1. **Auto-Evaluation**

The program works correctly.

1. **Theoretical temporal complexity analysis**

The worst case in this problem would be the case where every service points have been implicated in cargo process.

Here is the algorithm’s code in java and the number of operations (bold numbers) in each code line:

int boxTransported = 0; **1**

int boxToTransport = boxsData.get(0); **2**

int boxRemaining = boxToTransport; **1**

int truckMaxCapacity = boxsData.get(1); **2**

int truckSpaceRemaining = truckMaxCapacity;  **1**

int i = 0; **1**

while (boxRemaining > 0) { **n + 1**

if (i != ordoredLists.get(1).size()) { **3n**

double element = ordoredLists.get(1).get(i); **3n**

if (element < truckSpaceRemaining) { **n**

double element2 = element + boxTransported;  **2n**

if (boxTransported < truckMaxCapacity && element2 < boxToTransport) { **2n**

boxTransported += element; **2n**

boxRemaining -= element; **2n**

truckSpaceRemaining -= element; **2n**

ordoredLists.get(1).remove(i); **2n**

ordoredLists.get(1).add(i, 0.0); **2n**

i++; **2n**

} else if (element2 >= boxToTransport) { **n**

element2 -= boxToTransport; **2n**

ordoredLists.get(1).remove(i); **2n**

ordoredLists.get(1).add(i, element2); **2n**

boxTransported = boxToTransport; **n**

boxRemaining = 0; **n**

} else {

boxRemaining = 0; **1**

}

} else if (element < truckMaxCapacity) { **n**

boxTransported += boxRemaining; **2n**

element -= boxRemaining; **2n**

ordoredLists.get(1).remove(i); **2n**

ordoredLists.get(1).add(i, element); **2n**

truckSpaceRemaining = 0; **n**

boxRemaining = 0; **1**

} else {

element -= boxToTransport; **2n**

ordoredLists.get(1).remove(i); **2n**

ordoredLists.get(1).add(i, element); **2n**

boxTransported = boxToTransport; **n**

boxRemaining = 0; **1**

}

} else {

boxRemaining = 0; **1**

}

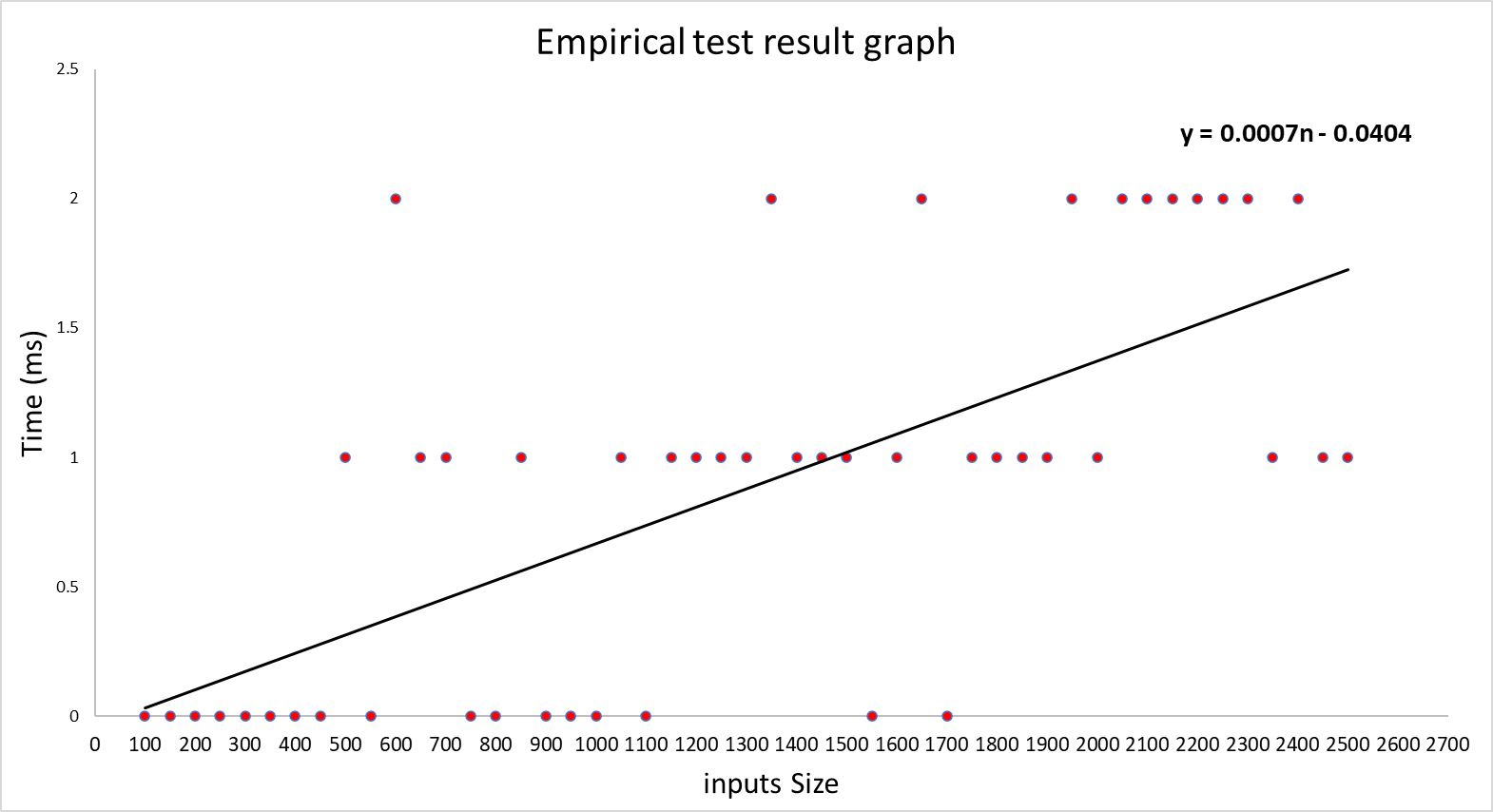
}

return ordoredLists;  **1**

In the worst case (in green), the polynomial function is: **22n + 9** so the algorithm is **O(n)** in the big-O notation.

1. **Empirical temporal complexity analysis**

Here’s the graphic demonstrating the algorithm running times for different sizes of the input:



**NB:** The data used to make this graphic may differ from the data available in the file “EmpiricTestData.txt” because the code for the test is executed every time the program runs so it gives different outputs. The other graphics may slightly differ from this one, but they are still linear.

The polynomial equation of this graphic is: **f(n) = 0.0007n – 0.0404**. So, based on this experience, we can also conclude that this algorithm is **O(n)** in the Big O notation.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Inputs size | Time (ms) |  | Inputs size | Time (ms) |
| 100 | 0 |  | 1100 | 0 |
| 150 | 0 | 1150 | 1 |
| 200 | 0 | 1200 | 1 |
| 250 | 0 | 1250 | 1 |
| 300 | 0 | 1300 | 1 |
| 350 | 0 | 1350 | 2 |
| 400 | 0 | 1400 | 1 |
| 450 | 0 | 1450 | 1 |
| 500 | 1 | 1500 | 1 |
| 550 | 0 | 1550 | 0 |
| 600 | 2 | 1600 | 1 |
| 650 | 1 | 1650 | 2 |
| 700 | 1 | 1700 | 0 |
| 750 | 0 | 1750 | 1 |
| 800 | 0 | 1800 | 1 |
| 850 | 1 | 1850 | 1 |
| 900 | 0 | 1900 | 1 |
| 950 | 0 | 1950 | 2 |
| 1000 | 0 | 2000 | 1 |
| 1050 | 1 | 2050 | 2 |
| 1100 | 0 | 2100 | 2 |
| 1150 | 1 | 2150 | 2 |
| 1200 | 1 | 2200 | 2 |
| 1250 | 1 | 2250 | 2 |
| 1300 | 1 | 2300 | 2 |
| 1350 | 2 | 2350 | 1 |
| 1400 | 1 | 2400 | 2 |
| 1450 | 1 | 2450 | 1 |
| 1500 | 1 | 2500 | 1 |
| 1550 | 0 |