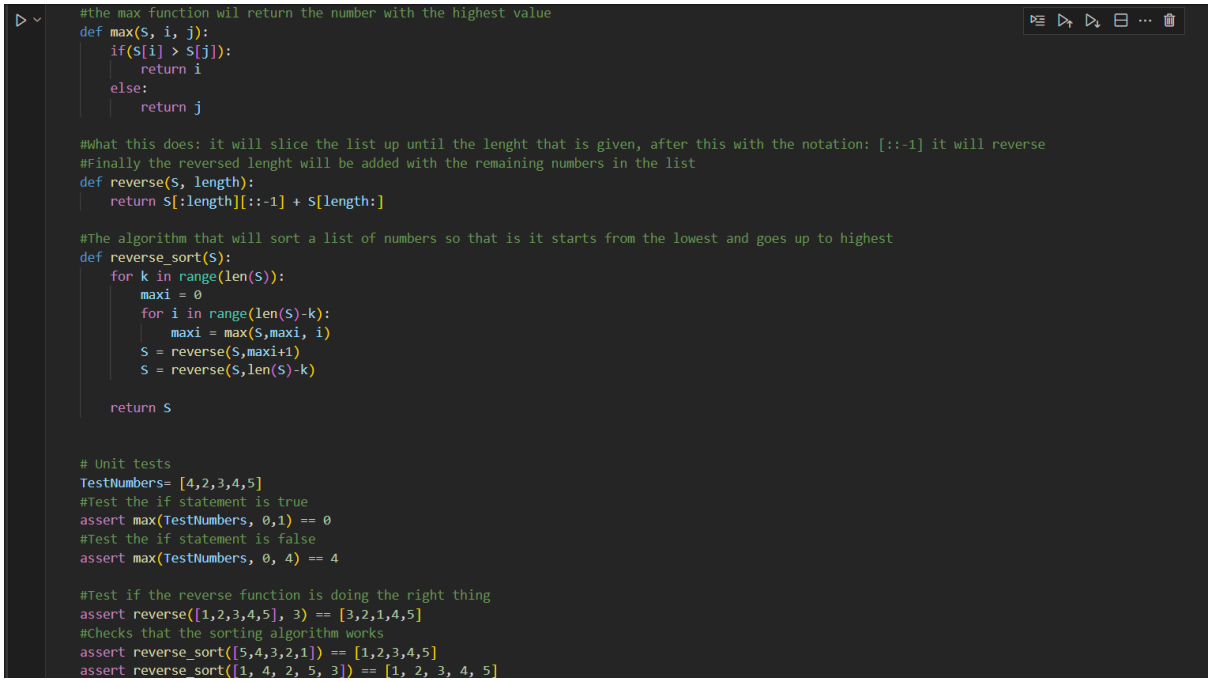
ALG1 A: Reverse sorting

Name: Lucas Jacobs
Pcn: 490692
Class: APS4-RB01
Teachers: Simona Orzan
Date: 11-02-2023
Word count: 347

Contents

The code



```python
#the max function wil return the number with the highest value
def max(S, i, j):
    if(S[i] > S[j]):
        return i
    else:
        return j

#What this does: it will slice the list up until the lenght that is given, after this with the notation: [::-1] it will reverse
#Finally the reversed lenght will be added with the remaining numbers in the list
def reverse(S, length):
    return S[:length][::-1] + S[length:]

#The algorithm that will sort a list of numbers so that is it starts from the lowest and goes up to highest
def reverse_sort(S):
    for k in range(len(S)):
        maxi = 0
        for i in range(len(S)-k):
            maxi = max(S,maxi, i)
        S = reverse(S,maxi+1)
        S = reverse(S,len(S)-k)

    return S


# Unit tests
TestNumbers= [4,2,3,4,5]
#Test the if statement is true
assert max(TestNumbers, 0,1) == 0
#Test the if statement is false
assert max(TestNumbers, 0, 4) == 4

#Test if the reverse function is doing the right thing
assert reverse([1,2,3,4,5], 3) == [3,2,1,4,5]
#Checks that the sorting algorithm works
assert reverse_sort([5,4,3,2,1]) == [1,2,3,4,5]
assert reverse_sort([1, 4, 2, 5, 3]) == [1, 2, 3, 4, 5]
```

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 $O(n^2)$ 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 $O(n^2)$. When n will get really big we can therefore neglect the constants, in this case, the 1 and 2 and the dividing by 2.