Exercise 1: (DPLL) [5 Points] Simulate modern DPLL (from Slide 16 of Lecture 5 slides) by hand on the formula
\[ F = (x_3 \lor x_4 \lor \neg x_1 \lor x_5) \land (\neg x_3 \lor x_4 \lor x_5) \land (x_3 \lor x_4 \lor \neg x_1) \land (x_1 \lor x_2) \land (\neg x_1 \lor \neg x_5) \land (\neg x_3 \lor \neg x_4 \lor x_5). \]
Select branching literals in the order \(x_1, x_2, x_3, \ldots\). 

Exercise 2: (Stålmarck’s Method) [5 Points] Simulate by hand Stålmarcks Method (from Slide 8 of Lecture 6 slides) by hand on the formula from exercise 1. Select variables in the order \(x_1, x_2, x_3, \ldots\). 

Exercise 3: (CDCL) [7 Points] Simulate CDCL (from slide 24 of Lecture 7) by hand on the formula
\[ F = (x_1 \lor x_{13}) \land (\neg x_1 \lor x_2 \lor x_{14}) \land (x_3 \lor x_{15}) \land (x_4 \lor x_{16}) \land (\neg x_5 \lor x_7 \lor x_8) \land (\neg x_6 \lor x_7 \lor x_9) \land (\neg x_1 \lor \neg x_9 \lor x_{10}) \land (x_9 \lor x_{11} \lor x_{14}) \land (x_{10} \lor \neg x_{11} \lor x_{12}) \land (\neg x_2 \lor x_{11} \lor x_{12}). \]
Select branching literals in the order \(x_1, x_2, x_3, \ldots\). Draw the implication graph for each conflict and learn the 1-UIP clause. 

Exercise 4: (Local Search Challenge) [10(+10) points Points] Implement a (stochastic) local search SAT solver. Follow the SAT Competition input/output format [link](http://www.satcompetition.org/2004/format-solvers2004.html) For a working solver you get 10 points. The author of the best solver receives a bonus of 10 points. The solvers will be evaluated on satisfiable random 3-SAT problems. (like the ones here: [link](https://baldur.iti.kit.edu/sat/files/local-sat.zip)). You don’t need to start from scratch, use solver stub from the local-sat.zip package, it already contains the input parsing. 

Exercise 5: (Hidoku Challenge) [12(+12) Points] Hidoku a.k.a Hidato a.k.a Number Snake is a logic puzzle where the goal is to fill a grid with consecutive numbers that connect horizontally, vertically, or diagonally. The grid is rectangular and some of the cells are pre-filled. Example:

\[
\begin{array}{|c|c|}
\hline
1 & 5 \\
\hline
7 & 14 \\
\hline
16 & \\
\hline
\end{array}
\quad
\begin{array}{|c|c|c|c|}
\hline
1 & 4 & 5 & 2 \\
\hline
7 & 6 & 13 & 8 \\
\hline
11 & 12 & 14 & 9 \\
\hline
10 & 16 & 15 & \\
\hline
\end{array}
\quad
\begin{array}{|c|c|}
\hline
1 & 5 \\
\hline
2 & 14 \\
\hline
& 16 \\
\hline
\end{array}
\]

Unsolved Hidoku  \hspace{1cm} It’s solution  \hspace{1cm} Unsatisfiable Hidoku

The input is a single string looking like this (for the example above):

4,4:1,0,0,5;0,7,0,0;0,0,0,14;0,0,16,0;
The first two numbers are the width and height of the grid followed by the values separated by commas, rows are separated by semicolons, 0 represents an empty cell. The output format:

sol:1,3,4,5;2,7,6,13;8,11,12,14;9,10,16,15;

A Hidoku puzzle may be unsatisfiable, in that case output sol: UNSAT

Implement a SAT solving based Hidoku solver. For a working solver you get 12 points. The fastest solver will receive a bonus of 12 points. Here are some example inputs [link](https://baldur.iti.kit.edu/sat/files/hidokus.txt) for testing.