

Curso > Week 6... > 12. Sea... > Exercis...

Audit Access Expires 5 de ago de 2020

You lose all access to this course, including your progress, on 5 de ago de 2020.

Exercise 6

Finger Exercises due Aug 5, 2020 20:30 -03

Exercise 6

2/4 points (graded)

ESTIMATED TIME TO COMPLETE: 8 minutes

Here is another version of a sorting function:

```
def mySort(L):
""" L, list with unique elements """
clear = False
while not clear:
    clear = True
    for j in range(1, len(L)):
        if L[j-1] > L[j]:
            clear = False
            temp = L[j]
            L[j] = L[j-1]
            L[j-1] = temp
```

Compare this to:

```
def newSort(L):
""" L, list with unique elements """
for i in range(len(L) - 1):
    j=i+1
    while j < len(L):
        if L[i] > L[j]:
            temp = L[i]
            L[i] = L[j]
            L[j] = temp
```

1. Do these two functions result in the same sorted lists?



Explanation:

Yes, both mySort and newSort correctly sort a list.

mySort :

A list is sorted if every pair of successive elements in a list are in the correct order. mySort implements this idea more directly than in other sorting algorithms we have seen. The basic idea is that every time it finds two successive elements in the wrong order, it will swap them. Because all lists can be sorted, it will eventually run out of things that are in the wrong order. At this point the list is sorted, and the algorithm terminates.

Another way of thinking about mySort is that in each iteration, if an element e is bigger than the one after it, e moves down one location. Then, e is checked against the next element, and so on, until the algorithm finds an element bigger than e. So, in the first pass, the biggest element drops to the bottom of the list. Then, in the second pass, the second biggest drops to the second to last position in the list, and so on for the remaining iterations. In each pass through the list, the next biggest element drops to its proper location, so that after n iterations, the list is sorted. This algorithm is typically known as 'bubble sort' as elements bubble (up or down) one element at a time.

newSort :

newSort is basically a slight variant of Selection Sort (see previous problem). In each iteration, newSort tries to find the smallest element in the unsorted part of the list and appends it to the sorted part of the list. newSort maintains that the element at the i th position is the smallest element between the i th and j th positions. So, when j reaches the end of the list, the j th position must have been the smallest element in the unsorted portion (from position i to the end) of the list.

2. Do these two functions execute the same number of assignments of values into entries of the lists?

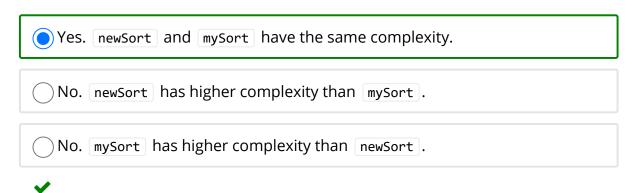
○ Yes. They execute the same number of assignments. ✔		
No. newSort may use more - but never fewer - inserts than mySort.		
No. mySort may use more - but never fewer - inserts than newSort.		
No. Either function may use more inserts than the other.		

Explanation:

This is pretty complicated to prove, so don't worry if this guestion was hard for you! Here's a sketch of why both mySort and newSort execute the same number of assignments:

newSort is, loosely speaking, performing mySort in the opposite direction, moving up the next smallest element to the beginning of the list. However, instead of swapping the successive elements, it instead swaps with the eventual position the smallest element will have to end up in. The number of swaps it needs ends up being the same as that of mySort because moving in either direction will encounter the same number of inconsistent pairwise elements.

3. Is the worst-case order of growth of these functions the same?



Explanation:

Yes. mySort is $O(n^2)$. In each iteration, mySort checks n-1 successive pairwise elements, and also moves the next biggest element to the bottom of the list (see explanation of how mySort works under the first question of this problem). So, after at most n iterations, it will have moved the n biggest elements to their correct locations in which case it has sorted the list! So the worst case time complexity for mysort is $O(n^2)$.

In <code>newSort</code> , <code>i</code> iterates over each element of the list, and <code>j</code> checks between 1
and up to $[n-i]$ elements. That's $[n]$ iterations for $[i]$, and for each $[i]$, we are
looking for the smallest element by checking about <code>n/2</code> elements on average.
That's kind of like $n * n/2$ checks, which is a complexity of $O(n^2)$.

4. Do these two functions examine the same number of entries in the list?

Yes. newSort and mySort examine the same number of entries.		
No. newSort examines more entries than mySort.		
No. mySort examines more entries than newSort.		
No. mySort and newSort examine different numbers of entries, but one cannot always say which function will examine the most entries. ✔		

Explanation:

For L = [1, 2, 3] mySort performs 2 comparisons and newSort performs 3 comparisons. For L = [3, 2, 1] mySort performs 6 comparisons and newSort performs 3 comparisons.

Enviar

1 Answers are displayed within the problem

Exercise 6

Ocultar discussão

Topic: Lecture 12 / Exercise 6

Add a Post

Show all posts	•	por atividade recente 🗸		
There are no posts in this topic yet.				
×				
•				

© All Rights Reserved