Assignment n°1: Reverse Algorithm

Every code part is written in Python.

Question a

I developed the `reverse` function using a for loop going from 0 to the index in the list of the value to which to reverse.

I didn't catch the 2 following cases:

- The index passed as parameter is lower or equal to 0.
- The index is higher than the list length.

I also developed the `max` function returning the index of the maximum value between the two parameters.

```python
def reverse(lst: list, idx: int) -> None:
    for i in range(idx):
        if i > idx:
            print(f'List after Reverse: {lst}')
            return
        lst[idx], lst[i] = lst[i], lst[idx]
        idx -= 1
    print(f'List after Reverse: {lst}')


def max(lst: list, idx1: int, idx2: int) -> int:
    return idx1 if lst[idx1] > lst[idx2] else idx2
```

Question b

The algorithm I choose was a 3 steps:

- Find the index of the minimum value
- Reverse all values to this minimum index
- Reverse all values to an iterative i value where i increments at each loop

If the minimum value index is equal to our position on the list, we don't need to sort the values from 0 to this index, because they are already sorted.

If the minimum value index isn't equal to our position on the list, we need to reverse our list because it means the list isn't sorted for the moment.

In that case, we are making sure the lowest values are at the end of our list. So we need to sort on the portion of the list that isn't sorted from 0 to the minimum value. Then, we need to reverse it on the size of the list we're looking to make sure to comes at the end of our list.

As an example:

- We got the list [5, 4, 2, 1, 3].

At the first iteration, we will be at the end of our list. Value i will be equal to the list length. Minimum value will be at index 3 (value equals to 1).

We need to reverse the list to the third value and then make it on the portion of the list that is equals to i in order to have the lowest value of the part of the list to be sorted at the end of the list.

```python
def reverse(lst: list, idx: int) -> None:
    for i in range(idx):
        if i > idx:
            print(f'List after Reverse: {lst}')
            return
        lst[idx], lst[i] = lst[i], lst[idx]
        idx -= 1
    print(f'List after Reverse: {lst}')


def max(lst: list, idx1: int, idx2: int) -> int:
    return idx1 if lst[idx1] > lst[idx2] else idx2


def reverse_sort(lst: list) -> None:
    for i in range(len(lst) - 1, 0, -1):
        min_value = i
        for j in range(0, i):
            if max(lst, j, min_value) == min_value:
                min_value = j
        if i == min_value:
            continue
        reverse(lst, min_value)
        reverse(lst, i)
    print(f'Sorted List: {lst}')
```

Question c

The unit tests are made using Pytest. This unit tests are made for `reverse`, `max` and `reverse_sort` functions (more like functionnal testing than unit testing).

I also created a `is_reverse_sorted` function checking if the list is reverse sorted, returning true in that case, false otherwise.

```python
from sources.reverse import reverse, max, reverse_sort
from copy import deepcopy


def is_reverse_sorted(lst: list) -> bool:
    for i in range(len(lst) - 1):
        if lst[i] < lst[i + 1]:
            return False
    return True


class TestReverseAlgorithm:
    long_long_initial_list = [
        49, 47, 46, 45, 32, 33, 31, 30, 34, 38,
        37, 36, 35, 1, 2, 8, 9, 10, 18, 19,
        21, 22, 28, 26, 27, 25, 23, 24, 20, 4,
        5, 3, 50, 6, 7, 11, 12, 13, 14, 15,
        16, 17, 29, 39, 40, 44, 43, 42, 41, 40
    ]
    longer_initial_list = [16, 17, 18, 19, 20, 1, 2, 3, 4, 5, 15, 14, 13, 12, 11, 10, 6, 7,
    initial_list = [2, 3, 4, 5, 1]

    def test_NormalReverse(self):
        lst = deepcopy(self.initial_list)
        reverse(lst, 3)
        assert lst == [5, 4, 3, 2, 1]

    def test_LowerThanSize(self):
        lst = deepcopy(self.initial_list)
        reverse(lst, 1)
        assert lst == [3, 2, 4, 5, 1]

    def test_MaxIndex(self):
        lst = deepcopy(self.initial_list)
        assert max(lst, 0, 1) == 1

    def test_MinIndex(self):
        lst = deepcopy(self.initial_list)
        reverse(lst, 1)
        assert max(lst, 0, 1) == 0

    def test_EmptyList(self):
        lst = []
        reverse_sort(lst)
        assert lst == []

    def test_OneItemList(self):
        lst = [1]
        reverse_sort(lst)
        assert lst == [1]
    def test_BasicTest(self):
        lst = deepcopy(self.initial_list)
        reverse_sort(lst)
        assert is_reverse_sorted(lst)

    def test_LongListTest(self):
        lst = deepcopy(self.longer_initial_list)
        reverse_sort(lst)
        assert is_reverse_sorted(lst)

    def test_LongLongListTest(self):
        lst = deepcopy(self.long_long_initial_list)
        reverse_sort(lst)
        assert is_reverse_sorted(lst)
```

Question d

The time complexity of the algorithm I've done, considering the reverse operation as a complexity of O(1), is O(2n^2).

I used 2 loops, one nested in another. Therefore, we can already have the O(n^2) complexity.

I also used twice the `reverse` instruction. Therefore, our complexity will be O(2n^2) simplified as O(n^2).

Question e

My `reverse` function has a loop. From there, we can consider our `reverse` function complexity to be O(n).

As we already have our O(n^2) complexity for the `reverse_sort` function, we can assume our complexity will be O(n^2 + 2n) also simplified as O(n^2).

Therefore, our reverse sorting algorithm might be O(n^2).