Due September 15 (11:59 p.m.)

Instructions

- **Answers should be written in the boxes by modifying the provided Latex template or some other method answers are clearly marked and legible. Submit via Gradescope.**
- **Honors questions are optional. They will not count towards your grade in the course. However you are encouraged to submit your solutions to these problems to receive feedback on your attempts.**
- **You must enter the names of your collaborators or other sources (including use of LLMs) as a response to Question 0. Do NOT leave this blank; if you worked on the homework entirely on your own, please write "None" here. Even though collaborations in groups of up to 3 people are encouraged, you are required to write your own solution.**
- **For the late submission policy, see the website.**
- Acknowledgment - Thank you to Amir, Daji, Oded, Peter and Rotem for help in setting the problem set.

1-0 List all your collaborators and sources: ($-\infty$ points if left blank)

- Tutor
- Textbook
- Lecture 3 notes

Problem 1-1 – Big-O (30 points)

Recall that $f = O(g)$ is defined for functions f and g (both from $\mathbb{N}$ to $\mathbb{R}^+$) to mean that there exist positive constants n_0 and C such that:

$$f(n) \leq C \cdot g(n) \text{ for all } n \geq n_0.$$

We also define $f = \Omega(g)$ if there exist positive constants n_0 and C such that:

$$f(n) \geq C \cdot g(n) \text{ for all } n \geq n_0.$$

And $f = \Theta(g)$ if there exist positive constants C_1, C_2 and an n_0 such that

$$C_1 \cdot g(n) \leq f(n) \leq C_2 \cdot g(n) \text{ for all } n \geq n_0$$

(3 points each) For each of the following functions f, g, state whether $f(n) = O(g(n))$, $f(n) = \Omega(g(n))$, or $f(n) = \Theta(g(n))$. It is possible that multiple options might be true.

Justify your answers in 1-2 sentences (i.e., by using one of the methods for proving asymptotics, the definitions or the limit definitions).

1. $f(n) = 2n^2 \quad g(n) = n^{3/2}(\log n)^3$

Solution:

$$\begin{aligned} f(n) &= 2n^2 \\ g(n) &= n^{3/2}(\log n)^3 \\ \lim_{n\to\infty} \frac{f(n)}{g(n)} &\to \frac{2n^2}{n^{3/2}(\log n)^3} \\ &\to \frac{2n^{0.5}}{(\log n)^3} \\ &\to \text{approaches } \infty \\ f &= \Omega(g) \end{aligned}$$

This limit for Ω goes to ∞, limit for O must go to 0, $f \neq O(g)$, following that $f \neq \Theta(g)$.

2. $f(n) = 25 \cdot 3^n \quad g(n) = n \cdot 4^{n/2}$

Solution:

$$\begin{aligned} f(n) &= 25 * 3^n \\ g(n) &= n * 4^{n/2} \\ \lim_{n\to\infty} \frac{f(n)}{g(n)} &\to \frac{25 * 3^n}{n * 4^{n/2}} \\ &\to 25 * \frac{3^n}{n * 4^n} && \text{higher growth order of exponential} \\ &\to \text{approaches } \infty \\ f &= \Omega(g) \end{aligned}$$

This limit for Ω goes to ∞, limit for O must go to 0, $f \neq O(g)$, following that $f \neq \Theta(g)$.

3. $f(n) = \log n \log\log n \quad g(n) = n(\log\log n)^3$

Solution:

$$\begin{aligned} f(n) &= \log n \log\log n \\ g(n) &= n(\log\log n)^3 \\ \lim_{n\to\infty} \frac{f(n)}{g(n)} &\to \frac{\log n \log\log n}{n(\log\log n)^3} \\ &\to \frac{\log n}{n(\log\log n)^2} && \text{higher growth order of linear and quadratic} \\ &\to \text{approaches 0} \\ f &= O(g) \end{aligned}$$

This limit for O goes to 0, limit for Ω must go to ∞, $f \neq \Omega(g)$, following that $f \neq \Theta(g)$.

4. $f(n) = 2n^2 \quad g(n) = n!$ (Recall $n! = 1 \cdot 2 \cdot 3 \ldots (n-1) \cdot n$)

Solution:

$$f(n) = 2n^2$$
$$g(n) = n!$$
$$\lim_{n\to\infty} \frac{f(n)}{g(n)} \to \frac{2n^2}{n!}$$ factorial growth order grows very quick
$$\to \text{approaches 0}$$ take n ≥ 5 and $C = 1$, limit holds for all values
$$f = O(g)$$

This limit for O goes to 0, limit for Ω must go to ∞, $f \neq \Omega(g)$, following that $f \neq \Theta(g)$.

5. $f(n) = 4^{\log n} \quad g(n) = (\log n)^4$

Solution:

$$f(n) = 4^{logn}$$
$$g(n) = (logn)^4$$
$$\lim_{n\to\infty} \frac{f(n)}{g(n)} \to \frac{4^{logn}}{(logn)^4}$$ exponential growth order overtakes over time
$$\to \text{approaches } \infty$$
$$f = \Omega(g)$$

This limit for Ω goes to ∞, limit for O must go to 0, $f \neq O(g)$, following that $f \neq \Theta(g)$.

Problem 1-2 – Sum Inductions (15 points)

1. Our goal is to show the following fact:

$$\forall n \in \mathbb{N} \quad 1 + 2 + 3 + \ldots + n = \frac{n(n+1)}{2}$$

(a) (1 point) Show the base case.

Solution: Basis. Consider $n = 1$.

$$n = 1$$
$$1 = \frac{(1)(1+1)}{2}$$
$$1 = \frac{2}{2}$$
$$1 = 1$$

(b) (4 points) Show the inductive step.

Solution: Hypothesis. Let $k \in \mathbb{N}$ where $k \geq 1$. Consider $n = k$. Assume

$$1 + 2 + 3 + \ldots + k = \frac{k(k+1)}{2}.$$

Inductive Step. Consider $n = k + 1$. Then

$$\begin{aligned}
1 + 2 + 3 + \ldots + k + (k+1) &= \frac{(k)(k+1)}{2} + (k+1) && \text{from hypothesis above} \\
&= \frac{(k)(k+1) + 2(k+1)}{2} && \text{fraction/denom simplification} \\
&= \frac{k^2 + k + 2k + 2}{2} && \text{distribution} \\
&= \frac{(k+1)(k+2)}{2} && \text{factoring} \\
&= \frac{(k+1)((k+1)+1)}{2} && \text{matches initial sequ. formula for } n = k+1
\end{aligned}$$

2. Now, what if we sum the first n odd numbers? We will show:

$$\forall n \in \mathbb{N} \quad 1 + 3 + 5 + \ldots + (2n-1) = n^2$$

(a) (1 point) Show the base case.

Solution: Basis. Consider $n = 1$.

$$\begin{aligned}
n &= 1 \\
(2(1) - 1) &= (1)^2 \\
(2 - 1) &= 1 \\
1 &= 1
\end{aligned}$$

(b) (9 points) Show the inductive step.

Solution: Hypothesis. Let $k \in \mathbb{N}$ where $k \geq 1$. Consider $n = k$. Assume

$$1 + 3 + 5 + \ldots + (2k - 1) = k^2$$

Inductive Step. Consider $n = k + 1$. Then

$1 + 3 + 5 + \ldots + (2k - 1) + (2k - 1 + 2) = k^2 + (2k - 1 + 2)$ from hypothesis above

$= k^2 + 2k + 1$ simplification

$= (k + 1)^2$ matches initial sequ. formula for n = k + 1

Problem 1-3 – Ordering Big-O (10 Points)

Rank the following functions by order of growth (You need not prove the correctness of your ranking). That is, find an order f_a, f_b, $f_c \ldots f_e$ so that $f_a = \mathcal{O}(f_b)$, $f_b = \mathcal{O}(f_c)$, and so on:

1. $\sqrt{n}$
2. $n^{\frac{1}{n}}$
3. n^n
4. $2^{\log_{10}(n)}$
5. n

Solution:

$f_a \to n^{1/n}$ implying $f_a = \mathcal{O}(f_b)$

$f_b \to 2^{\log_{10}(n)}$ implying $f_b = \mathcal{O}(f_c)$

$f_c \to \sqrt{n}$ implying $f_c = \mathcal{O}(f_d)$

$f_d \to n$ implying $f_d = \mathcal{O}(f_e)$

$f_e \to n^n$

Problem 1-4 – Asymptotics in the Wild (5 Points)

1. (1 point each) For each of the following scenarios, determine the Big-O complexity. (There is one each of $O(1)$, $O(n)$, $O(n^2)$, and $O(2^n)$).
   (a) Washing n dishes by hand.
   (b) Introducing n friends to one another at a party.
   (c) Eating a single chip out of a bag of n chips (you can pick any one).
   (d) Trying every possible combination of n potential ice cream toppings.

Solution:

a. $\mathcal{O}(n)$
b. $\mathcal{O}(n^2)$
c. $\mathcal{O}(1)$
d. $\mathcal{O}(2^n)$

2. (1 point) What is another real-world example of an $O(n)$ activity?

Solution: Someone going around and shaking hands with everyone at a party of n people.

Problem 1-5 – Find the Bug (10 Points)

Find the bug in the following proof and explain the error in 1-2 sentences. Define a recurrence:

$$T(n) = 2T(n/2) + n, \qquad T(1) = 2$$

"Claim": $T(n) = O(n)$
"Proof": By induction on n. Assume the claim holds for $1, 2, \ldots, n-1$. Let us prove it for n:

$$\begin{aligned} T(n) &= 2 \cdot T(n/2) + n \\ &= 2 \cdot O(n/2) + n \\ &= O(n) \ . \end{aligned}$$

Solution: The first problem is that the proof uses the initial claim within the proof itself by substituting the $\mathcal{T}(n/2)$ for $\mathcal{O}(n/2)$. This appears logically invalid because the proof to prove the claim shouldn't be using the claim itself. The second problem is that the proof doesn't even seem to use induction even though it says it will. The third problem is that the proof seems to be distributing the outer 2

into the parentheses, canceling both 2s, resulting in $\mathcal{O}(n)$. However, the $\mathcal{T}$ within the recurrence is a function with an input of $n/2$. The multiplicative 2 is acting on the output of the function $T(n/2)$, it can't just be distributed. In fact, the correct claim should be $\mathcal{T}(n) = \mathcal{O}(n \log n)$ because the n is being halfed till $n = 1$, which creates $logn$ levels. Both branches of each level are being considered, which accounts for the multiplicative 2; linear work is being done at each level, accounting for the additive n.

Problem 1-6 – Multiplications (10 Points)

Describe a procedure that, given four integers a, b, c, d, outputs the three numbers ab, cd and $ad + bc$ and uses only three multiplications (four would be obvious). You are free to use as many additions and subtractions as you wish. (Hint: Consider the product $(a + c)(b + d)$.)

Solution:

$1. a * b = ab$	first multiplication
$2. c * d = cd$	second multiplication
$3. ((a + c) * (b + d)) - ab - cd = ad + bc$	third multiplication

Problem 1-7 – Counting Inversions (20 points)

Let $A = A[1, \ldots, n]$ to be an array of n distinct integers. We call a pair (i, j) an *inversion* of A if $i < j$ but $A[i] > A[j]$. In this problem, we will work up how to count the number of inversions of an array.

1. (2 points) List all inversions of the array $A = \langle 9, 2, 5, 7, 1, 10 \rangle$.

Solution:
Inversions with 0-based indices:
(0,1), (0,2), (0,3), (0,4), (1,4), (2,4), (3,4)

2. (6 points) Assume A is some permutation of the set $[n] = \{1, \ldots, n\}$. Which array(s) maximize the number of inversions? State the number of inversions as a function of n.

Solution: The array that would maximize the number of inversions would be the array that is element-ordered from largest to smallest. Moving from the end of this array, the number of inversions starts at 0 and increments by 1 as you move backward through the array to the first index with the largest index. So, from the end of the array moving backwards, the number of inversions will be $0, 1, 2, 3, 4 \ldots$. This is a regularly incrementing sequence of formula $(n(n+1))/2$ where we're not tracking the number of elements, but rather the number of inversions, so $n-1$. Thus, the number of inversions as a function would be $((n-1)((n-1)+1))/2$.

3. (2 points) If an array A has k inversions, what will the running time of Insertion Sort on A be in terms of n and k? Explain why.

Solution:
The running time of Insertion Sort on A will be $\Theta(n+k)$. I am pretty sure Insertion runs on a $\Theta(n)$ complexity for the outer loop iterations anyway. Every time the inner loop correctly sorts an element, it removes the inversion that element had. This ends up with the number of left-shifts/sorts being equal to the number of inversions in the array. The math ends up being the linear time the outer loop is running in, along with the number of inversions it has to deal with: $\Theta(n+k)$.

4. (10 points) Give an algorithm that determines the number of inversions in an array consisting of n numbers in $\Theta(n \log n)$ worst-case time. Also prove the correctness and run time bounds for your algorithm. (Hint: Modify merge sort.)

Solution: The algorithm edits the mergeSort algorithm by counting the number of inversions during the actual merge. You'd be adding (index variable iterating through array of first half of sorted elements subtracted from the length of that array) to an inversion count in the final else case of the algo. We would also pass that inversion count as a parameter to each recursive call of MergeSort that sorts each new half array, as well as return both the sorted array as well as the inversion count.

Given that A and B are sorted:
$i = 1, j = 1$
while $i + j - 1 \leq m + n\{$
if $j > n$ then $C[i+j-1] = A[i] \&\& i = i+1$
elif $i > m$ then $C[i+j-1] = B[j] \&\& j = j+1$
elif $A[i] \leq B[j]$ then $C[i+j-1] = A[i] \&\& i = i+1$
else $C[i+j-1] = B[j] \&\& j = j+1 \&\& inv+ = len(A) - i$

Proof of correctness that the algo returns a sorted array and inversion count:
Base case $n = 1$. Claim holds for the base case by returning the array itself and 0, representing the number of inversions.
Inductive Hypothesis. Let A be any array of length n. The algo splits A into length left n_1 and right n_1, with $n_1 + n_2 = n$. Algorithm(left) will return n_1 sorted along with the number of inversions in the left half and vice versa for the right. Then, Algorithm(left, right, inv), merging the two sorted halves and summing into inv. Any cross-inversion is counted only once because each cross-inversion has one index in the left half and one in the right. Merge processes elements in order and doesn't revisit. By strong induction, the algo is correct for all n.

Proof of runtime bounds with recursion tree:
$\mathcal{T}(n)$ indicates the worst-case. Can write the recurrence as $\mathcal{T}(n) = 2\mathcal{T}(\frac{n}{2}) + \Theta n$.
Recurrence Tree Description: Root cost is cn and has two childs at each level/from each node to merge/traverse. Depth of tree will be $log_2 n$ until we have subarrays of len 1. Total work ends up being $\Theta(n \log n)$.

Drawn recurrence tree:
Image File