CSC110 Lecture 25: Worst-Case Running Time Analysis

Hisbaan Noorani

November 18, 2020

Contents

1 Exercise 1: Worst-case running time analysis practice

Consider the following function, which has an early return:

```python
def are_disjoint_lists(nums1: List[int], nums2: List[int]) -> bool:
    """Return whether nums1 and nums2 are disjoint sets of numbers."""
    for x in nums1:     # n steps
        if x in nums2:  # n steps
            return False

    return True
```

Note: For your analysis in this exercise, assume both input lists have the same length n.

1. Find a tight upper bound (Big-O) on the worst-case running time of `are_disjoint_lists`. By "tight" we mean it should be possible to prove the same lower bound (Omega), but we're not asking you to do it until the next question.

   Use phrases like *at most* to indicate inequalities in your analysis.

   Let $n = |\text{nums1}| = |\text{nums2}|$

   One iteration takes *at most* $n + 1$ steps when x is the last element in the list

   The for loop iterates n times.

   $WC_f = n \cdot (n + 1) = n^2 + n \in \wr(n^2)$

2. Prove a matching lower bound on the worst-case running time of `are_disjoint_lists`. Remember that this means finding an input family whose asymptotic running time is the same as the upper bound you found in Question 1.

   Let $n = |\text{nums1}| = |\text{nums2}|$

   Let nums1 and nums2 be lists that are disjoint.

   $\forall x \in \text{nums1}, x \notin \text{nums2}$

   Then, a single iteration of the for loop takes at least n steps.

   The loop iterates n times.

   Therefore $WC_f \in \Omega(n^2)$

3. Using Questions 1 and 2, conclude a tight Theta bound on the worst-case running time of `are_disjoint_lists`.

   Since we've shown that $WC_f \in \wr(n^2)$ and $WC_f \in \Omega(n^2)$, we can say that $WC_f \in \Theta(n^2)$

2 Exercise 2: Lists vs. sets

Now consider the following function, which is the same as the previous one, but operates on sets instead of lists:

```python
def are_disjoint_sets(nums1: Set[int], nums2: Set[int]) -> bool:
    """Return whether nums1 and nums2 are disjoint sets of numbers."""
    for x in nums1:     # n steps
        if x in nums2:  # 1 step
            return False

    return True
```

Note: all parts of this question explores a few variations of the analysis you did in Exercise 1. To save time, don't rewrite your full analysis. Just describe the parts that would change, and the final bound that you get.

1. Analyse the worst-case running time of `are_disjoint_sets`, still assuming that the two input sets have the same length.

   Big-O

   One iteration takes at most 1 step.

   The loop iterates n times.

   $WC_f \in \wr(n)$

   Omega

   Let $n = |\text{nums1}| = |\text{nums2}|$

   Let nums1 and nums2 be sets that are disjoint.

   $\forall x \in \text{nums1}, x \notin \text{nums2}$

   Then a single iteration of the loop takes at least 1 step.

   The loop iterates n times.

   Therefore $WC_f \in \Omega(n)$

   Theta

   Since we've shown that $WC_f \in \wr(n)$, and $WC_f \in \wr(n)$, we can say that $WC_f \in \Theta(n)$

2. Now let's consider what happens if we take the assumption that `nums1` and `nums2` have different lengths. For this question, let n_1 be the length of `nums1` and n_2 be the length of `nums2`.

   (a) What would the worst-case running time of `are_disjoint_lists` be in this case, in terms of n_1 and/or n_2?

   Using similar analysis to *Ex1*, we can say that $WC_f \in \Theta(n_1 \cdot n_2)$

   (b) What would the worst-case running time of `are_disjoint_sets` be, in terms of n_1 and/or n_2?

   Using similar analysis to *Ex2 Q1*, we can say that $WC_f \in \Theta(n_1)$

   (c) What would the worst-case running time of `are_disjoint_sets` be, in terms of n_1 and/or n_2, if we switched the `nums1` and `nums2` in the function body?

   Switching nums1 and nums2 would result in a $WC_f \in \Theta(n_2)$

   (d) Can you write an implementation of `are_disjoint_sets` whose worst-case running time is $\Theta(\min(n_1, n_2))$?

```python
def are_disjoint_sets(nums1, nums2) -> bool:
    """Return whether the sets are disjoint or not, with efficiency Theta(min(n_1, n_2)"""
    if(len(nums1) > len(nums2)):
        for x in nums2:
            if x in nums1:
                return False
        return True
    else:
        for x in nums1:
            if x in nums2:
                return False
        return True
```

3 Additional exercises: Substring search

Here is an algorithm which takes two strings and determines whether the first string is a substring of the second.

```python
def substring(s1: str, s2: str) -> bool:
    """Return whether s1 is a substring of s2."""
    for i in range(0, len(s2) - len(s1)):                              # n2 - n1 iterations
        # Check whether s1 == s2[i: i + len(s1) - 1]
        found_match = all(s1[j] == s2[i + j] for j in range(0, len(s1)))    # n1 iterations

        # If a match has been found, stop and return True.
        if found_match:
            return True

    return False
```

1. Let n_1 represent the length of `s1` and n_2 represent the length of `s2`. Find a tight asymptotic upper bound on the worst-case running time of `substring` in terms of n_1 and/or n_2.

   The worst case for this function would be if `s2` was not a substring of `s1`. This means that the function would not return until it reaches the end of the string.

   This would lead to a worst case running time $\in \Theta(n_1 \cdot (n_2 - n_1))$

2. Find an input family whose running time matches the upper bound you found in Question 1. You may assume that $n_1 \mid n_2$ (this may make it a bit easier to define an input family).

   Hint: you can pick `s1` to be a string of length n_1 that just repeats the same character n_1 times.

   If you take the string `s1` to be one character repeated n_1 times, and you take the string `s2` to be a different character repeatead n_2 times, then the two strings will be separate and distinct. This means that `s2` would not be a substring of `s2`. This means that the function will go through its entire loop and will then after doing all of that, return `False`.