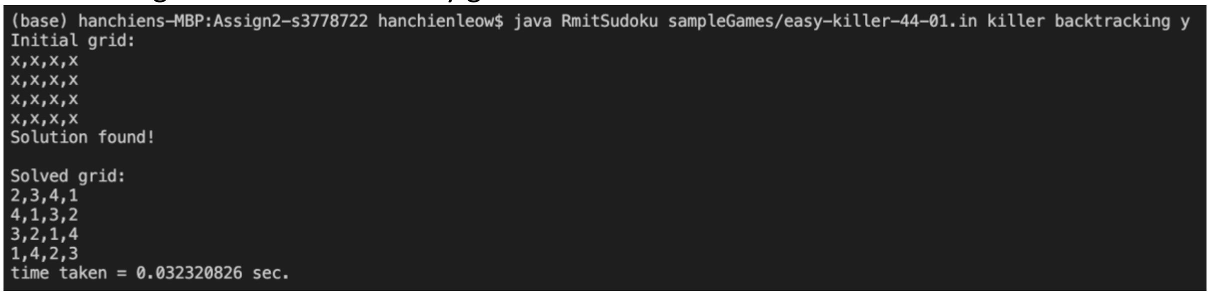
Implementations:

The implementation for the Killer Advanced Sudoku solver used the concept from Algorithm X, which utilizes the binary matrix representation. Also, the implementation is similar to Dancing Links, but instead uses Java's array list without the doubly linked list. The columns satisfied by a row can be obtained in constant time using the stored attributes of the row, so horizontal links are not required. Array list provides tools for easy iteration, so the rows of a column can be deleted without the need for vertical links.

Complexity Analysis:

Backtracking approach

Time complexity: O(N^(n*n)), where N is the number of possibilities for each square (i.e., 9 in classic Sudoku) and n is the number of spaces that are blank. Also, it is n*n because the backtracking is based on a 2d array grid.

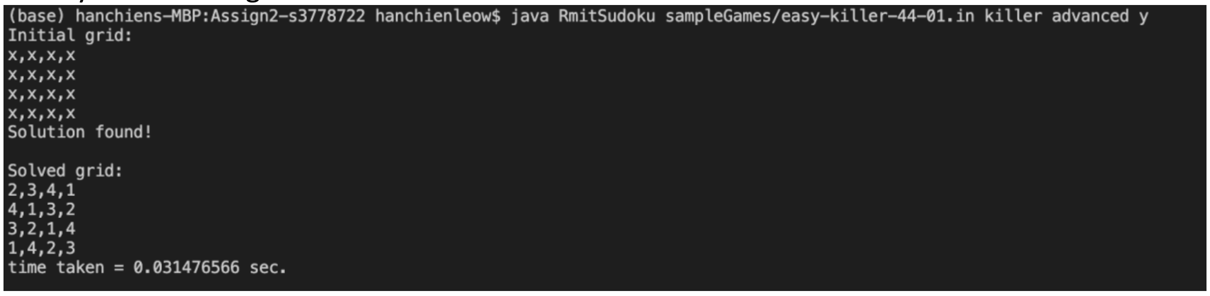


```
(base) hanchiens-MBP:Assign2-s3778722 hanchienleow$ java RmitSudoku sampleGames/easy-killer-44-01.in killer backtracking y
Initial grid:
x,x,x,x
x,x,x,x
x,x,x,x
x,x,x,x
Solution found!

Solved grid:
2,3,4,1
4,1,3,2
3,2,1,4
1,4,2,3
time taken = 0.032320826 sec.
```

Advanced approach

Time complexity: O(N^m), where N is the length / dimension of the sudoku and m as the size of the block. So, it will be O(N^3) for a classic sudoku. It is O(N^m) because this is mainly based on algorithm x.



```
(base) hanchiens-MBP:Assign2-s3778722 hanchienleow$ java RmitSudoku sampleGames/easy-killer-44-01.in killer advanced y
Initial grid:
x,x,x,x
x,x,x,x
x,x,x,x
x,x,x,x
Solution found!

Solved grid:
2,3,4,1
4,1,3,2
3,2,1,4
1,4,2,3
time taken = 0.031476566 sec.
```

Judging from the time complexity, the advanced approach should be more efficient than the backtracking approach. Furthermore, the time taken from both approaches proves that my justification is correct.