ALG1 A: Reverse sorting

Name: Lucas Jacobs

Pcn: 490692

Class: APS4-RB01

Teachers: Simona Orzan

Date: 11-02-2023

Word count: 347

Contents

[The code 3](#_Toc127104419)

[The logic of the algorithm 3](#_Toc127104420)

[The time complexity of the algorithm 3](#_Toc127104421)

# The code

Text

Description automatically generated

# The logic of the algorithm

To begin we have three functions called: max, reverse, and reverse\_sort. The max and reverse function is explained in the code. Furthermore, we have the function reverse\_sort, this needs a bit more explanation.

What the function does is it will go threw the whole list and with every iteration, it will first find the highest number, then it will use the reverse function twice so that the highest number will come at the end of the list. This is maybe still a bit vague, so let’s give an example.

For instance, we have the list: S[4, 3, 2, 5, 1]

For the first

|  |  |  |  |
| --- | --- | --- | --- |
| iteration | Biggest number | After the first reverse | After the second reverse/After iteration |
| 1 | 5 | [5, 2, 3, 4, 1] | [1, 4, 3, 2, 5] |
| 2 | 4 | [4, 1, 3, 2, 5] | [2, 3, 1, 4, 5] |
| 3 | 3 | [3, 2, 1, 4, 5] | [1, 2, 3, 4, 5] |
| 4 | 2 | [2, 1, 3, 4, 5] | [1, 2, 3, 4, 5] |
| 5 | 1 | [1, 2, 3, 4, 5] | [1, 2, 3, 4, 5] |

Sorted list: [1, 2, 3, 4, 5]

So in the second iteration: The biggest number is now 4 because the second loop will look only at the length of the list minus the number of the iteration where in (So minus the number of the first loop).

# The time complexity of the algorithm

The function ‘reverse\_sort’ has an O(n²) time complexity.

To explain this: the first loop run k times. The loop inside of it runs i-k times. With each iteration k will increase by one therefore we get: n + (n-1) + (n-2) + … This can be rewritten as n(n-1)(n-2)/2. We can simplify this to O(n²). When n will get really big we can therefore neglect the constants, in this case, the 1 and 2 and the dividing by 2.