Discuss the difference between the two implementations

The main differences between the two versions are the settings - mutation rate and population size, and how the crossover and mutation functions are implemented. Mutation rate and population size can be easily changed in both implementations.

Crossover is a lot more simple in version 1. There is a random split ranging from 0 to 8 in the crossover for every time a child is created. There is a 2/8 chance that the child would be a clone of one of its parents as the crossover split could be before the first value or after the last value.

Crossover in version 2 is a lot more complex. This pushes the child into having more unique values by prioritising the values on the right side that don't exist on the left side to be appended. This makes sense as a queen piece can't be on the same row as another queen piece. Because of this guided approach, it takes on average fewer generations to solve the problem.

Mutation is more chaotic in version 1, just leaving it up to random chance to decide whether or not a value changes to a completely random one. This isn't ideal as a value at one index could change to a value that already exists in another index, resulting in a lower fitness score.

Version 2 only provides a chance of swapping the indexes of the values, rather than changing them to a random one. This keeps the crossover's work of guiding the values towards being unique at each index. With a mutation rate of 0.08, there is only a 16% chance of two values getting swapped.

Overall Version 2 performs better on average because of its guided approach, rather than a more stochastic method.

Below is a table with the list of differences on each function.

Function	**8Q-V1**	**8Q-V2**
Imports		Version 2 imports the time library. This doesn't affect the results.
__main__	Calls the main() function	Runs the code through the __main__ function
Main function	Population size = 100 Generation = 1	Population size 50 Generation = 0
Class State	Mutation rate = 0.4	Mutation rate = 0.08

__init__	Uses list comprehension to create the random 1 * 8 array. This has slightly faster computation time due to the fewer steps.	Creates a random 1 * 8 numpy array and then converts it to a list.
fitness	These are exactly the same	
probability	These are exactly the same	
crossover_naive	Not included.	Included but never called.
crossover	Crossover is split randomly (0,8) every time.	There's multiple layers to this crossover. First it gets a random (0,8) crossover index for both parents. Parent 1 is the left side of the crossover, parent2 is the right. Then it extends the missing right values from parent 2 to the right side, and then the missing right values from parent 1 to the right side. So this keeps it in a similar pattern, but changes the order. This can create a list larger than 8 if the left crossover index is too low. It then does a while loop to shuffle things further, resulting in up to two splits. The values before the first split are the values from the left side, the values after the first split are the values from the right side, preferring to add values that aren't present before the first slice. If it's incomplete, the values after the second slice are the first missing values from the parent1.
mutate_naive	Not included.	Included but never called.
mutate	Each value in the child has	Mutation is a lot more

	a mutate rate (0.4) chance of mutating. There is a high chance that the child will be very different from the parent.	uncommon here. Each value has a 0.08 chance of being selected for mutation. If only 1 value is selected (k being the number of selections), no mutation occurs, if two are selected, the indexes the values are positioned in get swapped.
__str__	These are exactly the same	
Pick random by probability	These are exactly the same	
Generate next population	These are exactly the same	

Split code (based on function definition of 8Q-V1 and 8Q-V2) and present in individual cells.

(See code on github)

Explain the roles of each function of both versions 8Q-V1 and 8Q-V2.

(See text above each code cell on github)

Explain the routine call of the functions that are present in 8Q-V1 and 8Q-V2.

1. __init__ function is run and the State object and its properties are saved to memory.
2. Main function is run, creating state objects with random values.
3. While loop condition in the __main__ function is checked.
4. Fitness function calculates a fitness score.
5. If the fitness score < 28 - generate the next population.
6. generateNextPopulation gets states from the probability function
    a. Probability function assigns a probability value to the states. Higher fitness values give lower probability values.
7. generateNextPopulation selects parents from the pickRandomByProbability function
    a. pickRandomByProbability picks a random number and returns the first state that has a probability value less than random.

8. generateNextPopulation appends parents to State object
9. State object saves parents to parent1 and parent2 variables
10. Variables are run through the crossover function.
11. Mutate function is run.
12. Fitness function calculates fitness score
13. If statement in main is checked again. Repeat steps 5 to 12 if not 28.
14. Once fitness is 28, print parent values and print the solution state.
15. Printing these objects prints the board visualisation in the __str__ function.