1) The members of the team.
    a. Me.

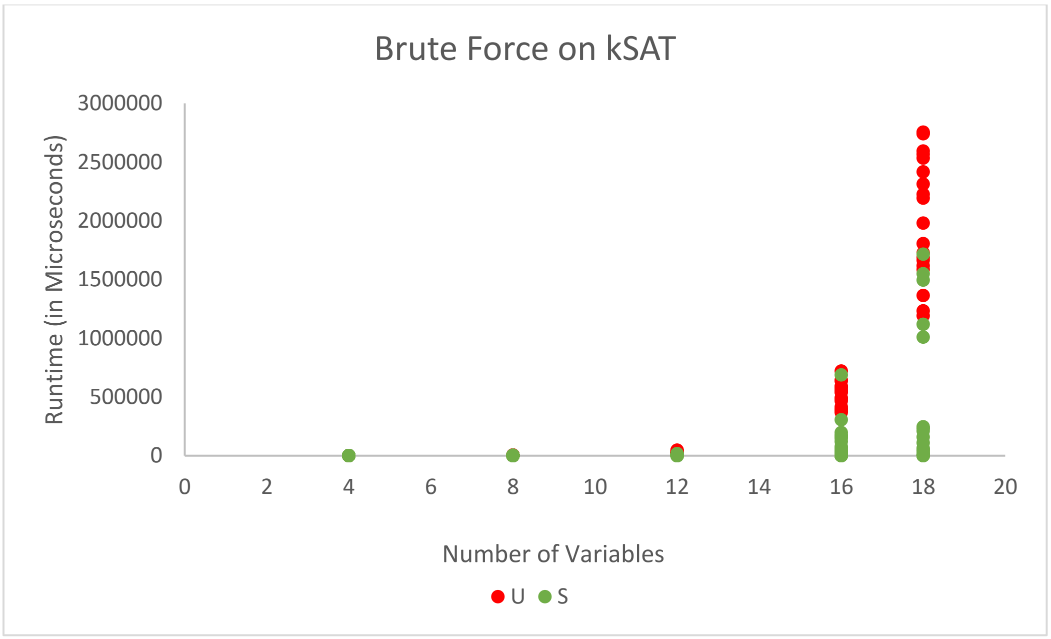
2) Approximately how much time was spent in total on the project, and how much by each student.
    a. I spent about 7-8 hours on the brute force section of this project.

3) A description of how you managed the code development and testing. Use of github in particular is a particularly strong suggestion to simplifying this process, as is some sort of organized code review.
    a. I used GitHub to document my updates to the assignment. I tested by creating a test.cnf file with particular tests taken from the kSAT.cnf for simplicity (like only first 50 tests) or because I got a different input (need to match the comment). Code development was just me writing and adjusting as I thought necessary.

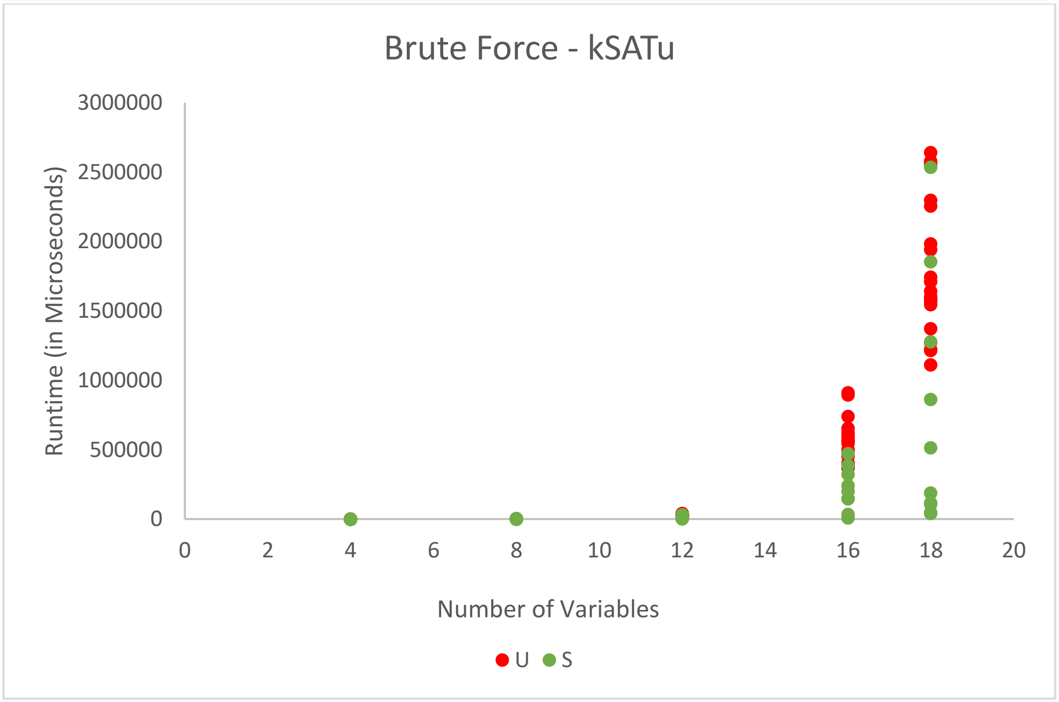
4) The language you used, and a list of libraries you invoked.
    a. I used Python with the time library for timing each wff and the xlwt library to write to a CSV file.

5) A description of the key data structures you used, especially for the internal representation of wffs, assignments, and choice point stacks.
    a. I used a class to hold the metadata for each CNF. The data was the max number of literals, number of variables, number of clauses, number of literals, what the answer was supposed to be (if commented), the wff itself, and the runtime. These were either all taken from the initial input from opening the file or calculated in main then added the CNF. Other necessary variables like the number of correct answers, number of satisfiable WFF's, etc. were calculated in main.

6) For each of the programs and each of the provided test files a chart of execution time versus number of variables. The points from the chart should come from your output run, with wffs that were unsatisfiable shown as red points, and wffs that were satisfiable shown as green. You may also want to use different symbols for wffs of different kSAT subsets (esp. 2SAT).
The representation of the output files as .csv was done to make it easy to import into a spreadsheet where graph generation is easy.

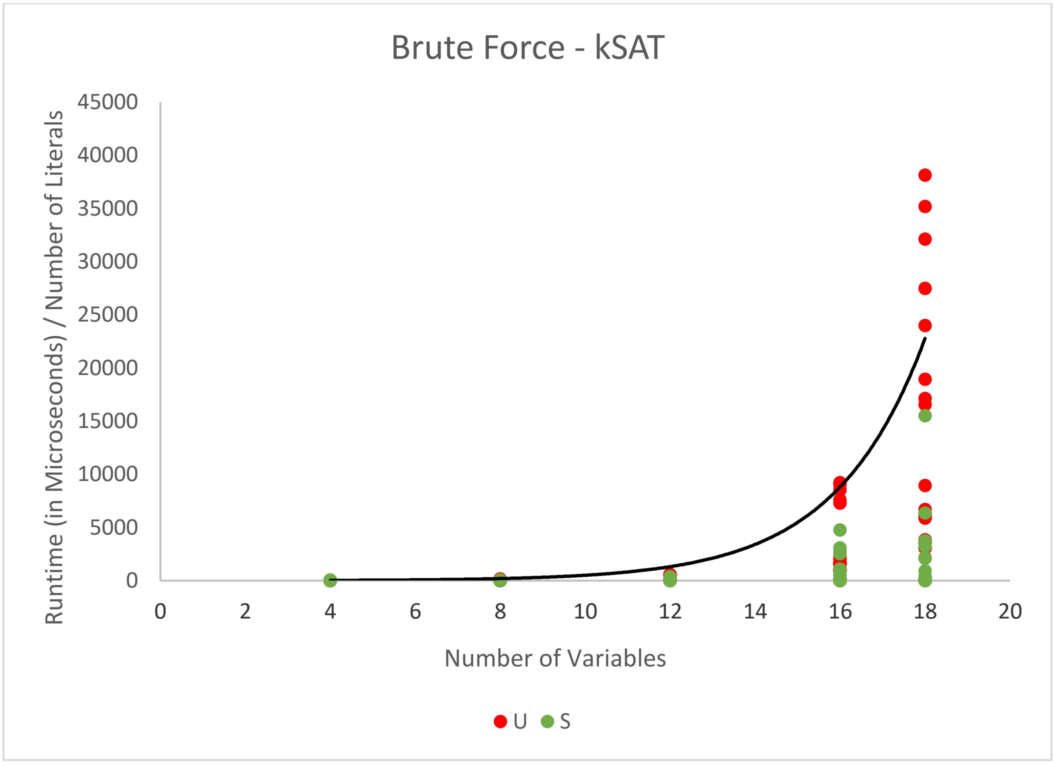


a.

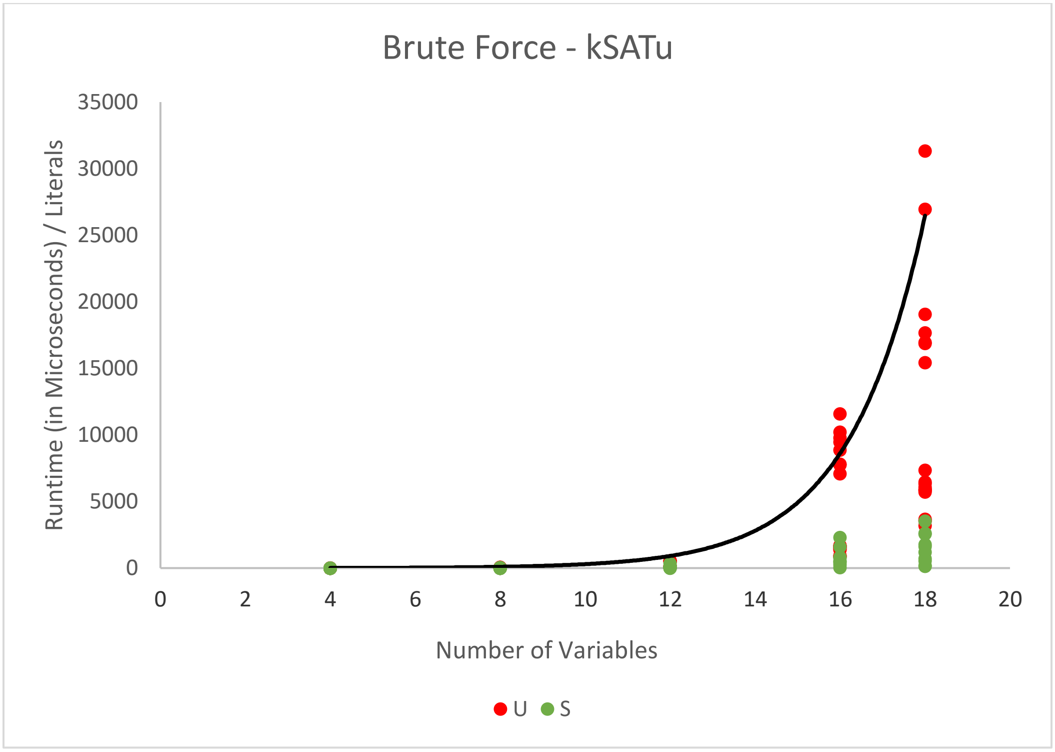


b.

7) From this data a curve fit to the worst-case times, again as a function of V. Here it may be useful as was done in class to divide the time by the total number of literals in each wff first.



a.



b.

8) A description of what you learned in terms of the relative complexity of the different solvers, especially as a function of the number of literals per clause. Especially important is what you observed in the transition from 2SAT to 3SAT and above.
    a. Obviously, I didn’t do any of the other solvers, but the complexity of the brute force algorithm was honestly fairly simple and straightforward. After all, brute force is simply generating inputs and checking whether the input works against the WFF. However, the other solvers would definitely be more complex, but also more efficient with less runtime. I would surmise that for the more literals per clause means that the more efficient algorithms would perform even more efficiently per literal (which makes sense, since each added literal adds to the runtime).

9) If you did any extra programs, or attempted any extra test cases, describe them separately.
    a. N/A