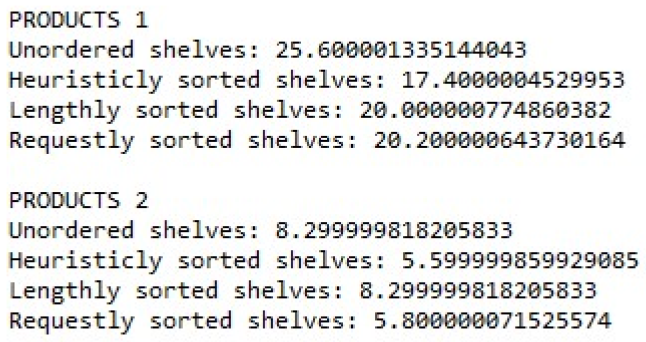
ALGORTIHMS USED:

- We used the average time algorithm to get the times to access every item of the shelve and later on we accumulated the times. The complexity of this time accessing technique is O(n^2) as we have a for loop inside a foreach.
- We have used a sorting algorithm similar to merge sort which is provided by java API. In my case I uses *Arrays.sort()* passing the array as parameter and the comparing criteria which I developed with lambda expression. As this method is similar to merge sort the complexity of the sorting is O(n logn);

ASKED QUESTIONS:

1. Done in the java files
2. As the heuristic is the best possible solution if we can obtain a greater value than it, that means our problem can be improved in some way.
3. As before
4. Done in the java files
5. The final algorithm has a complexity of O(n^2) for the average time and O(n logn) for the sorting in the best case. We can consider that this algorithm can not be improved a lot as we always had to do the same operations for getting the time, and the sort has the lowest possible complexity.

TIMES:



```
PRODUCTS 1
Unordered shelves: 25.600001335144043
Heuristicly sorted shelves: 17.4000004529953
Lengthly sorted shelves: 20.000000774860382
Requestly sorted shelves: 20.200000643730164

PRODUCTS 2
Unordered shelves: 8.299999818205833
Heuristicly sorted shelves: 5.599999859929085
Lengthly sorted shelves: 8.299999818205833
Requestly sorted shelves: 5.800000071525574
```

Sergio Faya Fernández

UO251005