ASSIGNMENT 1 :
IFT 2015

Name : Gbian Bio Samir
identifier : 20250793

10 juin 2023

1 Auto-Evaluation

The program works correctly.

2 Theoretical temporal complexity analysis

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

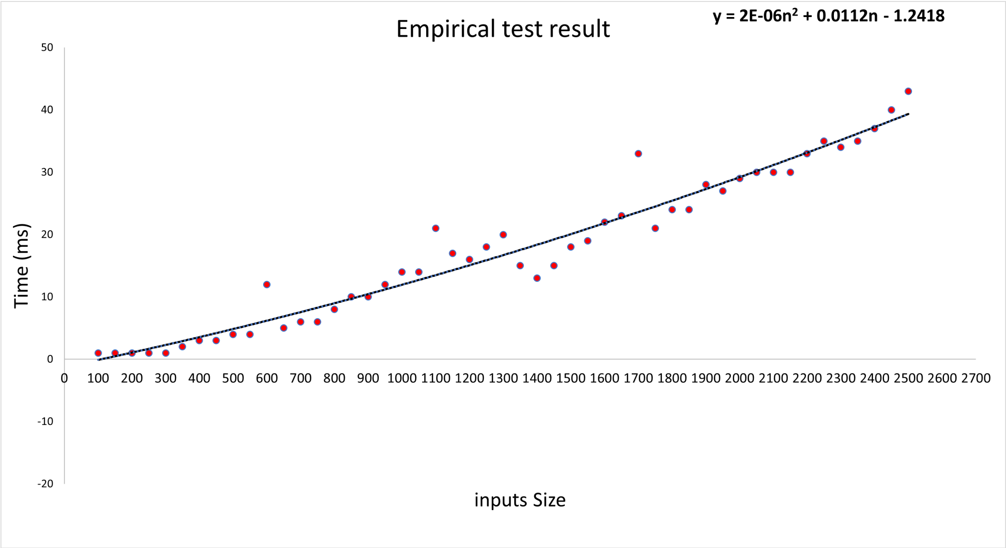
Here is the algorithm's code in java and the number of operations (bold numbers that are colored in red and blue) in each code line :

```
public static LinkedList<LinkedList<Double>> updateOrdoredLists(LinkedList<LinkedList<Double>>
ordoredLists, LinkedList<Integer> boxsData) {
    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
    double buildingVisited = 0 ; 1
    LinkedList<Double> buildingsVisited = new LinkedList¡¿() ; 1
    LinkedList<Double> orderedBoxs = ordoredLists.get(1) ;1

    while (boxRemaining > 0) { n + 1
        if (i != orderedBoxs.size()) { 2n
            double element = orderedBoxs.get(i) ; 2n
            if (element < truckSpaceRemaining) { n
                double element2 = element + boxTransported ; 2n
                if (boxTransported < truckMaxCapacity && element2 < boxToTransport) { 2n
                    boxTransported += element ; n
                    boxRemaining -= element ; n
                    truckSpaceRemaining -= element ; n
                    orderedBoxs.remove(i) ; n
                    orderedBoxs.add(i, 0.0) ; n
                    i++ ; n
                    buildingVisited++ ; n
                } else if (element2 >= boxToTransport) { 1
                    element2 -= boxToTransport ; 1
                    orderedBoxs.remove(i) ; 1
                    orderedBoxs.add(i, element2) ; 1
                    boxTransported = boxToTransport ; 1
                    boxRemaining = 0 ; 1
                    buildingVisited++ ; 1
                } else {
                    boxRemaining = 0 ; 1
                }
            } else if (element < truckMaxCapacity) { 1
                boxTransported += boxRemaining ; 1
                element -= boxRemaining ; 1
                orderedBoxs.remove(i) ; 1
                orderedBoxs.add(i, element) ; 1
                truckSpaceRemaining = 0 ; 1
                boxRemaining = 0 ; 1
                buildingVisited++ ; 1
            } else {
                element -= boxToTransport ; 1
                orderedBoxs.remove(i) ; 1
                orderedBoxs.add(i, element) ; 1
                boxTransported = boxToTransport ; 1
                boxRemaining = 0 ; 1
                buildingVisited++ ; 1
            }
        } else {
            boxRemaining = 0 ; 1
        }
return ordoredLists ; 1
}

public static LinkedList<LinkedList<Double>> ordoredLists(LinkedList<Double> distances, LinkedList<Double>
boxs, LinkedList<Double> buildings) {

    LinkedList<LinkedList<Double>> ordoredLists = new LinkedList<>() ; 1
    LinkedList<Double> orderedBuildings = new LinkedList<>() ; 1
    LinkedList<Double> ordoredBoxs = new LinkedList<>() ; 1
    LinkedList<Double> ordoredDistances = new LinkedList<>() ; 1
    ArrayList<Double> arrayDistances = new ArrayList<>(distances) ; 1
    int end = boxs.size() ; 2

    for(int i = 0 ; i < end ; ++i) { 2(n - 1)
        double smallestDistance = Collections.min(arrayDistances) ; 2n
        int smallestIndex = arrayDistances.indexOf(smallestDistance) ; 2n

        ordoredDistances.add(smallestDistance) ; n
        ordoredBoxs.add(boxs.remove(smallestIndex)) ; 2n

        int index = 2*smallestIndex ; 2n
        orderedBuildings.add(buildings.remove(index)) ; 2n
        orderedBuildings.add(buildings.remove(index)) ; 2n

        distances.remove(smallestIndex) ; n
        arrayDistances = new ArrayList<Double>(distances) ; n

    }
    ordoredLists.add(ordoredDistances) ;
    ordoredLists.add(ordoredBoxs) ;
    ordoredLists.add(orderedBuildings) ;
    return ordoredLists ;
}
```

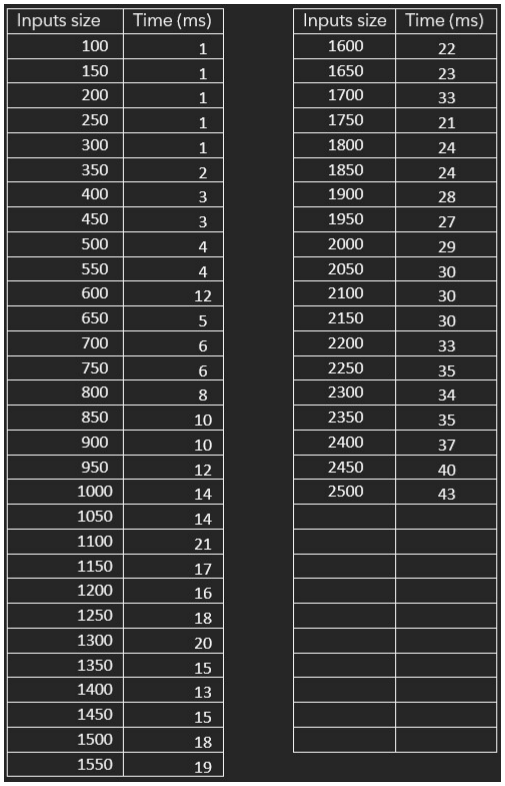
In the worst case (in red), the polynomial function is : **34n + 18** so the algorithm is **O(n)** in the big-O notation. Nevertheless, the complexity of the entier code is O(n^2) because of the methods used to read the .txt files. Only the methods used to make the comparison between the distances, order them, order the buildings and the boxes are O(n).

3 Empirical temporal complexity analysis

Here's the graphic demonstrating the algorithm running times for different sizes of the input the a part of the code :

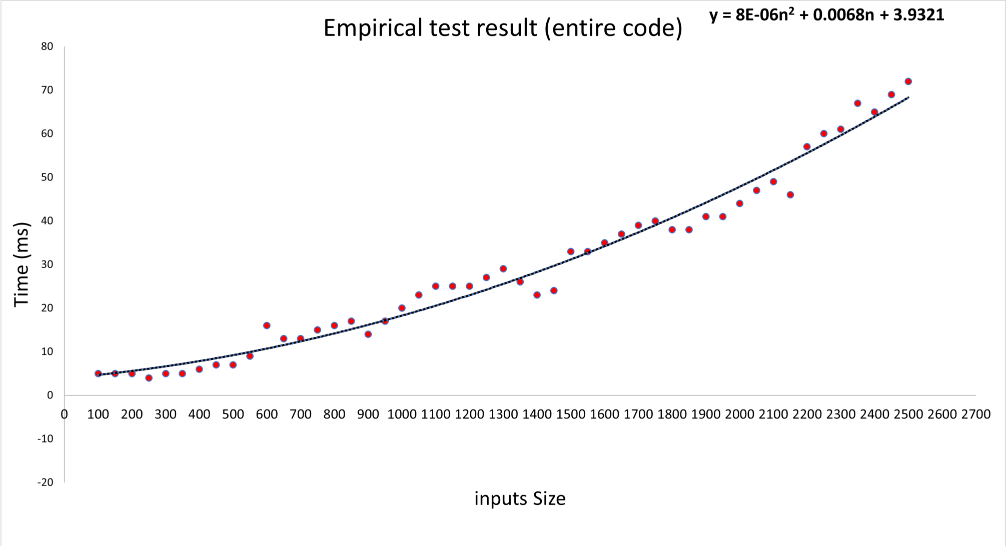


Here's the data used to make this graphic (they are also the result of the experience) :

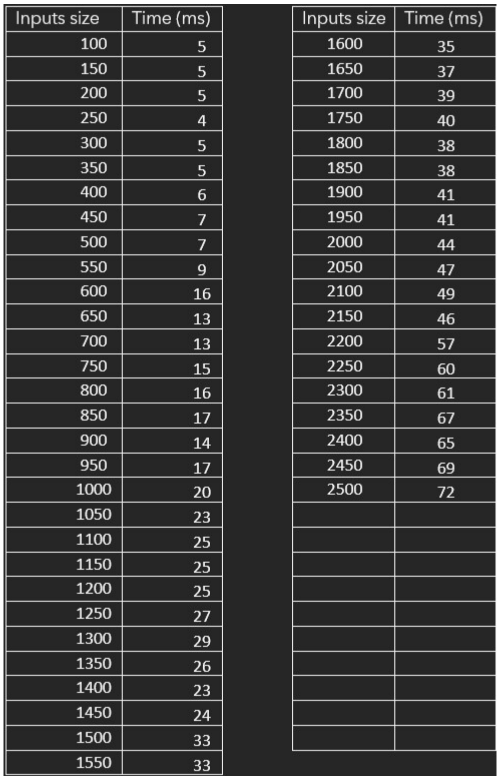


Inputs size	Time (ms)
100	1
150	1
200	1
250	1
300	1
350	2
400	3
450	3
500	4
550	4
600	12
650	5
700	6
750	6
800	8
850	10
900	10
950	12
1000	14
1050	14
1100	21
1150	17
1200	16
1250	18
1300	20
1350	15
1400	13
1450	15
1500	18
1550	19
1600	22
1650	23
1700	33
1750	21
1800	24
1850	24
1900	28
1950	27
2000	29
2050	30
2100	30
2150	30
2200	33
2250	35
2300	34
2350	35
2400	37
2450	40
2500	43

Here's the graphic demontrating the algorithm running times for the entire code :



Here's the data used to make this graphic :



Inputs size	Time (ms)
100	5
150	5
200	5
250	4
300	5
350	5
400	6
450	7
500	7
550	9
600	16
650	13
700	13
750	15
800	16
850	17
900	14
950	17
1000	20
1050	23
1100	25
1150	25
1200	25
1250	27
1300	29
1350	26
1400	23
1450	24
1500	33
1550	33
1600	35
1650	37
1700	39
1750	40
1800	38
1850	38
1900	41
1950	41
2000	44
2050	47
2100	49
2150	46
2200	57
2250	60
2300	61
2350	67
2400	65
2450	69
2500	72

Based on this experience, we can conclude that this algorithm is $\mathbf{O}(n^2)$ in the Big O notation. However, the slope is very small so, if we approximate $0.00002n^2$ to 0, the algorithm become O(n). The entire code is O(n^2).