CSC110 Fall 2021: Term Test 2
Question 2 (Analyzing Algorithm Running Time)

TODO: INSERT YOUR NAME HERE

Wednesday December 8, 2021

Question 2, Part 1

We define the function $g : \mathbb{N} \to \mathbb{R}^{\geq 0}$ as $g(n) = 7n(n-1)^2$. Consider the following statement:

$$g(n) \in \mathcal{O}(n^4)$$

(a) Rewrite the statement $g(n) \in \mathcal{O}(n^4)$ by expanding the definition of Big-O.

Solution:
$\exists c, n_0 \in \mathbb{R}^+$ s.t. $\forall n \in \mathbb{N}, n \geq n_0 \Rightarrow 7n(n-1)^2 \leq c \cdot n^4$

(b) Write the *negation* of the statement from (a), using negation rules to simplify the statement as much as possible.

Solution:
$\forall c, n_0 \in \mathbb{R}^+, \exists n \in \mathbb{N}$ s.t. $n \geq n_0 \wedge 7n(n-1)^2 > c \cdot n^4$

(c) Which of statements (a) and (b) is true? Provide a complete proof that justifies your choice.

In your proof, you may not use any properties or theorems about Big-O/Omega/Theta. Work from the expanded statement from (a) or (b).

Solution:

I think statement (a) is true.

Proof. Want to show: $\exists c, n_0 \in \mathbb{R}^+$ s.t. $\forall n \in \mathbb{N}, n \geq n_0 \Rightarrow 7n(n-1)^2 \leq c \cdot n^4$
Prove using Induction.
Take $c = 7, n_0 = 1$
Let n be an arbitrary natural number such that $n \geq (n_0 = 1)$
What we want to prove becomes: $\forall n \in \mathbb{N}, n \geq 1 \Rightarrow 7n(n-1)^2 \leq 7n^4$

Since $n \geq 1$,
Multiply both sides by n^2, we get $n^3 \geq n^2$
From this, we also know that $(n-1)^2 \leq n^3$
Also, since $n \geq 1$,
Multiply the inequality by -2, we get $-2n \leq -2$
Adding 1 to both sides, we get $-2n + 1 \leq -1$
Putting the two inequalities together, we have $n^2 - 2n + 1 \leq n^3 - 1 \leq n^3$

Factoring the polynomial on the left, we have $(n-1)^2 \leq n^3$
Multiply both sides by $7n$, we get $7n(n-1)^2 \leq 7n^4$
Which is what we want to prove.

$\square$

Question 2, Part 2

Consider the function below.

```
def f(nums: list[int]) -> list[int]:          # Line 1
    n = len(nums)                             # Line 2
    i = 1                                     # Line 3
    new_list = []                             # Line 4
    while i < n:                              # Line 5
        if nums[i] % 2 == 0:                  # Line 6
            list.append(new_list, i)          # Line 7
        else:                                 # Line 8
            new_list = [i * j for j in nums]  # Line 9
        i = i * 3                             # Line 10
    return new_list                           # Line 11
```

(a) Perform an *upper bound analysis* on the worst-case running time of `f`. The Big-O expression that you conclude should be *tight*, meaning that the worst-case running time should be Theta of this expression, but you are not required to show that here.

To simplify your analysis, you may omit all floors and ceilings in your calculations (if applicable). Use "at most" or $\leq$ to be explicit about where a step count expression is an upper bound.

Solution:

Let n be the length of the input list `nums`

There is one loop in the function which loops through `nums` with i increasing exponentially, which will run $\lceil log_3(n) \rceil$ times. Inside the loop, if then number is even, it takes $\mathcal{O}(1)$ to append the item at the end of `new_list`. If the number is odd, it sets `new_list` to a list comprehension which iterates through all number in `nums`, performing an $\mathcal{O}(1)$ multiplication every iteration, which takes exactly n steps, which is a larger running time than if the number is even. Therefore, the inside of the loop will take at most n steps, if all numbers `nums[i]` iterated are odd.

Since there are only constant-time operations outside the loop, the worst-case running time would be $\lceil log_3(n) \rceil$ iterations multiplied by at most n steps per iteration, which is $n \lceil log_3(n) \rceil$ steps.

Since $n \lceil log_3(n) \rceil \in \mathcal{O}(n \lceil log_3(n) \rceil)$, we can conclude that $WC_f(n) \in \mathcal{O}(n \lceil log_3(n) \rceil)$

(b) Perform a *lower bound analysis* on the worst-case running time of `f`. The Omega expression you find should match your Big-O expression from part (a).

Hint: you don't need to try to find an "exact maximum running-time" input. *Any* input family whose running time is Omega of ("at least") the bound you found in part (a) will yield a correct analysis for this part.

Solution:

Let n be the length of the input list `nums`, let `nums` be the list of length n which every number is 1.

In this case, the if statement inside the loop always runs line 9 that takes n steps, and then the $i = i * 3$ statement, which is 1 step, which is a total of $n+1$ steps. The loop still iterates $\lceil log_3(n) \rceil$ times. Since there are only constant-time operations outside the loop, the total number of steps for this input is $(n+1) \lceil log_3(n) \rceil + c$ which $c \in \mathbb{N}$ is a constant, which is $WC_f(n) \in \Omega(n \lceil log_3(n) \rceil)$

SUBMIT THIS FILE AND THE GENERATED PDF q2.pdf FOR GRADING