**3) Practice for final project defense presentation**

1. **The N-Queen problem has several ways it has been approached since it was first planted apart from brute force. Backtracking is probably the most known way to solve the problem using a depth first search or recursion. The problem can also be solved using metaheuristics and genetic algorithms.**

|  |  |
| --- | --- |
| **Value of N** | **Execution time (ms)** |
| 4 | 0 |
| 5 | 1 |
| 6 | 2 |
| 7 | 48 |
| 8 | 1080 |
| 9 | 28730 |
| 10 | - |
| N | O(n^n) |

1. **Reads the input (in this case a text file) with all the test cases and their board restrictions. It processes the input one board at a time, creating a list of all the illegal positions in the board. It calls the queens() method with this list and the size in order to compute the total number of possible solutions for the problem. This method permutes across all the possibilities of n-queens by trying to place a queen in each possible square. After each try of placing the queen the algorithm checks if it’s a valid placement through the puedoPonerReina() method. Once it places all the queens it means the board is complete. It is then checked against the list of illegal positions, making null any board that has queens on any of these spots.**
2. **The problem was solved through 3 methods containing a separate algorithm each but sharing the different data structures throughout the problem. Mainly there were two different data structures used in the problem. An Integer Array was used to represent both the board as a whole, and the positions of each queen within it, the index inside of the array would represent a column, while the value of said index would represent the row inside the column in which the queen was placed. Moreover, this facilitated and speed up the program since it eliminated all the cases where the queen shares the same column. The other data structured used was an ArrayList of Pairs. This list stored the row and columns of all the illegal positions the queens could not be at. The main() method takes the input, parsing it case by case, creating the ArrayList of all the illegal positions for a particular case to then see how many possible solutions there are. The queens() method receives both the ArrayList and the size of the board. It starts permutating across all possible solutions (brute force) by moving through each row and column and checking each possibility, and once its complete checks that the queens are not in any of the illegal positions. Lastly, the puedoPonerReina() method simply checks if a queen can be placed in a certain position without attacking other queens.**

***4) Practice for midterms***

1. *a*
2. *b*
3. *length-1*
4. *x+1, a[i]*