Activity 1. Salesman

*Greedy1*

This algorithm calculates the path by finding the smallest path between the closest nodes, to find the path and calculate the smallest cost that way. This approach includes a boolean array to identify if a node is already included in the path or not. Due to using a two for loop approach, we can consider the complexity of the method as O(n^2).

*Greedy2*

Since I didn’t finish the method because I didn’t exactly know how to program it, I can’t determine the complexity of the program. If we consider the first test case, which for the implementation I did works (since no more than 1 element is created), the complexity there is O(n^2).

Activity 2. SalesmanTimes

|  |  |
| --- | --- |
| nTimes = 1000 | Greedy1 |
| N | Time |
| 1280 | 4 |
| 2560 | 14 |
| 5120 | 45 |
| 10240 | 150 |
| 20480 | 527 |

Since Greedy2 wasn’t finished, I didn’t record the times.

Greedy1 works fine, but it is not a really optimal solution because getting a node as a target, may lead to, such node as a source may have very high cost paths to all nodes but to other that we’ve already visited, thus increasing the cost a lot. To see if the times follow the complexity trend, we calculate the theorical times t2. For n2 = 2560, n1 = 1280 and t1 = 4, the obtained t2 is 14, and the theorical t2 Is 16. To the n2 = 10240, n1 = 5120, t1 = 45, the obtained t2 = 150, close to the theorical t2 180. Thus, we can conclude that the complexity is indeed O(n^2)