Homework #1

Q1. The eight queens problem

(a) How big is the phenotype space for the eight queens problem?

In the eight queens problem, the goal is to place eight queens on an 8×8 chessboard such that no two queens threaten each other. This means no two queens can share the same row, column, or diagonal.

The **phenotype space** refers to the set of all valid configurations (solutions) that satisfy the problem's constraints.

Before applying the constraints (including invalid placements):

Choose 8 places from the total 64 spaces for the 8 queens.

$$C(64,\ 8)\ =\ 64!\,/\,(8!\ 56!)\ =\ 4426165368$$

After applying the constraints (only valid placements):

The actual number of valid solutions where no two queens threaten each other is much smaller. The number of valid solutions (phenotypes) for the eight queens problem is 92.

(b) Give a genotype to encode the 8x8 chessboard configuration.

One efficient encoding is using a **permutation-based genotype**, where the genotype represents the column position of each queen, with the index of the array representing the row.

Consider the following genotype: [0, 4, 7, 5, 2, 6, 1, 3]

This represents the following chessboard configuration:

- Queen 1 is placed in row 0, column 0.
- Queen 2 is placed in row 1, column 4.
- Queen 3 is placed in row 2, column 7.
- Queen 4 is placed in row 3, column 5.
- Queen 5 is placed in row 4, column 2.
- Queen 6 is placed in row 5, column 6.
- Queen 7 is placed in row 6, column 1.
- Queen 8 is placed in row 7, column 3.

This way, no queen will be placed in the same row or column, effectively reducing possible configurations.

(c) How big is the genotype space you give in (b)?

This encoding reduces the problem space to just $8! = 40,320$ permutations, which avoids placing two queens in the same row or column.

(d) Briefly describe why the proposed genotype is able to cover the phenotype space.

The proposed **permutation-based genotype** effectively covers the **phenotype space** of the eight queens problem because it directly encodes valid row and column placements, reducing unnecessary exploration of invalid configurations:

1. **Row Constraint**: Since each element in the genotype corresponds to the column position for a specific row, no two queens can be placed in the same row. The mapping from the array index to the row ensures this.
2. **Column Constraint**: The genotype is a permutation of integers from 0 to 7, ensuring that no two queens share the same column, since each integer uniquely represents a column.
3. **Diagonal Checking**: While the diagonal constraint isn't automatically enforced by the genotype, it allows the entire phenotype space to be represented. Any permutation can be checked for diagonal conflicts, so all possible valid solutions (phenotypes) are reachable by searching through the genotype space.

Thus, by using this encoding, the entire valid phenotype space (the 92 solutions) can be explored, while significantly reducing the number of invalid configurations that would otherwise be generated with a more naive encoding approach.

Q2. Precision Calculation

Given a function f (x) : [0, 1] → R. We want to find an optimal x value with a required precision of 0.001 of the solution. That is, we want to be sure that the distance between the found optimum and the real optimum is at most 0.001. How many bits are needed at least to achieve this precision for a bit-string genetic algorithm?

To achieve a precision of 0.001 in the range [0,1], we need to divide the range into 1000 parts (since 1 / 0.001 = 1000).

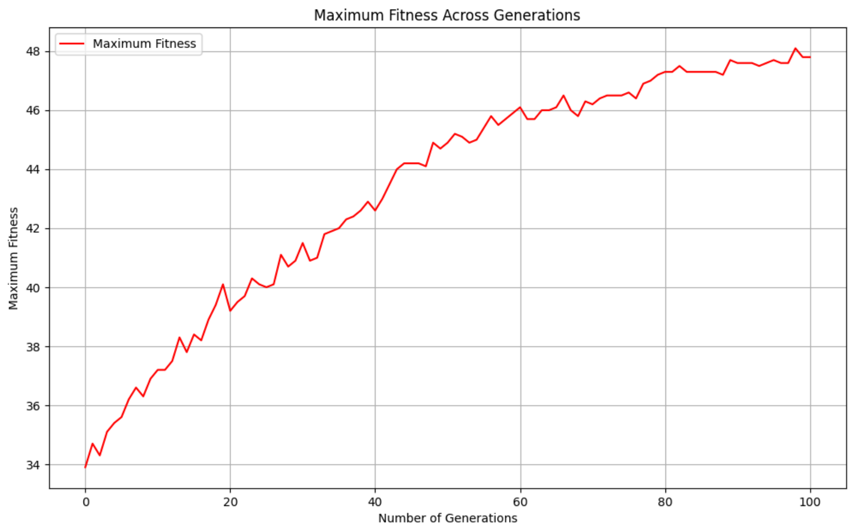
To represent 1000 different values, we need at least 10 bits, because $2^{10} = 1024$, which is the smallest power of 2 greater than 1000.

Therefore, we need 10 bits to represent values with a precision of 0.001.

Q5. Compare the results for problems 3 and 4

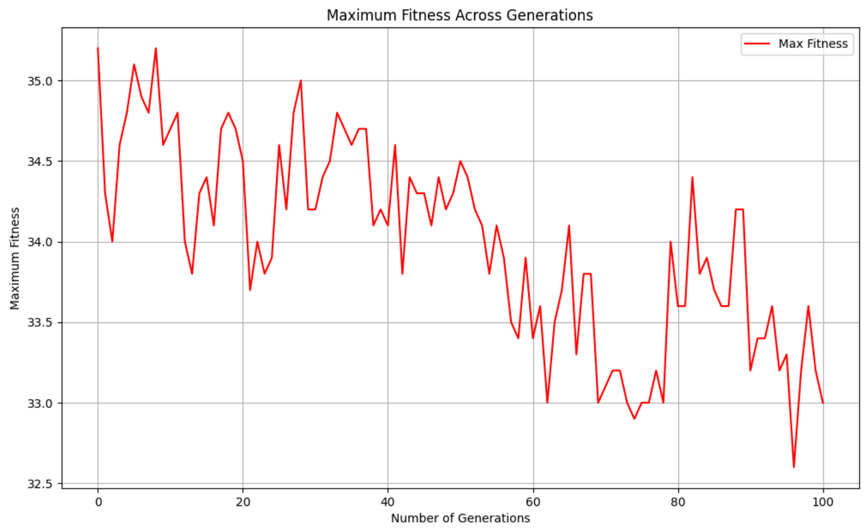
Problem 3 (Original Fitness Function)

- The plot below illustrates a consistent increase in maximum fitness across generations. This pattern is typical for a genetic algorithm solving the OneMax problem, where the objective is to maximize the count of 1s in a bit sequence.
- As generations advance, the algorithm becomes more adept at finding solutions with a higher number of 1s, demonstrating its effectiveness and efficiency.



Problem 4 (Modified Fitness Function)

- This plot shows significant fluctuations and lacks a clear upward trend. In fact, it even trends downward at times, indicating that the algorithm is having difficulty making steady progress.
- This behavior arises because the altered fitness values make it harder for the algorithm to distinguish between good and bad solutions, hindering its ability to concentrate on improvement across generations.



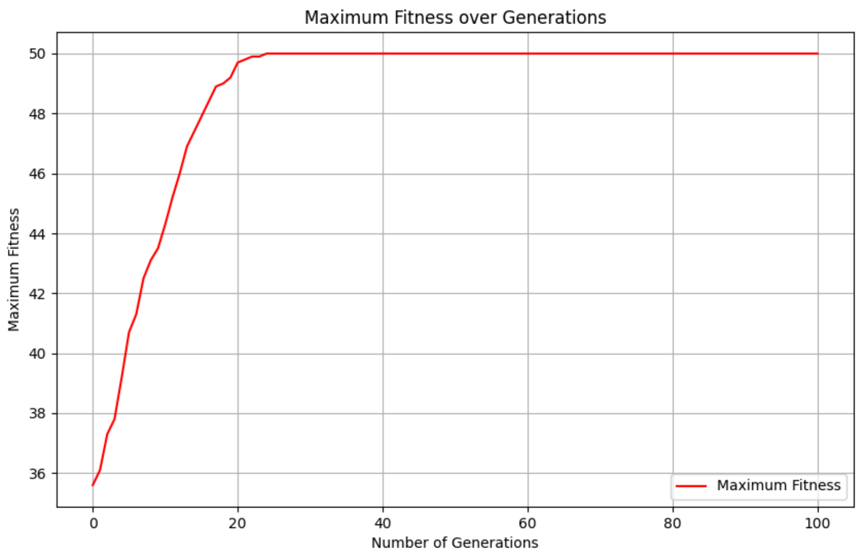
Difference between Problem 3 and Problem 4

- Clear Trend vs. Fluctuated Progress
    - In the original version, the algorithm showed a distinct and consistent progression, demonstrating that it was effectively evolving towards better solutions.
    - The modified version displayed signs of fluctuated progress, with fitness values varying without a clear upward trajectory.
- Selection Effectiveness:
    - The modifications to the fitness function diminished the disparities between the fitness scores of different solutions, making it more challenging for the algorithm to reliably choose the best individuals.

Q8. Compare the results for problems 6 and 7.

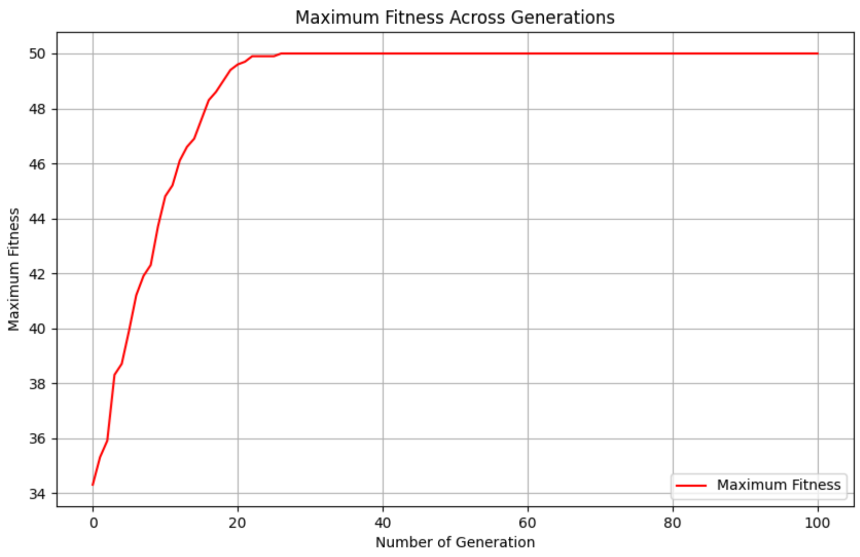
Problem 6 (Original Fitness Function)

- This plot demonstrates a consistent increase in maximum fitness across generations. This pattern is typical for a genetic algorithm addressing the OneMax problem, where the objective is to maximize the count of 1s in a bit sequence.
- As the generations progress, the algorithm improves at identifying solutions with a higher number of 1s, indicating its effectiveness and efficiency.
- It converges in the early generations because tournament selection can quickly lead to a homogeneous population, resulting in rapid convergence.



Problem 7 (Modified Fitness Function)

- This plot displays a comparable trend to the original, showing a consistent increase in maximum fitness across generations. It also converges in the early generations for the same reason.



Difference between Problem 6 and Problem 7

- Fitness Evaluation
    - The primary difference between the two fitness functions lies in their fitness evaluation functions. The second one has an additional 1000 added to the original function. However, since we use tournament selection, the modification of the fitness function will not impact the results.

Q9. Compare the results for problems 3,4,6, and 7.

Table Overview

#	Method	Fitness Function
Q3	Roulette Wheel	Sum(bits)
Q4	Roulette Wheel	Sum(bits) + 1000
Q6	Tournament	Sum(bits)
Q7	Tournament	Sum(bits) + 1000

Roulette Wheel (Problem 3 and Problem 4)

- **[Problem 3]** With the sum of bits as the fitness function, the algorithm demonstrated rapid convergence and high stability. The final fitness value was moderate.

- **[Problem 4]** When the fitness function was modified to sum(bits) + 1000, the convergence rate slowed, and the maximum fitness value decreased.

Tournament Selection (Problem 6 and Problem 7)

- **[Problem 6]** With the sum of bits as the fitness function, tournament selection resulted in rapid convergence, leading to a uniform population in the final stages due to the nature of tournament selection.
- **[Problem 7]** For the sum(bits) + 1000 fitness function, the algorithm produced the same result as in Problem 6. Since modifying the fitness function does not influence tournament selection, the overall outcome remained unchanged.

Conclusion

The results show that tournament selection converges more quickly than roulette wheel selection but tends to reduce population diversity at a faster rate. This is because tournament selection favors the fittest individuals, leading to faster, though potentially premature, convergence.

Furthermore, altering the fitness function (from sum(bits) to sum(bits) + 1000) had a significant impact on the performance of roulette wheel selection, resulting in a different convergence rate and fitness outcomes. However, for tournament selection, the change in the fitness function had little to no effect on the final results. This indicates that tournament selection is less sensitive to fitness function modifications compared to roulette wheel selection, yielding consistent outcomes regardless of fitness scaling.