**N-Queens Problem**

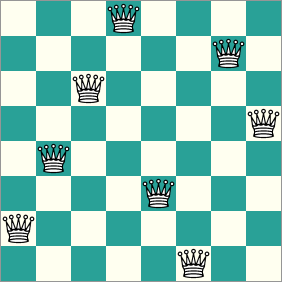
* **Genetic Algorithm :**

Genetic Algorithms are a family of algorithms whose purpose is to solve problems more efficiently than usual standard algorithms by using natural science metaphors with parts of the algorithm being strongly inspired by natural evolutionary behavior; such as the concept of **mutation**, **crossover** and **natural selection**.

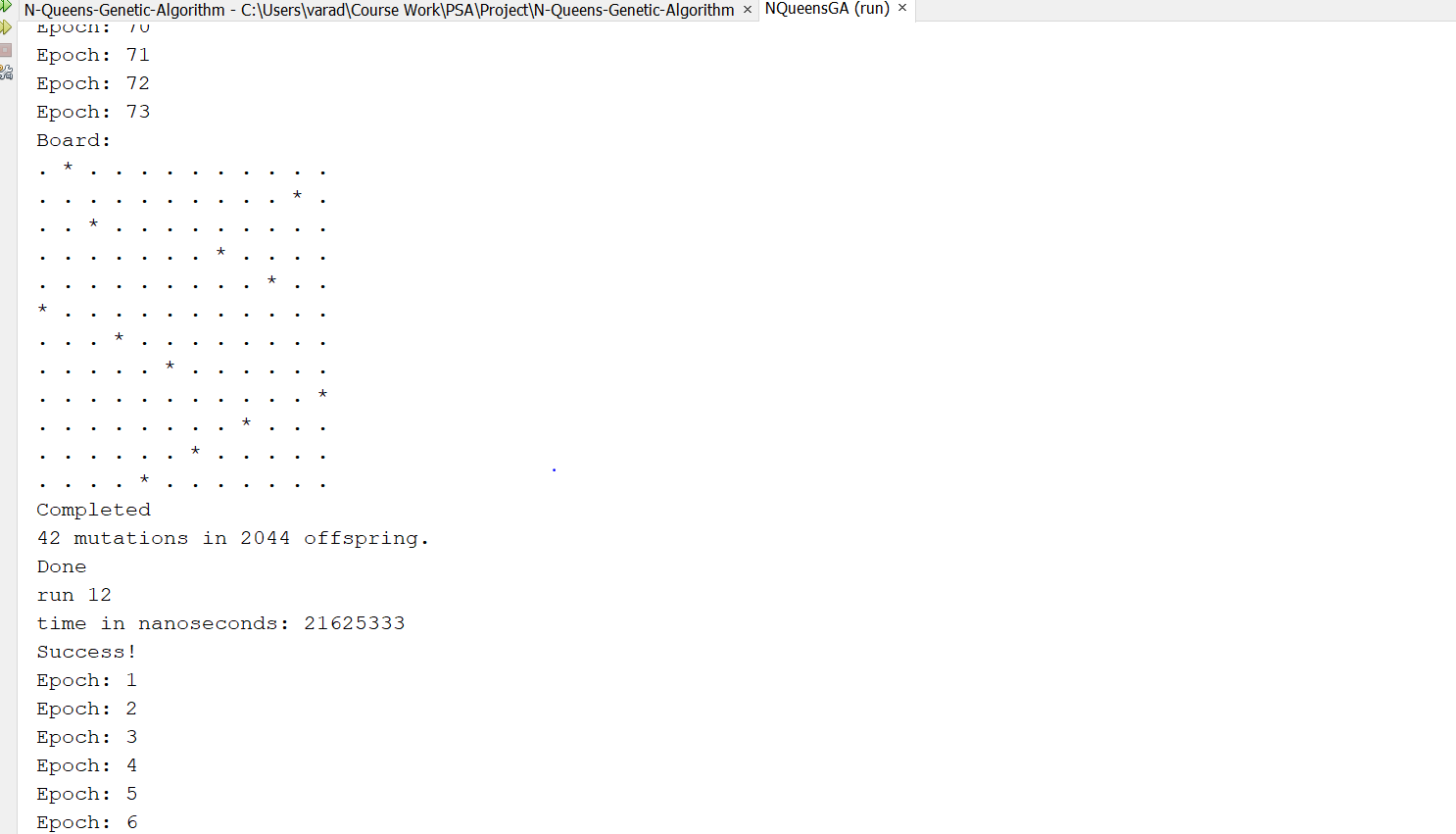
When applying genetic algorithms one aims to construct a model that, with some randomness, tries different individuals (possible solutions, differentiated by a list of values that defines its genetic information) to a problem , measure its **fitness** - which would mean to evaluate whether this possible solutions are perfect solutions or just good to some extent, and to measure this degree of ‘goodness’ - and to make the better solutions to breed and produce a new set of possible solutions with better fitness, and somehow closer to the perfect solution.

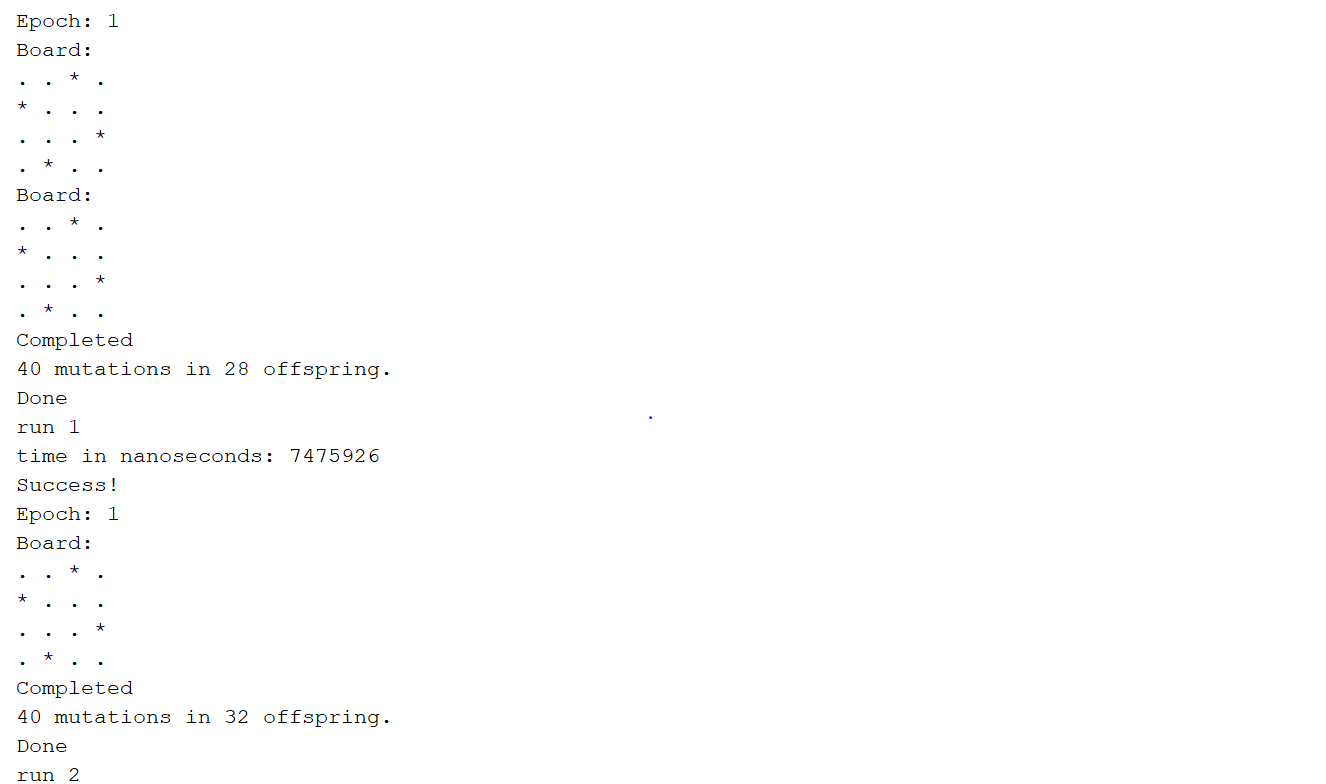
* **N-Queens Problem:**

In 1848, A German Chess player Max Bezzel composed the 8-Queens Problem which aims to place 8 Queens in the chess board in such a way that no two Queens can attack each other. In 1850 Franz Nauck gave the 1st solution to this problem and generalized the problem to N-Queen problem for N non- attacking Queens on an N x N Chessboard. Time complexity of an N-Queen problem is O(n!). Here, we are proposing a heuristic approach to obtain the best solutions for this problem. We are depicting all the arrangements of an N x N board as an N-tuple (c1, c2, c3… cN), where ci represents the position of the queen to be in i th column and c th row. Fig.1 shows an arrangement of 8 x 8 chessboard and its 8-tuple representation.

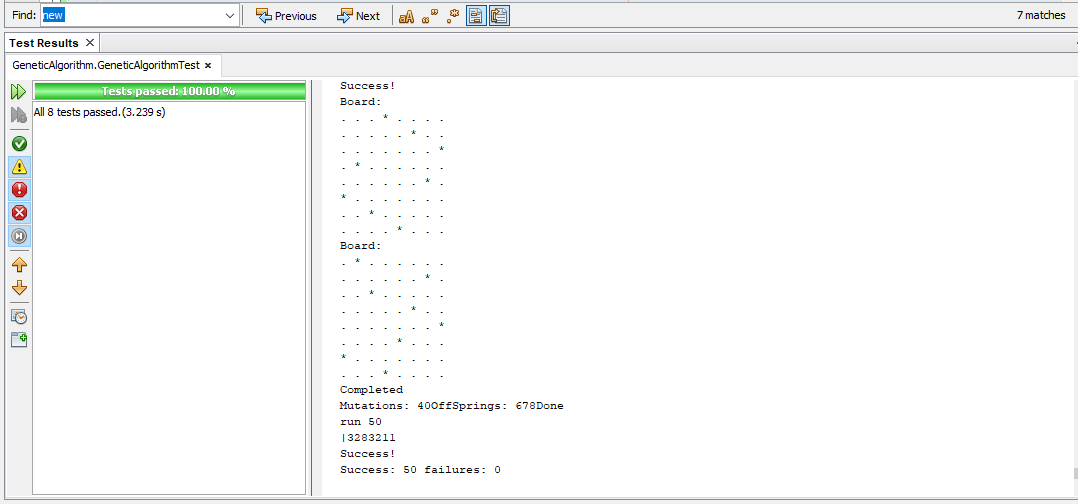


* **Output:**





* **UNIT TEST CASES Executed:**

******

* **Analysis:**

Analysis for Mutation vs OffSprings for N = 8 :

Output Results in Table for 50 Runs :

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr no.** | **Mutations** | **OffSprings** | **Total time in NanoSeconds** |
| 1 | 40 | 22 | 5384804 |
| 2 | 41 | 132 | 5117170 |
| 3 | 41 | 1098 | 38038493 |
| 4 | 40 | 208 | 3012049 |
| 5 | 40 | 66 | 993320 |
| 6 | 40 | 356 | 5202855 |
| 7 | 41 | 282 | 3308600 |
| 8 | 40 | 432 | 4523363 |
| 9 | 40 | 216 | 1275766 |
| 10 | 40 | 92 | 410798 |
| 11 | 41 | 1410 | 30293627 |
| 12 | 41 | 778 | 8207500 |
| 13 | 40 | 204 | 1446080 |
| 14 | 40 | 72 | 531745 |
| 15 | 41 | 850 | 9181426 |
| 16 | 40 | 426 | 1994399 |
| 17 | 41 | 1398 | 15798978 |
| 18 | 40 | 322 | 3509239 |
| 19 | 41 | 392 | 2103709 |
| 20 | 40 | 18 | 546556 |
| 21 | 40 | 250 | 1331832 |
| 22 | 42 | 1964 | 17407260 |
| 23 | 40 | 78 | 267636 |
| 24 | 40 | 340 | 978863 |
| 25 | 41 | 686 | 2652028 |
| 26 | 40 | 362 | 1144593 |
| 27 | 40 | 256 | 754599 |
| 28 | 40 | 888 | 4401005 |
| 29 | 40 | 242 | 1363215 |
| 30 | 40 | 66 | 392815 |
| 31 | 40 | 28 | 281740 |
| 32 | 40 | 78 | 417498 |
| 33 | 40 | 82 | 436892 |
| 34 | 40 | 162 | 789508 |
| 35 | 40 | 518 | 2411191 |
| 36 | 43 | 2536 | 26820356 |
| 37 | 40 | 116 | 444297 |
| 38 | 40 | 250 | 1122025 |
| 39 | 40 | 294 | 2600193 |
| 40 | 40 | 358 | 1413992 |
| 41 | 41 | 766 | 3630891 |
| 42 | 41 | 678 | 5278315 |
| 43 | 40 | 134 | 1083237 |
| 44 | 41 | 850 | 12489320 |
| 45 | 40 | 536 | 2595962 |
| 46 | 40 | 352 | 1463358 |
| 47 | 40 | 180 | 1322312 |
| 48 | 40 | 124 | 546908 |
| 49 | 41 | 426 | 2690816 |
| 50 | 41 | 982 | 13621572 |

Analysis for Success runs vs Failures runs for N in the range 4 to 20 :

|  |  |  |
| --- | --- | --- |
| **N** | **Success** | **Faliures** |
| 4 | 50 | 0 |
| 5 | 50 | 0 |
| 6 | 50 | 0 |
| 7 | 50 | 0 |
| 8 | 50 | 0 |
| 9 | 50 | 0 |
| 10 | 50 | 0 |
| 11 | 50 | 11 |
| 12 | 50 | 58 |
| 13 | 26 | 100 |
| 14 | 9 | 100 |
| 15 | 4 | 100 |
| 16 | 0 | 100 |
| 17 | 0 | 100 |
| 18 | 0 | 100 |
| 19 | 0 | 100 |
| 20 | 0 | 100 |

* **Complexity :**

For a board of size N by N there are N\*(N-1)/2 pairs of non attacking queens . For examples for N=8 , number of pairs of nonattacking queens are 28 [9].

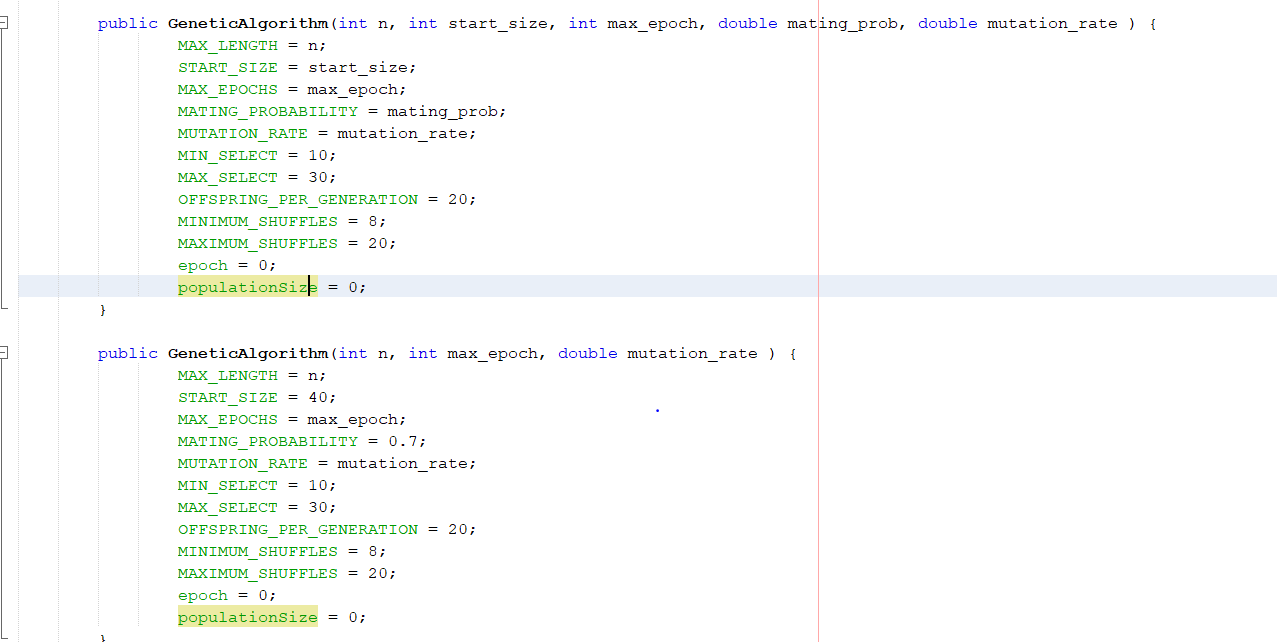
Time Complexity for Nqueens Problem using backtracking 🡪

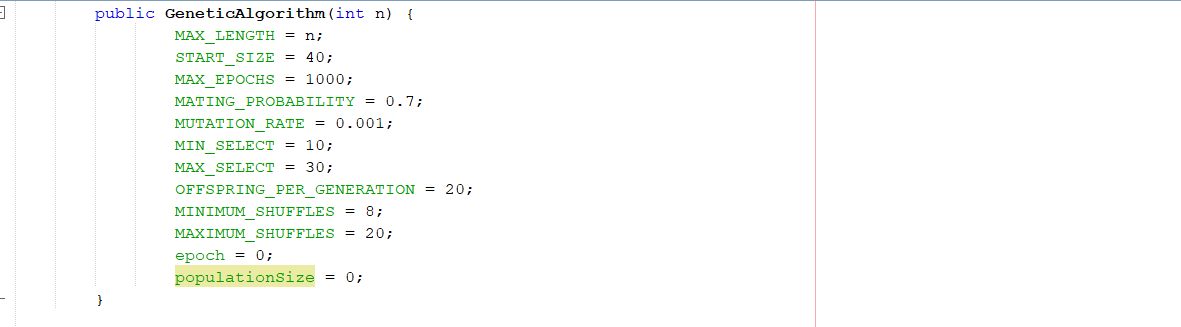
T(n) = n\*T(n-1) + O(n^2)

This shows that the problem is getting solved with every generation and we are iterating twice over the entire array.

If we solve this time complexity equation further we will get this equation translates to O(n!).

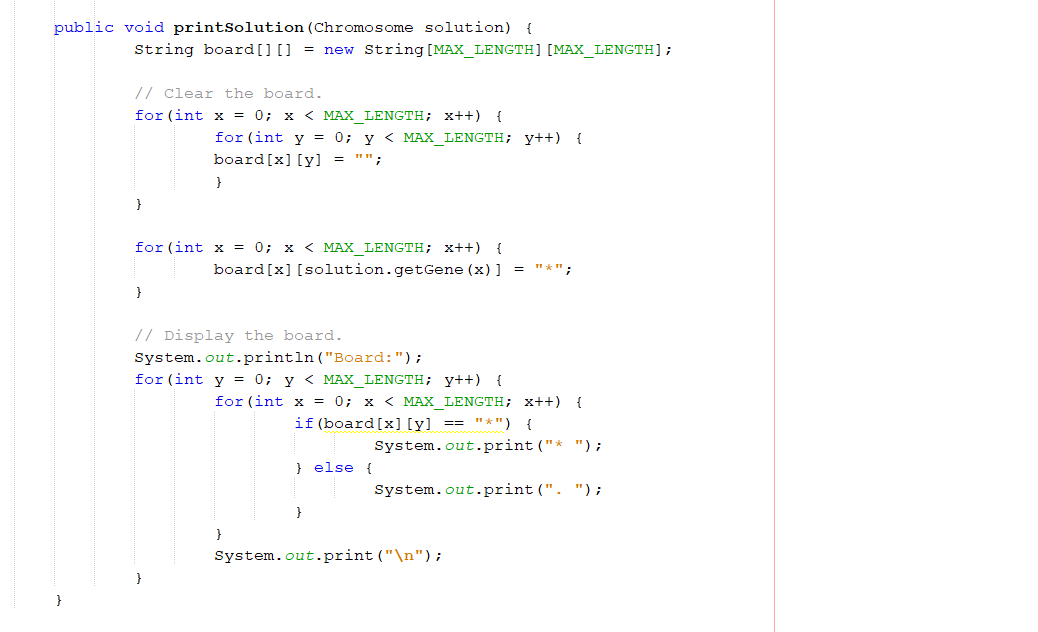
* **Code Snippets:**



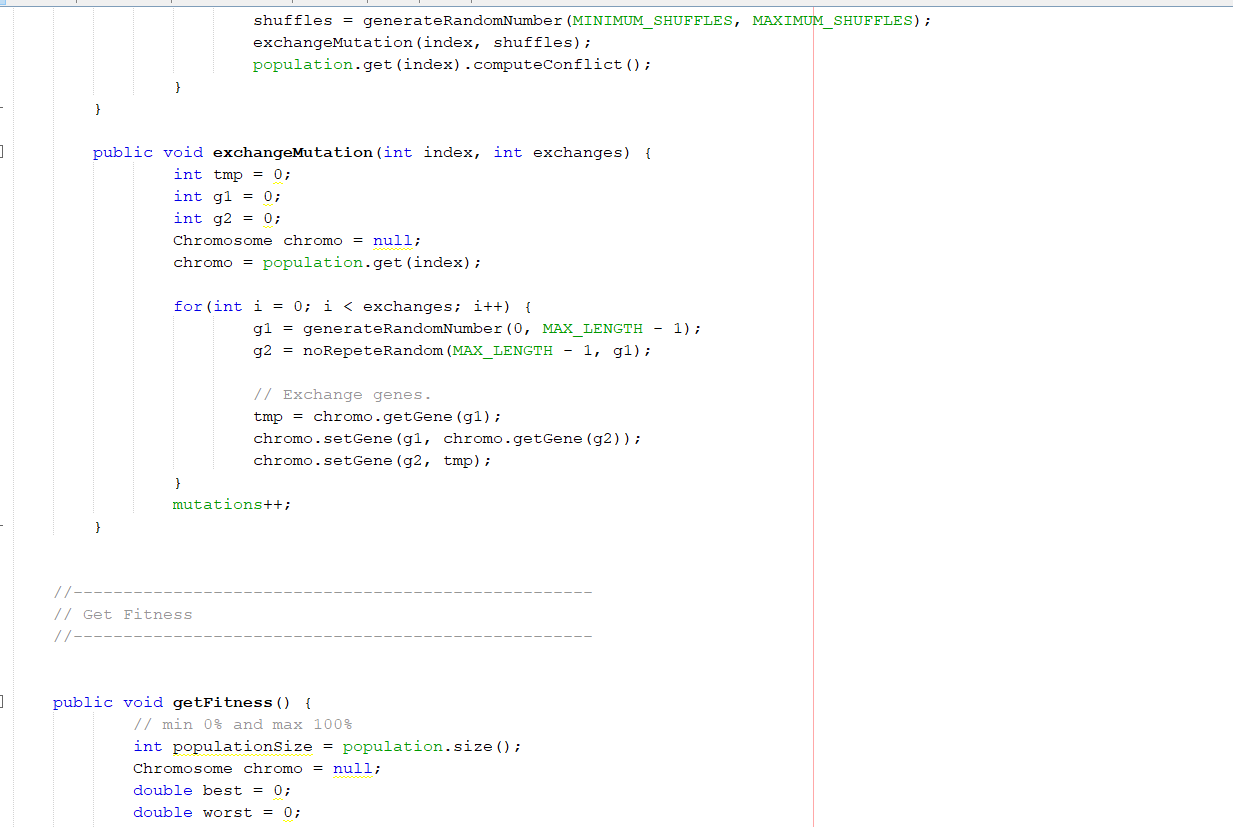












* **References:**

<https://gist.github.com/aliva/5355681>

<https://github.com/hajix/N-Queen>

<https://developers.google.com/optimization/cp/queens>

<https://www.geeksforgeeks.org/n-queen-problem-backtracking-3/>

<https://arxiv.org/pdf/1802.02006.pdf>

<https://kushalvyas.github.io/gen_8Q.html>

<https://www.kaggle.com/mrknoot/genetic-algorithms-solving-the-n-queens-problem>

<https://datajenius.com/articles/solving-n-queens-with-genetic-algorithms>

<https://ieeexplore.ieee.org/document/6802550>

<https://stackoverflow.com/questions/21059422/time-complexity-of-n-queen-using-backtracking>

Roulette Selection

<https://stackoverflow.com/questions/298301/roulette-wheel-selection-algorithm>

<https://stackoverflow.com/questions/177271/roulette-selection-in-genetic-algorithms>

Partially Mapped Crossover

<https://github.com/DEAP/deap/blob/master/deap/tools/crossover.py>

<https://stackoverflow.com/questions/52350699/how-to-perform-partially-mapped-crossover-operator-pmx>