jonel107@student.liu.se
felli125@student.liu.se

Task 1

A:

Depth first search does NOT work in a timely manner at all. “Harder” takes over ten minutes. It is also unable to get a solution for easy at all. Depth first search is good for small searches where it can follow paths to find solutions, but difficult problems which exponentially get harder do not fit well for this type of search.

Backtracking search does work in a timely manner for the easy puzzle, anything larger and it takes far too much time. We do not know if it would actually reach a solution however, as we decided not to wait that long. Works similarly to depth first search but with only one variable assignment per node. It is able to backtrack when there are no legal variables left to assign, which makes it possible to avoid infinite loops. Since it works similarly to depth first search, but with some improvements, it is able to clear an easy level puzzle, but the more difficult it gets the time it takes to solve goes up exponentially (if it even solves it at all).

AC-3 search does work in a timely manner for easy, in fact it is very fast, but for the harder and hardest difficulties it could not come to a solution. After applying AC-3 you are left with an equivalent CSP of the original one, but the arc consistent CSP which will, in most cases, be faster to search because its variables have smaller domains. When we have many cells it has a harder time making a decision because there isn’t enough information to go off of to check for inconsistencies. If it can’t make a decision it cannot solve the issue, therefore it will fail to solve the puzzle.

Min conflicts search was also quite fast, but it does not get a solution no matter the difficulty. It is often quite effective. It selects the value which violates the least constraints, which avoids as many conflicts as possible. For each square/variable it checks for all possible conflicts that occur *at the time*. The issue is that the empty slots, which will later have to be filled with numbers, might lead to future conflicts if the wrong number is chosen. This makes it rather easy to get stuck, as choosing the wrong number even ones can lead to the whole puzzle failing.

So, which works in a timely manner? For easy difficulty: AC-3 and Backtracking search. For harder: none of them finished with a solution (at least those who finished faster than over 10 minutes). Hardest: None of them finished (and some were simply not attempted).

Depth first graph search
Easy: 0:01:14.666789
However: failed-domain
Harder: Over 10 minutes
Hardest: Not attempted

Backtracking search
Easy: 0:00:24.581022
Harder: Over 10 minutes
Hardest: Not attempted

AC-3
Easy: 0:00:0.020007

jonel107@student.liu.se
felli125@student.liu.se

Harder: 0:00:0.021998
However: failed-domain
Hardest: 0:00:0.021005
However: failed-domain

Min conflicts
Easy: 0:00:54.259086
However: failed-domain
Harder: 0:00:53.743398
However: failed-domain
Hardest: 0:00:53.730372
However: failed-domain

B:
Forward checking makes the solution faster and also makes it possible to solve the harder and hardest puzzle. Surprisingly, the harder possible took longer time than the hardest. This might be because of how the different orders of numbers will create different difficulties for the solution.

Minimum remaining values took a long time to solve the puzzle on its own, long enough that we stopped it after five minutes before a solution could be found. On its own it is not good for use in solving sudoku.

Forward checking + minimum remaining values together is by far the best solution when looking at the results. It is the fastest and has no problem getting the results no matter the difficulty. *It is clear that this is the best combination for solving sudoku problems.*

BT:
Easy: 0:00:24.581022
Harder: Over ten minutes, stopped
Hardest: Over five minutes, stopped

BT+FC:
Easy: 0:00:0.011002
Harder: 0:00:48.623949
Hardest: 0:00:0.265059

BT+MRV:
Easy: Over five minutes, stopped
Harder: Not even attempted
Hardest: Not even attempted

BT+FC+MRV:
Easy: 0:00:0.005002
Harder: 0:00:0.012001
Hardest: 0:00:0.050008

jonel107@student.liu.se
felli125@student.liu.se

Task 2

A:

Depth first search: At around 22 it starts to take around a minute to solve.

Backtracking: At around 29 seems to be the limit of where it will solve the problem within a reasonable amount of time.

AC-3: At around 500 the process took approximately a minute, while 600 took a bit over 2 minutes. 1000 took far too long to complete.

Min Conflicts: You can go as high as you'd like, there isn't really necessarily a cap of how high you can set n, it will solve the problem in about the same amount of time each time.

Hard to say if the question is how many to pick before reaching the error statement, because most, if not all, algorithms can handle large numbers, it'll just take an unreasonably long time. If we are looking for what numbers work within a reasonably short time, that is entirely different. We have therefore mostly focused on the "within reasonable time" train of thought.

Some algorithms start with random cells, which is why some have a harder time with different n:s. In order of best to worst: **Min Conflicts**, **AC-3**, **Backtracking** and finally **Depth first search**. **Depth first search** is not optimal (the worst) for this as searching down one path for a goal node is not optimal for deciding each move in chess. **Backtracking** has a similar issue as it will do pretty much the same as **depth first search** but with the ability to go back if a move is illegal. **AC-3** only checks at the closest possible options (one move in each possible direction) and does not take future moves into account, this makes it work perfectly well for a problem such as chess where you only want to move in one direction at ones. **Min conflicts** is the absolute best as it looks for the option with the least weight possible and makes that most optimal choice without having to calculate much else.

B:

n = 20
BT: 0:00:02.198105
BT+FC: 0:00:01.205865
BT+MRV: 0:00:00.020117
BT+FC+MRV: 0:00:00.002009

BT + FC + MRV gives the best result, at least when looking at n = 20.
FC makes sure not to move the current node before looking further along the tree whether there exists a legal value to move to. It does not detect all failures, but it does still help.
MRV selects the variable with the smallest domain and will immediately detect failure if the variable does not have a legal value.

The two help each other help each other out and cover issues that the other cannot handle, making the result a lot faster and more efficient. The reason why **MRV** is better than **FC** in this case is since in Chess it is more important to look at the best possible current move rather than every other possible move after it.

jonel107@student.liu.se
felli125@student.liu.se

C:
The runtime of min conflict is *independent* of the problem size. NQueens is fairly easy for local search because of the relatively high density of solutions in state space. No matter what size n it should be able to solve the puzzle in 50 steps.

D:
If you're working with a puzzle which contains a lot of rules, using constraint-based problem solving is quite useful. When using traditional means it is harder to find the solution as we do not know exactly what we are looking for, resulting in searching every single possible move. By adding constraints we limit the amount of moves we need to consider, making the solution work a lot faster. However, depending on the problem it can be rather difficult to find a search algorithm that works for all possible versions of a puzzle/problem, making it more complex when it comes to deciding on what approach to use.