Exercise 3

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

Exercise 3

7/9 points (graded)

ESTIMATED TIME TO COMPLETE: 12 minutes

For the following programs, fill in the best-case and the worst-case number of steps it will take to run each program.

For these questions, you'll be asked to write a mathematical expression. Use +, -, / signs to indicate addition, subtraction, and division. Explicitly indicate multiplication with a * (ie say "6*n" rather than "6n"). Indicate exponentiation with a caret (^) (ie "n^4" for n^4). Indicate base-2 logarithms with the word log2 followed by parenthesis (ie "log2(n)").

1. Program 1:

```
def program1(L):
    multiples = []
    for x in L:
        for y in L:
            multiples.append(x*y)
    return multiples
```

What is the number of steps it will take to run Program 1 in the best case? Express your answer in terms of *n*, the number of elements in the list `L`. You can assume list appending takes 1 step.

2

✔ **Answer:** 2

2

What is the number of steps it will take to run Program 1 in the worst case? Express your answer in terms of *n*, the number of elements in the list `L`. You can assume list appending takes 1 step.

3 * n ^ 2 + n + 2

Answer: 3*n^2 + n + 2

$3 \cdot n^2 + n + 2$

Explanation:
In the best case scenario, `L` is an empty list. So we execute only the first assignment statement, then the return statement. Thus in the best case we execute 2 steps. Note that since the list is empty, no assignments are performed in the `for x in L` line.
In the worst case scenario, `L` is a long list. In this case we go through the loop `for x in L` *n* times. Every time through this loop, we execute an assignment of a value to `x`, and then the inner loop `for y in L`. The assignment takes 1 step on each iteration; how many steps does the inner loop take on each iteration?
The inner loop has three operations (assignment of a value to y, x*y, and list appending). So the inner loop executes 3*n times on *each iteration* of the outer loop. Thus the nested loop structure is executed n * (3*n + 1) = 3*n**2 + n times!
Adding in two steps (for the first assignment statement, and the return statement) we see that in the worst case, this program executes 3*n**2 + n + 2 steps.

2. Program 2:

```
def program2(L):
    squares = []
    for x in L:
        for y in L:
            if x == y:
                squares.append(x*y)
    return squares
```

What is the number of steps it will take to run Program 2 in the best case? Express your answer in terms of *n*, the number of elements in the list `L`.

2

✔ **Answer:** 2

2

What is the number of steps it will take to run Program 2 in the worst case? Express your answer in terms of *n*, the number of elements in the list `L`.

4*n^2 + n + 2

Answer: 4*n^2 + n + 2

$4 \cdot n^2 + n + 2$

Explanation:
In the best case scenario, `L` is an empty list. So we execute only the first assignment statement, then the return statement. Thus in the best case we execute 2 steps. Note that since the list is empty, no assignments are performed in the `for x in L` line.
In the worst case scenario, `L` is a long list of one repeated number (ie [2, 2, 2, 2, ...]. In this case we go through the loop `for x in L` n times. Every time through this loop, we perform one assigment of a value to the variable `x`, then we execute the inner loop `for y in L` n times.
The inner loop performs one assigment of a value to the variable `y`. It then has one operation that is checked every time (`if x == y`). Since the WORST case is when the list is composed of identical elements, this check is always true - so the third and fourth operations (x*y, and list appending) are always performed. So the inner loop executes 4*n times on *each iteration* of the outer loop. Thus the nested loop structure is executed n * (4*n + 1) = 4*n**2 + n times!
Adding in two steps (for the first assignment statement, and the return statement) we see that in the worst case, this program executes 4*n**2 + n + 2 steps.

3. Program 3:

```
def program3(L1, L2):
    intersection = []
    for elt in L1:
        if elt in L2:
            intersection.append(elt)
    return intersection
```

What is the number of steps it will take to run Program 3 in the best case? Express your answer in terms of *n*, the number of elements in the list `L1` (assume `len(L1) == len(L2)`).

2

✔ **Answer:** 2

2

What is the number of steps it will take to run Program 3 in the worst case? Express your answer in terms of *n*, the number of elements in the list `L1` (assume `len(L1) == len(L2)`).

2+n^2+2*n

✔ **Answer:** n^2 + 2*n + 2

$2 + n^2 + 2 \cdot n$

Explanation:
In the best case scenario, `L1` is an empty list. So we execute only the first assignment statement, then the return statement. Thus in the best case we execute 2 steps.
In the worst case scenario, every element of `L1` is the same as the very last element of `L2`. In this case we go through the loop `for elt in L1` n times. Every time through this loop, we perform one assigment of a value to the variable `elt`, then we execute the check `if elt in L2`.
How long does it take to execute this check? Well in the *worst case*, `elt` is the very LAST element of `L2`. In order to check `if elt in L2`, Python iterates over the elements of `L2` until it either finds the one you're looking for, or `L2` runs out of elements. Thus in the worst case, the check `if elt in L2` takes n steps. After this, we perform one append operation. So the conditional plus the append takes n + 1 steps on *each iteration* of the loop. Thus the `for` loop takes n * (n + 2) = n**2 + 2*n steps!
Adding in two steps (for the first assignment statement, and the return statement) we see that in the worst case, this program executes n**2 + 2*n + 2 steps.

4. In the last video, Professor Grimson introduced the idea of "asymptotic complexity", which means we describe running time in terms of number of basic steps. We've described the best- and worst-case running times in terms number of basic steps for the three programs above. Now, we'd like you to give the complexity order (ie, "Big O" running time) of each of the above programs.

   Recall that "Big O" notation gives an upper bound on asymptotic growth of a function. So, should you use the best-case or the worst-case running times for each program? Review the video again if you're unsure of what to put for the following boxes.

Note: Your answer should be expressed with a capital letter O, then a mathematical term similar to one described in the introduction to this problem - for example, O(n^5).

1. What is the complexity order of Program 1?

O(n^2) ✔

Answer: O(n^2)

2. What is the complexity order of Program 2?

O(n^2) ✔

Answer: O(n^2)

3. What is the complexity order of Program 3?

O(n^2) ✔

Answer: O(n^2)

Explanation:
Remember the following rules when determining the complexity order of a function:

1. If running time is a sum of multiple terms, keep one with the largest growth rate (so n**3 + 100n**2 + 500,000 is O(n**3)).

2. If the remaining term is a product (eg 3n**2), drop any multiplicative constants (so 3n**2 is O(n**2)).

It's also good to note that if you have a function that takes a constant number of steps - regardless of the size of the input - the function is O(1), even if it takes 3,000,000 steps every time! This is because the function does not take any additional time as the input grows large.
Finally, pay attention to the fact that Programs 1, 2, and 3 were all O(n**2). This is important! Generally, a nested loop structure has O(n**2) complexity. This is not the best, as we'll discover in the next lectures in this sequence.

Reminder: You do not lose points for trying a problem multiple times, nor do you lose points if you hit "Show Answer". If this problem has you stumped after you've tried it a few times, feel free to reveal the solution.

Click the "Reset" button to clear your answers.

Enviar

Exercise 3

Ocultar discussão

Topic: Lecture 11 / Exercise 3

Add a Post

Show all posts por atividade recente

- Unaccounted Extra Step — 2
  There seems to be an extra step for every question in this exercise that I can't seem to figure its s...
- ? Spoiler: Big O makes sense, the counting steps logic is eluding me — 2
  So far, I'm finding it easy to identify the biggest exponent and use Big O notation to call it out. Th...
- Suggestions welcome! — 5
  Man, this is hard. Anyone have tips or resources on this topic or these questions?