Project 1 Readme Team Farls

1	Team Name: Farls
2	Team members names and netids: Christian Farls (cfarls)
3	Overall project attempted, with sub-projects: 2-SAT Solver
4	Overall success of the project:
5	Approximately total time (in hours) to complete: 9
6	Link to github repository: https://github.com/christianfarls/farls2SATsolver/tree/main
7	List of included files (if you have many files of a certain type, such as test files of different sizes, list just the folder): (Add more rows as necessary). Add more rows as necessary.

File/folder Name	File Contents and Use
Code Files	
code/	2sat_farls.py analysis_farls.py
Test Files	
data/	2SAT.cnf.csv check_farls.csv
Output Files	
output/	output_farls.txt timing_results_farls.csv
Plots (as needed)	
plots/	2SAT_plot.png check_farls_plot.png

8	Programming languages used, and associated libraries: Python (time, pandas, matplotlib)
9	Key data structures (for each sub-project): A list is used to keep track of all of the clauses once they are parsed from the csv file. These lists contain clauses represented as a list of integers, where each integer represents a literal. Also, a dictionary is used to track the current assignment of

	variables during the unit propagation and DPLL recursion.
10	General operation of code (for each subproject): The code begins by parsing the input csv file and extracting each test individually. For each test case, the DPLL algorithm is used to determine if the formula is satisfiable. Unit propagation is first performed to simplify the clauses, then it tries different variable assignments recursively. The results are finally recorded to both a csv file (for a later plotting/visual analysis) and a txt file (for more detailed results).
11	What test cases you used/added, why you used them, what did they tell you about the correctness of your code. I created a 10-test check file to test if my algorithm was correct. This first showed me that my code was having trouble understanding satisfiability, as it was marking unsatisfiable tests as satisfiable and vice versa. By changing the values of my short test file, I was able to determine what the issue was and properly debug my algorithm.
12	How you managed the code development: I first started by reading articles and watching videos on the SAT problem to understand exactly what this algorithm attempts to accomplish. From here, I analyzed the provided test csv data and realized it would need to be parsed before an algorithm could be created. After finding a way to parse the data into a series of lists, I was able to write python code based on pseudocode that would correctly perform the DPLL algorithm for 2-SAT problems. To display the results, I decided the matplotlib library would be appropriate, as I've used it in previous courses. I implemented this to provide a visual display of the performance of the algorithm over the various test cases with an increasing number of clauses. Finally, I decided to output the detailed results to a txt file so I could calculate by-hand that my outputs were correct.
13	Detailed discussion of results: For the plot generated from the 100-test attempt, the results are as expected. As the number of clauses increases, the execution time increases. One interesting thing to note, though, was that there seems to be much more differentiation in the execution times of the unsatisfiable trials than the satisfiable ones. This is not necessarily surprising, though, since it would take more time to call a result unsatisfiable than satisfiable: for a satisfiable test, unit propagation would help eliminate most variables quickly for a faster result; for an unsatisfiable test, multiple iterations of variable assignments must be tested before a final conclusion can be reached. There is an outlier of a 47-clause satisfiable test, though, which is interesting. When replicating the test, there is always an outlier here, as well as in the 45-clause test sometimes. This is most likely due to the nature of the clauses, themselves, as unit propagation performance changes based on the nature of the problem. There is a generally linear increase in execution time, though, but it looks slightly exponential. A better idea could be found if tests with more clauses were provided and the graph could be expanded.
14	How team was organized: N/A, this was an individual assignment.
15	What you might do differently if you did the project again: If I performed this project again, I would create a much larger test suite and analyze those results as the clauses get into the hundreds or even thousands. While I expected

	the 100-test suite to take a while, especially with mosts tests upwards of 30+ clauses, every trial was performed in under a millisecond. A larger test suite could lead to a better idea of how performance changes as the number of clauses increases exponentially.
16	Any additional material: Resources I used: Article: 2-SAT - Algorithms for Competitive ProgrammingCP-Algorithmshttps://cp-algorithms.com › graph › 2SAT Video: https://youtu.be/Ku-jJ0G4tIc?si=3VL9epkBdL5gXDED Highly recommend the video, it did a great job of explaining the problem.