Comment of part A:

Simple Backtracking:

For low values of queens, it is fast in returning a result, but the time grows exponentially, making it inefficient for larger values of N.

- N=6: 0.000417 seconds
- N=8: 0.001901 seconds
- N=10: 0.002607 seconds
- N=12: 0.007228 seconds

This approach becomes inefficient for N-queens > 12.

Reasoning:

This inefficiency arises because **we do not prune invalid paths**, resulting in redundant operations. It is a blind search, as we explore **without any heuristics**.

Backtracking with Forward Checking

This approach is faster than simple backtracking for values of N < 12.

- N=3 to N=5: Computation times are lower than Simple Backtracking, around 0.00003 to 0.000039 seconds
- N=6 to N=12: Shows improved performance over Simple Backtracking
- For N > 12, performance results vary

Reasoning:

Forward Checking eliminates some invalid options, allowing for faster search by pruning certain values. However, it does not apply variable ordering or value selection heuristics, limiting its efficiency compared to more advanced methods.

Forward Checking with MRV and LCV Heuristics

N=3 to N=5: Slightly higher times than Forward Checking alone.

N=6 to N=12: Times are similar to Forward Checking.

N=13 to N=29: Demonstrates significantly better performance than Forward Checking, with computation times remaining low even as N increases.

- N=14: ~0.001501 seconds
- N=16: ~0.006589 seconds
- N=18: ~0.002233 seconds
- N=20: ~0.003165 seconds
- N=22: ~0.002273 seconds
- N=28: ~0.006698 seconds

Reasoning:

- MRV: Selects the column with the fewest constraints.
- LCV: Chooses the row with the fewest constraints
- Improved pruning: The combination of MRV and LCV enhances pruning, effectively reducing the search space and leading to faster solutions

Comparing the performance of simple backtracking and forward checking coupled with MRV and LCV heuristics highlight the differences in performance and extendability. Backtracking seems to work for small values of N, however, adoption such strategy with respect to larger values is not viable since N grows fast without any restrictions, due to the fact of performing a straightforward exhaustive search. There is a reason why this method does not work well for large N, say N greater than 12, the reason being the endless increase of N which causes time to increase as well, that is, at an increasing rate. In comparison, forward checking in combination with the above heuristics of MRV and LCV while solving CSP possesses clear benefits because of precautions that are taken in order to select paths without many constraints enabling further trimmings of the search space. This style allows better performace in terms of reducing the search space and the turnaround time is kept low even as N increases. MRV helps to target the columns with the least amount of options left while LCV looks at the row effect and is directed at guiding the search and avoiding the excessive exploration of the search space. It follows then that it is forward checking with MRV as well as LCV which is not just quicker but it is also reasonable for larger sizes input regions since exhaustive search would be inordinately difficult and time-consuming to sustain in such instances.

Part B:

Propose a state representation:
We represent the state of the N-Queens problem as a one-dimensional list or array state, where [i] representsthe i-th row.

state=[1,3,5,7]
Queen of column zero is in row 1
Queen of column 1 is in row 3
Queen of column 2 is in row 5
Queen of column 3 is in row 7

Initial State Proposal: Generate a random permutation of row indices for the queens, ensuring that no two queens share the same row.

Reason: By ensuring no two queens are in the same row, we eliminate row conflicts from the start.

Propose Actions (Tweaking) to Make the Run Time Faster:

Move a Queen Within Its Column, change the row position of the queen to any other row

Reason:
Limited Move Set: This reduces the branching factor

Efficient Neighbor Generation: Only N-1 possible moves per queen, making it computationally efficient to explore neighbors

Propose Heuristics for Making the Moves:

The heuristic h(state) is defined as the total number of pairs of queens that are attacking each other.
min-conflicts (number of attacks).

Results Summary Using Simulated Annealing

Key Findings:

- Scalability: Simulated Annealing successfully found solutions for N ranging from 4 to 100

- Average Computation Time: The average time increased gradually with N but remained within acceptable limits even for large N

Solution Existence: For N ≥ 4, the algorithm consistently found solutions in most runs. For N = 2 and N = 3, where solutions do not exist, the algorithm correctly indicated failure

Based on the results obtained for solving the N-Queens problem with values of N, it is evident that the heuristic method of Simulated Annealing significantly outperforms the previous backtracking methods in terms of scalability and efficiency. While Simple Backtracking and Backtracking with Forward Checking become impractical for larger N due to exponential increases in computation time and resource consumption, especially without the aid of heuristics, Simulated Annealing maintains relatively low average computation times even as N increases, demonstrating a more linear growth pattern. The addition of LCV to backtracking algorithms does enhance their performance by intelligently guiding the search and pruning the search space, yet they still struggle with scalability and can exhibit significant variability in computation times for larger N values. In contrast, simulated annealing employs a probabilistic approach that allows it to escape local minimal and effectively explore the solution space, consistently finding solutions across multiple runs for large N. Therefore, when comparing these methods, it becomes clear that heuristic approaches like Simulated Annealing offer a more efficient and scalable solution to the N-Queens that has large values of N.

Part C:

Nature of Genetic Algorithms:

Population-Based Search: maintain a population of candidate solutions, which can be computationally intensive to process, especially for larger population sizes

Iterative Process: The algorithm evolves the population over many generations, and the total computation time increases with the number of generations

N	Algorithm	Average Time (seconds)	Solution Exists
3	Genetic Algorithm	0.001634	TRUE
4	Genetic Algorithm	0.001701	TRUE
5	Genetic Algorithm	0.001978	TRUE
6	Genetic Algorithm	0.002201	TRUE

Genetic algorithm is a very efficient method to solve the N-Queens problem for board sizes ranging between N=3 and N=6, where the algorithm generates a solution each time N increases, and the average time of calculation gets larger.

It is then logical to say that it finds these solutions in much lesser time than the latter. Looking at its advantages how it organizes to perform a search in an optimal time frame in a solution space, the genetic algorithm is one right step ahead among other cybernetics which are already developed.

Take for example methods like backtracking. While they are also known as systematic search methods, the search space explored is imperative in the particularity of searching for the most attractive configuration.

With such exhaustive search, backtracking then seems unable to zero on a point as N increases and as a result the time taken for the case of large values of N becomes unmanageable for real time application in simulations of large chess boards.

The genetic algorithm is able to avoid such hassle to some extent, thanks to a remaining portion of the problem solving processes that are beyond mechanics that is, this algorithm has elements of other heuristics such as simulated annealing that is search space limitations are taken into account. Unlike most other heuristic methods, in this case of the genetic algorithm, it integrated with the traditional approach of simulated annealing so that it can cultivate the search as well as the overcoming of local optimum.

The aim of the search in the genetic algorithm is to explore many paths simultaneously. In the end the genetic algorithm has the capacity to look into multiple directions at the same time which means that the chances of achieving the global optimum in shorter time are more. Taking it as a whole, the genetic algorithm once more comes our first hand in a very efficient way for the N-Queens problem particularly as the size of the board increases.

Its performance is enhanced with respect to finding solutions within shortest times with no appreciable increase in performance time as opposed to the usual methods like implementing backtracking and other heuristics such as simulated annealing.

Appendix: Source of code part A: https://www.geeksforgeeks.org/n-queen-problem-backtracking-3/