CMPSC 465 Data Structures & Algorithms
Fall 2024 Mingfu Shao and Debarati Das

HW 1

Formatting: Start a new page for each problem.

1. (16 pts.) Asymptotic Growth

In each of the following situations, indicate whether $f(n) = O(g)$, $f(n) = \Omega(g)$, or $f(n) = \Theta(g)$.

- $f(n) = n^3 + n^2, \quad g(n) = n^3$
- $f(n) = n \log n, \quad g(n) = (\log n)^3$
- $f(n) = 2^n, \quad g(n) = 2^{n-1}$
- $f(n) = n^3 2^n, \quad g(n) = 3^n$
- $f(n) = \sqrt{n}, \quad g(n) = \log n$
- $f(n) = \log(n!), \quad g(n) = n \log n$
- $f(n) = n!, \quad g(n) = 2^n \cdot n^2$
- $f(n) = n^{\log \log n}, \quad g(n) = \log(n^{\log n})$

Answer:

- $f(n) = \Theta(g)$
  Explanation: $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n^3+n^2}{n^3} = \lim_{n\to\infty}(1 + \frac{1}{n}) = 1$
- $f(n) = \Omega(g)$
  Explanation: $g(n) = (\log n)^2 \log n$, n grows faster than $(\log n)^2$
- $f(n) = \Theta(g)$
  Explanation: $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{2^n}{2^{n-1}} = 2$
- $f(n) = O(g)$
  Explanation: $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n^3 2^n}{3^n} = \lim_{n\to\infty} n^3 (\frac{2}{3})^n = 0$
- $f(n) = \Omega(g)$
- $f(n) = \Theta(g)$
  Explanation 1: The proof is similar to how we proved problem 2.1 in recitation 1. We know $\log(n!) = \sum_{k=1}^{n} \log k = \log 1 + \log 2 + \cdots + \log n$.

  Upper bound: Since $k \le n$, every term in the sum is at most n, so

  $$\log(n!) \le \sum_{k=1}^{n} \log n = n \log n.$$

  Lower bound: We only look at the second half of the sum. Each of these terms is at least $\log \frac{n}{2}$, and there are $\frac{n}{2}$ such terms (assuming without loss of generality that n is even). Therefore:

  $$\log(n!) \ge \frac{n}{2} \log \frac{n}{2} = \frac{n}{2}(\log n - \log 2) = \frac{1}{2} n \log n - \frac{\log 2}{2} n.$$

Explanation 2: If you're familiar with Stirling's approximation, you can also prove it more easily by computing the limit:

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=\lim_{n\to\infty}\frac{\log(n!)}{n\log n}\approx\lim_{n\to\infty}\frac{n\log n-n}{n\log n}=\lim_{n\to\infty}(1-\frac{1}{\log n})=1$$

- $f(n)=\Omega(g)$,
  Explanation: $f(n)=n(n-1)(n-2)!=O(n^2n!)$, n^2 grows slower than 2^n, thus $f(n)=\Omega(g)$
- $f(n)=\Omega(g)$
  Explanation: Recall the logarithm theorem $a^{\log_b c}=c^{\log_b a}$ and $\log_b a^c=c\log_b a$, and thus we have:

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=\lim_{n\to\infty}\frac{(\log n)^{\log n}}{(\log n)^2}=\lim_{n\to\infty}(\log n)^{(\log n-2)}=\infty$$

2. (17 pts.) Let f,g be any functions from $\mathbb{N}$ to $(0,\infty)$. Then prove or disprove the following:

- (3 pts) $f=o(g)\Rightarrow f=O(g)$
- (3 pts) $f=O(g)\vee f=\Omega(g)$
- (3 pts) $f=O(g)\wedge f\neq\Theta(g)\Rightarrow f=o(g)$
- (4 pts) Let $\mathcal{F}$ denote the set of all functions from $\mathbb{N}$ to $(0,\infty)$. Then Θ defines an equivalence relation on $\mathcal{F}$. *(A relation $\mathcal{R}$ on a set S is an equivalence relation if it is reflexive, symmetric and transitive.)*
- (4 pts) For any increasing sublinear function $t:(0,\infty)\to(0,\infty)$, $f=O(g)\Rightarrow t\circ f=O(t\circ g)$. We define t is *sublinear* if for all $\alpha>0$, $x>0$, we have $t(\alpha x)\leq\alpha t(x)$. Also, $t\circ f$ is a function defined as $(t\circ f)(n)=t(f(n))$.

Answer:

- True. *Proof.* Since $f=o(g)$, by the definition of small-o, we know that for every $c>0$, there exists integer $M_c\geq 0$ such that $f(n)\leq c\cdot g(n)$ when $n\geq M_c$. Pick $c=1$. Then there must exists integer $M\geq 0$ such that $f(n)\leq g(n)$ when $n\geq M$. This shows that $f=O(g)$, by the definition of big-O.
- False. Consider counter-example where $f(n)=n$ and $g(n):=\begin{Bmatrix}n^2, & n \text{ is even}\\ 1, & n \text{ is odd}\end{Bmatrix}$. We first prove $f\neq O(g)$. This is because there does not exist $c>0$ and $N\geq 0$ such that $f(n)\leq c\cdot g(n)$ when $n\geq N$. To see it, note that n can be an arbitrarily large odd number, and for such odd n, we have $f(n)=n$ and $g(n)=1$ and it is not possible to have $f(n)=n\leq c\cdot g(n)=c$ since c is a constant and n can be arbitrarily large. Similarly, we can show $f\neq\Omega(g)$, i.e., there does not exist $c>0$ and $N\geq 0$ such that $f(n)\geq c\cdot g(n)$ when $n\geq N$. This is because n can be an arbitrarily large even number, and for such even n, we have $f(n)=n$ and $g(n)=n^2$ and it is not possible to have $f(n)=n\geq c\cdot g(n)=c\cdot n^2$ since c is a constant and n can be arbitrarily large.
- False. Counter-example: $f(n):=\begin{Bmatrix}n, & n \text{ is even}\\ 1, & n \text{ is odd}\end{Bmatrix}$ and $g(n)=n$. Clearly, $f=O(g)$ because $f(n)\leq g(n)$. It is also obvious that $f\neq\Omega(g)$ using a similar argument in above part. (Details: n can be an arbitrarily large odd integer, and for such n we have $f(n)=1$ and hence not possible to have $f(n)=1\geq c\cdot f(n)=c\cdot n$ since c is a constant while n can be arbitrarily large.) Combined, $f\neq\Theta(g)$. But, $f\neq o(g)$. This is because, for any $0<c<1$ we cannot find $N_c\geq 0$ such that $f(n)\leq c\cdot g(n)$, since we can always find an abitrarily large even n and for such n we have $f(n)=n>c\cdot g(n)=c\cdot n$ as $c<1$.

- True. *Proof.* We show reflexivity, symmetry, and transitivity for Θ.
  - **Reflexivity:** We need to show that every function is Θ-related to itself. For any function $f \in \mathcal{F}$, since $f(n) = f(n)$ for all n, it follows that $f = O(g)$ and $f = \Omega(n)$ and hence $f = \Theta(f)$.
  - **Symmetry:** We need to show that if $f = \Theta(g)$, then $g = \Theta(f)$. By the definition of Θ, if $f = \Theta(g)$, then we have $f = O(g)$ and $f = \Omega(g)$. $f = O(g)$ implies $g = \Omega(f)$, and $f = \Omega(g)$ implies $g = O(f)$. Combining $g = \Omega(f)$ and $g = O(f)$ we get $g = \Theta(f)$.
  - **Transitivity:** We need to show that if $f = \Theta(g)$ and $g = \Theta(h)$, then $f = \Theta(h)$.
    We know that $f = \Theta(g)$ and $g = \Theta(h)$ imply $f = O(g)$ and $g = O(h)$; we first use these two to derive that $f = O(h)$. By definition of big-O, there exists $c_1 > 0$ and $N_1 \geq 0$ such that $f(n) \leq c_1 \cdot f(n)$ when $n \geq N_1$, and there exists $c_2 > 0$ and $N_2 \geq 0$ such that $g(n) \leq c_2 \cdot h(n)$ when $n \geq N_2$. Let $c = c_1 \cdot c_2$ and $N = \max\{N_1, N_2\}$. Therefore, when $n \geq N$, we must have $f(n) \leq c_1 \cdot g(n) \leq c_1 \cdot c_2 \cdot h(n) = c \cdot g(n)$. This proves that $f = O(h)$.
    We also know that $f = \Theta(g)$ and $g = \Theta(h)$ imply $f = \Omega(g)$ and $g = \Omega(h)$. We can use these two to derive that $f = \Omega(h)$ in a similar way. By definition of big-Omega, there exists $c_1 > 0$ and $N_1 \geq 0$ such that $f(n) \geq c_1 \cdot f(n)$ when $n \geq N_1$, and there exists $c_2 > 0$ and $N_2 \geq 0$ such that $g(n) \geq c_2 \cdot h(n)$ when $n \geq N_2$. Let $c = c_1 \cdot c_2$ and $N = \max\{N_1, N_2\}$. Therefore, when $n \geq N$, we must have $f(n) \geq c_1 \cdot g(n) \geq c_1 \cdot c_2 \cdot h(n) = c \cdot g(n)$. This proves that $f = \Omega(h)$.
- True. *Proof.* $f = O(g)$ implies there exists some $N \geq 0$ and $c > 0$ s.t. $f(n) \leq c \cdot g(n)$ for all $n \geq N$. For the same c and N, since t is increasing, we have $t \circ f(n) = t(f(n)) \leq t(c \cdot g(n))$. By further using the sublinearity property of t, we get $t(c \cdot g(n)) \leq c \cdot t(g(n)) = c \cdot t \circ g(n)$. Combined, for the same c and N, we have $t \circ f(n) \leq c \cdot t \circ g(n)$ for all $n \geq N$. By definition, this proves $t \circ f = O(t \circ g)$.

3. (15 + 5 (bonus) pts.) For each pseudo-code below, give the asymptotic running time in Θ notation. You may assume that standard arithmetic operations take $\Theta(1)$ time. (The 4th one is a bonus problem.)

1.
```
k := 0;
for i := 1 to n do
    for j := i to n do
        k := k + 1;
    end
end
```

Explanation:

The outer loop runs for $i = 1$ to n. For a given i, the inner loop runs for j from i to n, and for each j it takes $\Theta(1)$ time to do the "$k := k + 1$". The total running time is therefore:

$$\sum_{i=1}^{n}\sum_{j=i}^{n} 1 = \sum_{i=1}^{n}(n - i + 1) = \sum_{i=1}^{n}(n + 1 - i) = (n + 1) \cdot n - \sum_{i=1}^{n} i = \frac{n(n + 1)}{2}$$

Answer: Therefore, the running time is $\Theta(n^2)$.

2.
```
for i := 1 to n do
    for j := 1 to n do
        k := j;
        while k ≥ 1 do
            k := k − 1;
        end
    end
end
```

Explanation:

- The outer ‘for‘ loop runs from $i = 1$ to n, thus executing n times.
- The inner ‘for‘ loop also runs from $j = 1$ to n, meaning it also executes n times.
- Inside the inner loop, there's a ‘while‘ loop that runs while $k \geq 1$, where k starts as j and decrements by 1 until $k = 0$. This means the ‘while‘ loop runs j times for each value of j.

Hence, the total running time of the algorithm is:

$$T(n) = \sum_{i=1}^{n} \sum_{j=1}^{n} j = \sum_{i=1}^{n} \frac{n(n+1)}{2} = \frac{n^3 + n^2}{2}$$

Answer: Therefore, the total running time is $\Theta(n^3)$.

3.
```
for i := 1 to n do
    j := 1;
    while j ≤ n do
        j := j × 2;
    end
end
```

Explanation:

- The outer ‘for‘ loop runs from $i = 1$ to n, so it executes n times.
- Inside the outer loop, the ‘while‘ loop starts with $j = 1$ and keeps doubling j until $j > n$. This means the number of iterations of the ‘while‘ loop is proportional to the number of times j can be doubled before it exceeds n.

The value of j increases as $1, 2, 4, 8, \ldots$, up to $2^k \leq n$. The number of iterations is therefore $\log_2(n)$. Thus, the ‘while‘ loop runs $\Theta(\log n)$ times for each iteration of the outer ‘for‘ loop.

Answer: Since the outer loop runs n times, the total running time of the algorithm is:

$$T(n) = n \times \Theta(\log n) = \Theta(n \log n)$$

4.
```
for i := 1 to n do
    j := i;
    while j ≤ n do
        j := j + i;
    end
end
```

Explanation:

- The outer ‘for‘ loop runs from $i = 1$ to n, meaning it executes n times.
- Inside the outer loop, the ‘while‘ loop starts with $j = i$ and increments j by i in each iteration until $j > n$.

For a given i, the number of iterations of the ‘while‘ loop is the number of steps it takes to go from i to n by increments of i. This is approximately $\frac{n}{i}$.
Thus, the total number of iterations of the ‘while‘ loop over all values of i is:

$$\sum_{i=1}^{n} \frac{n}{i}$$

This is n times the harmonic series:

$$n \times H_n = n \times (\log n + O(1))$$

Answer: Therefore, the total running time is $\Theta(n \log n)$.

4. (15 pts.) Master’s Theorem

Solve each of the following recurrences. Give the closed form of $T(n)$ in Θ-notation. You may assume that n is of a special form (e.g., a power of two or another number) and that the recurrence has a convenient base case that is $\Theta(1)$.

- (3 pts) $T(n) = 4 \cdot T(n/4) + n$
- (3 pts) $T(n) = 6 \cdot T(n/3) + n^2$
- (3 pts) $T(n) = 4 \cdot T(n/8) + \log n$
- (3 pts) $T(n) = 2 \cdot T(n/2) + n \log n$
- (3 pts) $T(n) = 8 \cdot T(n/3) + 2^n$

Answer:

- $T(n) = \Theta(n \log n)$
  Explanation: Apply the Master’s theorem, where $a = 4, b = 4, d = 1$. We have $d = \log_b a$. Hence, we apply Case 1, leading to $T(n) = \Theta(n^d) = \Theta(n \log n)$.
- $T(n) = \Theta(n^2)$
  Explanation: We have $a = 6, b = 3, d = 2$, which means $d > \log_b a = \log_3 6$. Hence, we apply Case 2, leading to $T(n) = \Theta(n^2)$.
- $T(n) = \Theta(n^{2/3})$
  Explanation: We have $a = 4, b = 8, d = 0$, which means $d < \log_b a = \log_8 4 = 2/3$. Hence, we apply Case 2, leading to $T(n) = \Theta(n^{2/3})$.
- $T(n) = \Theta(n \log^2 n)$
  Explanation: We have $a = 2, b = 2, d = 1$, which means $d = \log_b a$. Hence, we apply Case 1, leading to $T(n) = \Theta(n \log^2 n)$.
- $T(n) = \Theta(2^n)$
  Explanation: The running time of the merging step takes $\Theta(2^n)$, which grows faster than any polynomial-time function. This makes us guess that the running time is dominated by $\Theta(2^n)$. We now prove it.

Note that $T(n) = 8 \cdot T(n/3) + 2^n \geq 2^n$. Hence $T(n) = \Omega(2^n)$. We therefore just need to prove $T(n) = O(2^n)$. By definition, that requires to show the existence of $c > 0$ and $N \geq 0$ such that $T(n) \leq c \cdot 2^n$ when $n \geq N$. Without loss of generality, we assume that n is a power of 3, that is, we assume there exists k such that $n = 3^k$. Now we need to prove there exists $c > 0$ and $K \geq 0$ such that $T(3^k) \leq c \cdot 2^{3^k}$ when $k \geq K$.

We prove above by induction. We choose $c = 2$ and $K = 1$. The base case is $k = 1$, i.e., $n = 3^k = 3$. It is easy to verify that $T(n) = T(3) = 8 \cdot T(1) + 2^3 = 8 + 8 = 16$, while $c \cdot 2^{3^k} = 2 \cdot 2^3 = 16$. So, the base case holds.

For the inductive step, assume that $T(3^k) \leq c \cdot 2^{3^k}$ holds up to $n = 3^k$. Now we prove it for $n = 3^{k+1}$. We can write:

$$\begin{aligned}
T(3^{k+1}) &= 8 \cdot T(3^{k+1}/3) + 2^{3^{k+1}} \\
&= 8 \cdot T(3^k) + 2^{3^{k+1}} \\
&\leq 8 \cdot c \cdot 2^{3^k} + 2^{3^{k+1}} \\
&= 16 \cdot 2^{3^k} + 2^{3^{k+1}} \\
&= 2^{3^k + 4} + 2^{3^{k+1}} \\
&\leq 2 \cdot 2^{3^{k+1}}
\end{aligned}$$

The last inequality holds because $3^k + 4 \leq 3^{k+1}$ when $k \geq 1$.

Rubric:

Problem 1, 16 pts

2 points for each correct solution

Problem 2, 17 pts

- Parts (i), (ii), (iii) and (v) - Full credit if correct proof shown. Partial credit (1 pt) if incorrect proof, but correct option out of T and F mentioned; zero otherwise.
- Part (iv) - Full credit if correct proof shown for all 3 cases shown. Partial credit is: +1 for correct proof of reflexivity, +1 for correct proof of symmetry and +1 or correct proof of transitivity.

Problem 3, 15 + 5 (Bonus) pts

For each part:

- full credit (+5) for correct solution with proper justification.

- partial credict (+3) for right answer with no or incorrect justification.

- partial credit (+4) for right answer with partially correct justification.

Problem 4, 15 pts

For each part:

- full credit (+3) for correct solution with proper justification.

- partial credict (+2) for right answer with no, or incorrect justification.