Team Members: Brooke Mackey, Thomas Mercurio, Jack Munhall
Time Spent Total:

- Brooke: 16 hours
- Thomas: 16 hours
- Jack: 16 hours

How did you manage the code development and testing?
We managed the code development with a shared Github repository. We created different branches for the three different sections (brute, backtracking and 2SAT) and pushed our changes to the relevant branch. To test our code and ensure best practices, we had code reviews after a certain amount of progress was made on a program (whenever the coder decided they wanted other people to look at it, generally). We tested our code by implementing a counter in each of our programs that would allow us to only run certain problems. This way, we did not have to wait for our program to run on the entire CNF file in order to know if it worked or not.

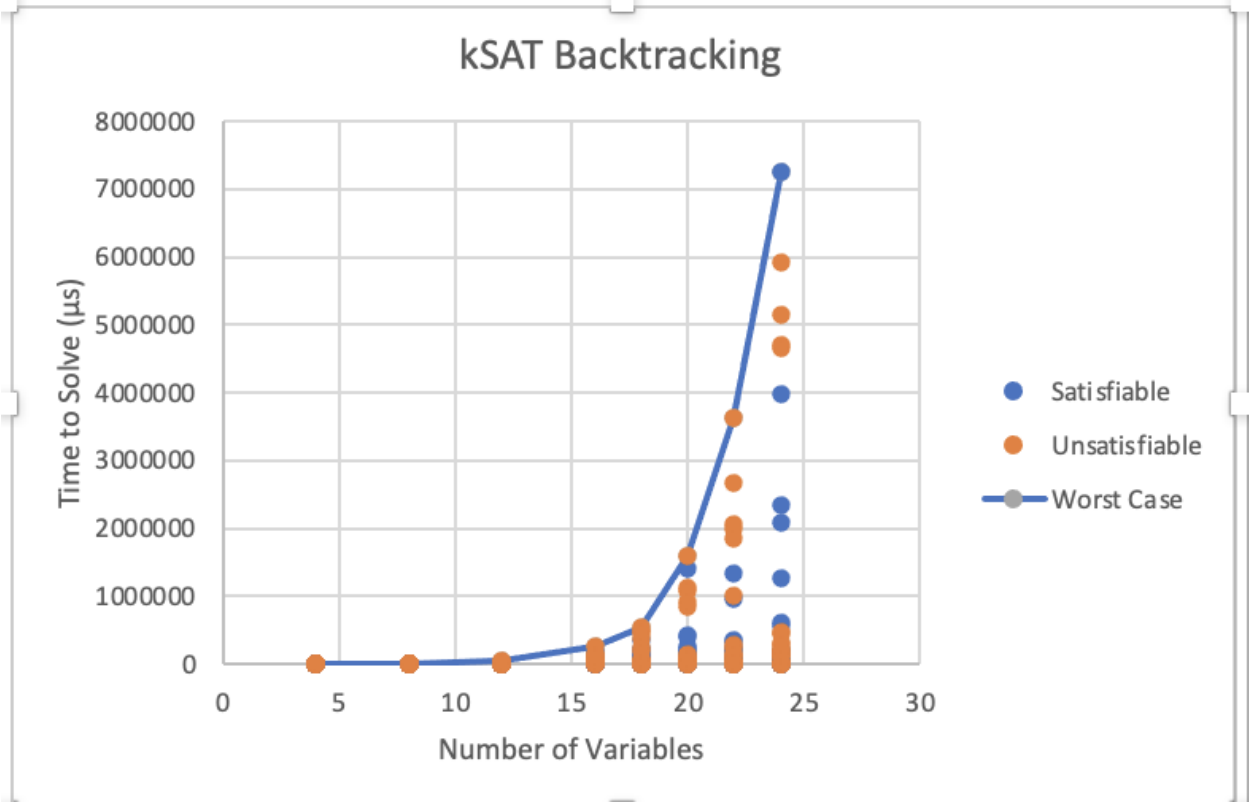
What language did you use and what libraries did you invoke?

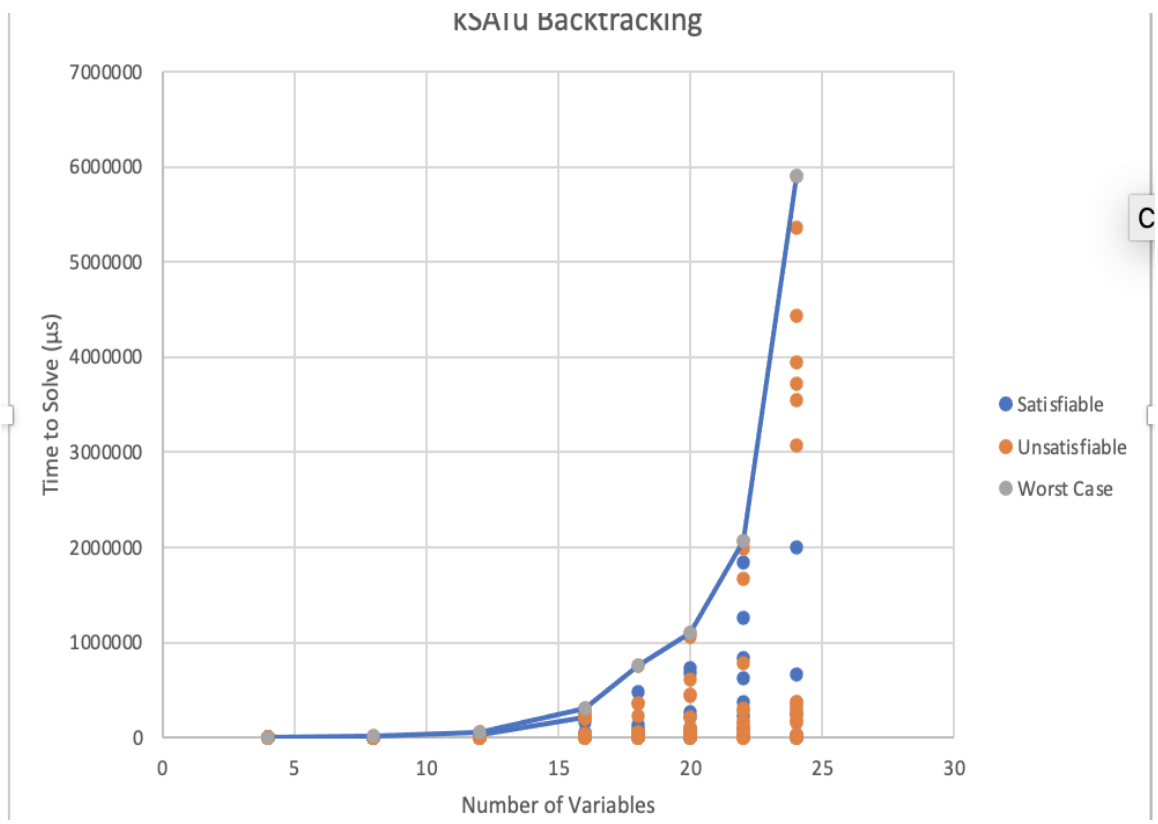
We used Python for our code, taking advantage of the sys, csv, and time modules for reading inputs, reading CSVs, and calculating the execution time of our solvers, respectively.

Describe the key data structures you used, especially for the internal representations of WFFs, assignments, and choice point stacks.

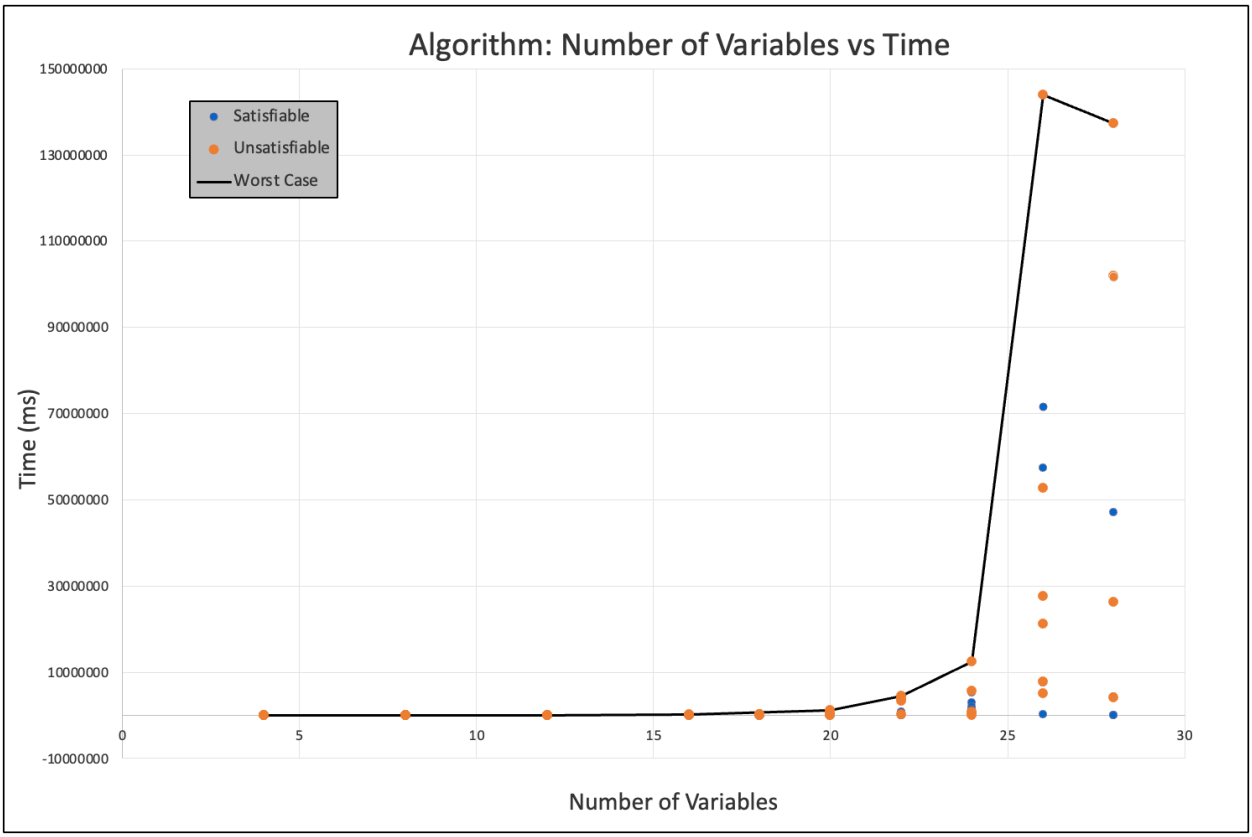
For all of the programs we read all of the inputs into a list of lists, storing the WFFs there. This made it easy to use embedded for loops to loop through each clause one at a time and check each literal in the clause as we went. We used binary numbers to represent our assignments and then used bit shifting to check our assignments for the brute and backtracking programs. For the 2SAT program, we used a dictionary to keep track of current variable assignments. Our backtracking program used recursion to keep track of choice points. Our 2SAT program used a list data structure in python (with the .pop() function, making it equivalent to a stack) to keep track of choice points. We kept iterating until the stack was empty.

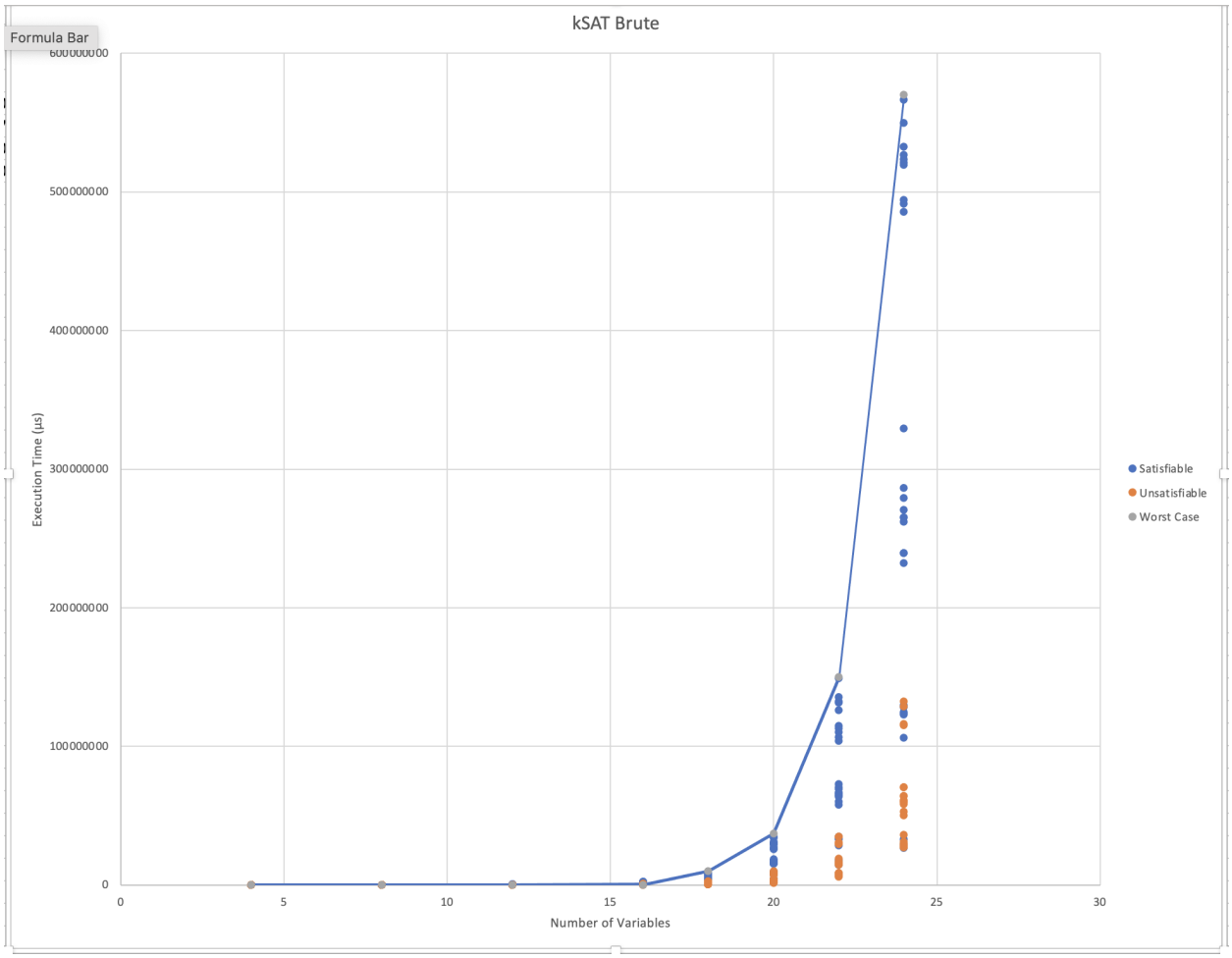
Execution Time vs. # of Variables Charts:

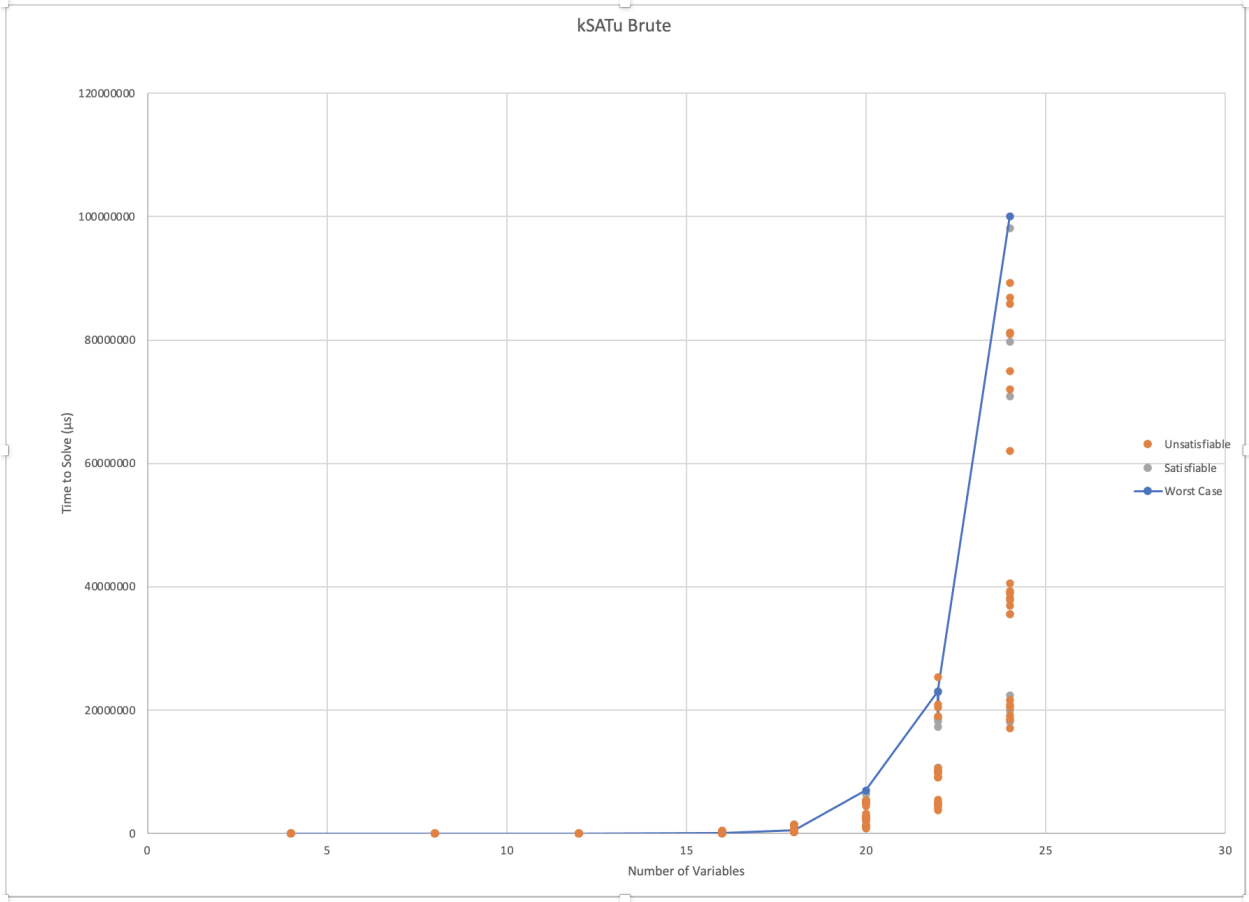

kSAT Backtracking
Time to Solve (µs)
8000000
7000000
6000000
5000000
4000000
3000000
2000000
1000000
0
0
5
10
15
20
25
30
Number of Variables
Satisfiable
Unsatisfiable
Worst Case


kSATu Backtracking
Time to Solve (µs)
7000000
6000000
5000000
4000000
3000000
2000000
1000000
0
0
5
10
15
20
25
30
Number of Variables
Satisfiable
Unsatisfiable
Worst Case

2SAT Graph:


Algorithm: Number of Variables vs Time
Satisfiable
Unsatisfiable
Worst Case
Time (ms)
150000000
130000000
110000000
90000000
70000000
50000000
30000000
10000000
-10000000
0
5
10
15
20
25
30
Number of Variables


kSAT Brute
Formula Bar
600000000
500000000
400000000
300000000
200000000
100000000
0
Execution Time (µs)
0
5
10
15
20
25
30
Number of Variables
Satisfiable
Unsatisfiable
Worst Case



What we learned about relative complexity:
In terms of complexity, we found that the brute force took the most computational power and time by a considerable margin. We found that the backtracking approach was significantly faster than the brute force and the 2sat solver was faster than both approaches. As the number of literals per clause increased, we found that the brute force seemed to take a lot longer relative to previous trials compared to the backtracking approach, which still did see large time differentials compared to previous trials with less literals, but not nearly to the same extent. As we transitioned from 2SAT to 3SAT and above, both of our approaches slowed down significantly, but backtracking still was able to solve the problems in a relatively helpful amount of time whereas brute force was probably not fast enough to be of practical use.

Extra programs:
N/A