## L8 PROBLEM 1 (3/3 points)

For these questions, you'll be asked to give the big-O upper bound runtime for some Python functions. In your answer, please omit the "O( )" portion of the answer and simply 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)").

What this all means is if an answer is, for example, $O\left(n^4\right)$, please simply write "n^4" in the box.

1. What is the big-O upper bound runtime for the function `look_for_things`, where *n* is defined as the number of elements in `myList`?

```
NUMBER = 3
def look_for_things(myList):
    """Looks at all elements"""
    for n in myList:
        if n == NUMBER:
            return True
    return False
```

n

$n$

**Answer:** n

2. What is the big-O upper bound runtime for the function `look_for_other_things`, where *n* is defined as the number of elements in `myList`?

```
NUMBER = 3
def look_for_other_things(myList):
    """Looks at all pairs of elements"""
    for n in myList:
        for m in myList:
            if n - m == NUMBER or m - n == NUMBER:
                return True
    return False
```

n^2

$n^2$

**Answer:** n^2

3. What is the big-O upper bound runtime for the function `look_for_all_the_things`, where *n* is defined as the number of elements in `myList`?

You do not need to account for the runtime of `get_all_subsets`; the code is provided so you can see how many subsets it generates, as that **will** be a factor in your answer.

```
def get_all_subsets(some_list):
    """Returns all subsets of size 0 - len(some_list) for some_list"""
    if len(some_list) == 0:
        # If the list is empty, return the empty list
        return [[]]
    subsets = []
    first_elt = some_list[0]
    rest_list = some_list[1:]
    # Strategy: Get all the subsets of rest_list; for each
    # of those subsets, a full subset list will contain both
    # the original subset as well as a version of the subset
    # that contains first_elt
    for partial_subset in get_all_subsets(rest_list):
        subsets.append(partial_subset)
        next_subset = partial_subset[:] + [first_elt]
        subsets.append(next_subset)
    return subsets

NUMBER = 3
def look_for_all_the_things(myList):
    """Looks at all subsets of this list"""
    # Make subsets
    all_subsets = get_all_subsets(myList)
    for subset in all_subsets:
        if sum(subset) == NUMBER:
            return True
    return False
```

2^n

$2^n$

**Answer:** 2^n

**EXPLANATION:**

The point of this exercise was to get you thinking about the complexity of functions, specifically the complexity of different ways to enumerate items. Keep these complexities in mind as you continue throughout this lecture sequence. It might sound great to always get the very best solution by enumerating all possible choices - but the downside to this approach is the terribly high complexity!

$O(2^n)$ is the complexity for the final question because we are enumerating all possible subsets of a set, known as the "power set" of a set. We will talk about power sets more in the next videos.

Check    Hide Answer

Show Discussion    New Post