EECS 203: Discrete Mathematics
Winter 2024
Homework 11

Due **Tuesday, April 23**, 10:00 pm

No late homework accepted past midnight.

Number of Problems: $6 + 0$ Total Points: $100 + 0$

- **Match your pages!** Your submission time is when you upload the file, so the time you take to match pages doesn't count against you.
- Submit this assignment (and any regrade requests later) on Gradescope.
- Justify your answers and show your work (unless a question says otherwise).
- By submitting this homework, you agree that you are in compliance with the Engineering Honor Code and the Course Policies for 203, and that you are submitting your own work.
- Check the syllabus for full details.

Individual Portion

1. Big-Oreo [15 points]

Give the tightest big-O estimate for each of the following functions. Justify your answers.

(a) $f(n) = (2^n + n^n) \cdot (n^3 + n \log n^n)$

(b) $g(n) = (n^n + n!) \cdot (n + 1) \quad + \quad (n^3 + 3^n) \cdot (\sqrt{n} + \log n)$

(c) $h(n) = (n^n + n^2) \cdot (n^n + n) \quad + \quad (\log 3 + n^n) \cdot (n^2 + n^n)$

Solution:

For each of these, we can expand out the terms, or we can multiply the fastest-growing terms in each sub-expression, since these are the terms that will end up dominating the big-O estimate. This is an application of the sum rule (to the sub-expressions), and then the product rule.

(a) For $f(n)$, the fastest growing term of the first expression is n^n. The fastest growing term of the second expression is n^3, which is greater than $n \log n^n = n^2 \log n$ (by log rules). So $f(n)$ is $O(n^n n^3) = O(n^{n+3})$.

(b) For $g(n)$, the product of the fastest growing terms in the first expression is $n^n n = n^{n+1}$, and the product of the fastest growing terms in the second expression is $3^n \sqrt{n}$. The two expressions are summed, so the faster growing product dominates the complexity, so $g(n) \in O(n^{n+1})$.

(c) For $h(n)$, the product of the fastest growing terms in the first expression is $n^n(n^n) = n^{2n}$ and the product of the fastest growing terms in the second expression is also $n^n(n^n) = n^{2n}$. So $h(n)$ is $O(n^{2n})$.

Draft Grading Guidelines [15 points]

For each part:

+2 correct answer
+3 correct justification

2. On the Run [20 points]

Give the tightest big-O estimate for the number of operations (where an operation is arithmetic, a comparison, or an assignment) used in each of the following algorithms. **Explain your reasoning.**

(a) **function** $\textsc{doubleTrouble}(a_1, \ldots, a_N \in \mathbb{R}, j \in \mathbb{R})$

```
    j ← 1
    for i := 1 to N do
        if i = j then
            j ← 2j
        end if
    end for
    return j
end function
```

(b)
```
function sumSquares(N ∈ ℤ⁺)
    if N = 1 then
        return 1
    end if
    value ← sumSquares(N − 1) + N^2
    return value
end function
```

(c)
```
function findLTMinProduct(a_1, ..., a_N ∈ ℝ)
    p ← 203
    for i := 1 to N do
        for j := 1 to N do
            if a_i a_j < p then
                p ← a_i a_j
            end if
        end for
    end for
    numLTMinProduct ← 0
    for k := 1 to N do
        if a_k < p then
            numLTMinProduct ← numLTMinProduct + 1
        end if
    end for
    return numLTMinProduct
end function
```

(d)
```
function subtractAndAdd(N ∈ ℤ)
    while N > 0 do
        if N is even then
            N ← N − 3
        end if
        if N is odd then
            N ← N + 1
        end if
    end while
    return N
```

end function

(e) **function** SEARCH($a_1, \ldots, a_N \in \mathbb{R}, target \in \mathbb{R}$)

```
function SEARCH(a_1, ..., a_N ∈ ℝ, target ∈ ℝ)
    left ← 1
    right ← N
    while True do
        mid ← ⌊(left+right)/2⌋
        if a_mid = target then
            return mid
        end if
        if right ≤ left then
            return −1
        end if
        if a_mid < target then
            left ← mid + 1
        end if
        if a_mid > target then
            right ← mid − 1
        end if
    end while
end function
```

Solution:

(a) $O(N)$. There is a constant amount of work done in each iteration of the loop. Since i is incremented by 1 with each iteration of the loop and does not change otherwise, the loop runs N times. Therefore, the function's run time is $O(N)$.

(b) $O(N)$. All operations except for the recursive call take constant time. Because the function keeps calling itself with an input that is decremented by 1 until the input itself is 1, the function is called N times. Because the run time for each of these layers takes constant time, the overall function's run time is $O(N)$.

(c) $O(N^2)$. There are three loops, the first two of which are nested. Within a single iteration of the outer loop, there is a constant amount of work done for each element of the list, so the outer loop does $O(N)$ work per iteration. Since the outer loop executes N times, the overall complexity of the two nested loops is $O(N \cdot N) = O(N^2)$. The third loop also does a constant amount of work for each element in the list, so it is $O(N)$. Therefore, the total run time is $O(N^2 + N) = O(N^2)$.

(d) $O(N)$. If N is even entering the loop, the first conditional statement will decrement N by 3, making it odd, and then the second conditional statement will increment N by 1, meaning the loop iteration overall decrements N by 2. If N is odd entering the loop, the second conditional statement will increment N by 1, making it even, but the next loop's iteration will decrement N by 2, so N is decremented by 1 overall. Regardless of whether N is initially even or odd, it ends up being decremented by 1 or 2, so N eventually becomes non-positive in $2N$ or less loop iterations, breaking out of the loop. Each iteration of the loop takes constant time, so the overall function's run time is $O(N)$.

(e) $O(\log N)$. For each iteration of the loop, the search space is cut roughly in half. These iterations occur until the *target* element is found or if it can be concluded that there is no valid element in the list with a value of *target*. Since there are N elements to start with and roughly half of the list is eliminated from consideration in each iteration of the loop, the run time is $O(\log N)$.

Draft Grading Guidelines [20 points]

Part a:

+2 reports $O(N)$ as runtime
+2 correct and complete explanation

Part b:

+2 reports $O(N)$ as runtime
+2 correct and complete explanation that does not cite the Master Theorem

Part c:

+2 reports $O(N^2)$ as runtime
+1 states that the nested loops are $O(N^2)$
+1 states that the other loop is $O(N)$

Part d:

+2 reports $O(N)$ as runtime
+2 correct and complete explanation

Part e:

+2 reports $O(\log N)$ as runtime
+2 correct explanation, does not need to be a formal proof

3. This One's Bound to be Fun! [18 points]

You are given the following bounds on functions f and g:

- $f(x)$ is $O(203^x x^2)$ and $\Omega(3^x \log x)$
- $g(x)$ is $O(\frac{x!}{2^x})$ and $\Omega(4^x)$

Find the following, simplify your answer as much as possible.

(a) Find the tightest big-O and big-Ω estimates that can be *guaranteed* of $f(x)(g(x))^2$.

(b) Find the tightest big-O and big-Ω estimates that can be *guaranteed* of $f(x) + g(x)$.

(c) Let $h(x) = f(x) - g(x)$. Prove or disprove that $h(x)$ is $\Omega(4^x)$.

Solution:

(a) $f(x)(g(x))^2 = O(203^x x^2 \cdot \frac{(x!)^2}{(2^x)^2}) = O(x^2(x!)^2(\frac{203}{4})^x)$
$f(x)(g(x))^2 = \Omega(3^x \log x \cdot (4^x)^2) = \Omega(48^x \log x)$

We should also check that these bounds are indeed tight. If $f(x) = 203^x x^2$, and $g(x) = \frac{x!}{2^x}$, then $f(x)(g(x))^2$ exactly equals our big-O estimate, so the upper bound is tight. Similarly if $f(x) = 3^x \log x$ and $g(x) = 4^x$, then the lower bound is tight.

(b) $f(x) + g(x) = O(\frac{x!}{2^x})$
$f(x) + g(x) = \Omega(4^x)$

Using similar reasoning to the above, our big-O bound is tight when $g(x) = \frac{x!}{2^x}$, and our big-Ω bound is tight when $f(x) = g(x) = 4^x$.

(c) Consider $f(x) = 5^x + 203$ and $g(x) = 5^x$. Then $f(x) - g(x) = 203 = \Theta(1) \neq \Omega(4^x)$.

Draft Grading Guidelines [18 points]

Part a:

+2 correct big-O
+2 correct big-Ω
+2 correct justification

Part b:

+2 correct big-O
+2 correct big-Ω
+2 correct justification

Part c:

+2 states intention to disprove

+2 $f(x)$ and $g(x)$ within the stated bounds
+2 shows $f(x) - g(x)$ are not $\Omega(4^x)$

4. Big Function Fun [16 points]

Prove or disprove the following:

(a) If $f(x)$ is $O(g(x))$ then $2^{f(x)}$ is $O(2^{g(x)})$.

(b) If $f(x)$ is $O(g(x))$ then $(f(x))^2$ is $O\big((g(x))^2\big)$.

Note that in these proofs you do not need to use the definition of big-O, but can use the properties for combining mathematical functions covered in lecture.

Solution:

(a) This is not necessarily true. Let $f(x) = 2x$ and $g(x) = x$. Then $f(x)$ is $O(g(x))$. Now $2^{f(x)} = 2^{2x} = 4^x$, and $2^{g(x)} = 2^x$, but 4^x is not $O(2^x)$. Indeed, $\frac{4^x}{2^x} = 2^x$, so the ratio grows without bound as x grows, so 4^x is not bounded by 2^x within a constant ratio.

(b) We know that for two functions, $f(x)$ and $g(x)$, if $f(x) \in O(\tilde{f}(x))$ and $g(x) \in O(\tilde{g}(x))$ then $f(x) \cdot g(x) \in O(\tilde{f}(x) \cdot \tilde{g}(x))$. So in this case $(f(x))^2 = f(x) \cdot f(x) \in O(g(x) \cdot g(x)) = O\big((g(x))^2\big)$.

Alternate Solution: Assume $f(x)$ is $O(g(x))$, then by definition there exists some $c \in \mathbb{R}$ where $f(x) \leq cg(x)$. So $(f(x))^2 \leq (cg(x))^2 = c^2(g(x))^2$. Since c^2 is just another constant, by definition, this means $(f(x))^2$ is $O((g(x))^2)$

Draft Grading Guidelines [16 points]

Part a:

+2 chooses to disprove
+3 correct counterexample
+3 correct justification

Part b:

+2 chooses to prove
+3 applies big-O multiplication rule
+3 correct justification

5. Roots and Shoots [16 points]

Suppose f satisfies $f(n) = 2f(\sqrt{n}) + \log_2 n$, whenever n is a perfect square greater than 1, and additionally satisfies $f(2) = 1$.

(a) Find $f(16)$.

(b) Find a big-O estimate for $g(m)$ where $g(m) = f(2^m)$.

Hint: Make the substitution $m = \log_2 n$.

(c) Find a big-O estimate for $f(n)$.

Solution:

(a)

$$\begin{aligned} f(2) &= 1 \\ f(4) = 2f(\sqrt{4}) + \log_2(4) &= 4 \\ f(16) = 2f(\sqrt{16}) + \log_2(16) &= 12 \end{aligned}$$

(b) Let $m = \log_2 n$, so that $n = 2^m$. Then our recurrence becomes $f(2^m) = 2f(2^{\frac{m}{2}}) + m$, since $\sqrt{2^m} = (2^m)^{\frac{1}{2}} = 2^{\frac{m}{2}}$.

Let $g(m) = f(2^m)$. Rewriting the above in terms of g we have $g(m) = 2g(\frac{m}{2}) + m$. The Master Theorem (with $a = 2$, $b = 2$, and $d = 1$) now tells us that $g(m)$ is $O(m \log m)$.

(c) Since $m = \log n$, and $g(m) = f(n)$, this says that our function is $O(\log n \cdot \log \log n)$.

Draft Grading Guidelines [16 points]

Part a:

+3 correct value for $f(4)$
+3 correct value for $f(16)$

Part b:

+2 correct substitution of m yielding $f(2^m) = 2f(2^{\frac{m}{2}}) + m$.
+2 identifies correct values for a, b and d from student's $g(m)$ (recurrence does not need to be correct, just applicable to the master theorem, and this does not need to be explicit)
+2 correctly applies master theorem to $g(m)$

Part c:

+4 correct answer based on part (b)

6. GG Brown Laboratory [15 points]

What is the tightest big-O bound on the runtime complexity of the following algorithm?

```
function BADSEARCH(n)
    if n ≥ 1 then
        BADSEARCH(⌊n/3⌋)
        for i := 1 to n do
            for j := 1 to ⌊n/2⌋ do
                print "Hello I am lost"
            end for
        end for
        BADSEARCH(⌊n/3⌋)
        print "Nevermind I got it"
    end if
end function
```

Solution:

$O(n^2)$.

The recurrence is $T(n) = 2T(\lfloor \frac{n}{3} \rfloor) + O(n^2)$. That means $a = 2$, $b = 3$, and $d = 2$, so $\frac{a}{b^d} = \frac{2}{9}$. Then since $\frac{a}{b^d} = \frac{2}{9} < 1$, by the master theorem, this algorithm is $O(n^2)$.

Grading Guidelines [15 points]

+5 identifies correct recurrence
+5 identifies correct coefficients from recurrence and attempts to use them to apply master theorem
+5 correctly applies master theorem

Grading of Groupwork 10

Using the solutions and Grading Guidelines, grade your Groupwork 10 Problems:

- Use the table below to grade your past groupwork submission and calculate scores.
- While grading, mark up your past submission. Include this with the table when you submit your grading.
- Write whether your submission achieved each rubric item. If it didn't achieve one, say why not.
- For extra credit, write positive comment(s) about your work.
- You don't have to redo problems correctly, but it is recommended!
- See "All About Groupwork" on Canvas for more detailed guidance, and what to do if you change groups.

	(i)	(ii)	(iii)	(iv)	(v)	(vi)	(vii)	(viii)	(ix)	(x)	(xi)	Total:
Problem 1												/15
Problem 2												/20
Total:												/35