Homework 2

References

Collaborated with Crystal Huang and Sewon Kim. See Master theorem (analysis of algorithms) – Wikipedia.

Question 1

$f = \mathcal{O}(g)$ is defined for functions f and g (both from $\mathbb{N}$ to $\mathbb{N}$) to mean that exist a positive constant C such that for sufficiently large n, $f(n) \leq C \cdot g(n)$. A bit more formally, there exist positive constants n_0 and C such that:

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

For each of the following statements either prove the statement if it is true or otherwise provide a counterexample and justify why your counterexample is indeed a counterexample:

1. If $f = \mathcal{O}(g)$ then $g = \mathcal{O}(f)$.

   Counterexample. Consider $f(n) = n$ and $g(n) = n^2$. Now $n \leq 1 \cdot n^2$ for all $n \geq 2$. There exist positive constants $n_0 = 2$ and $C = 1$ such that $f(n) \leq C \cdot g(n)$ for all $n \geq n_0$, therefore $f = \mathcal{O}(g)$.

   Assume, for the sake of contradiction, that $g = \mathcal{O}(f)$. Then there exist constants n_0' and C' for which $g(n) \leq C' \cdot f(n)$ for all $n \geq n_0'$. This implies that there exists a constant $\lim_{n\to\infty} \frac{g(n)}{f(n)}$. However, substituting f and g gives $\lim_{n\to\infty} \frac{n^2}{n}$, which is unbounded. This contradicts the hypothesis. We conclude that for $f(n) = n$ and $g(n) = n^2$, we have $g \neq \mathcal{O}(f)$.

   Hence, the claim that if $f = \mathcal{O}(g)$, then $g = \mathcal{O}(f)$, is false. □

2. If $f = \mathcal{O}(g)$ and $g = \mathcal{O}(f)$ and $\forall n, f(n) > g(n)$ then $f - g = \mathcal{O}(1)$.

   Counterexample. Consider $f(n) = 2n$ and $g(n) = n$.

   Now $2n \leq \frac{1}{2}n$ for all $n \geq 1$. There exist positive constants $n_0 = 1$ and $C = \frac{1}{2}$ such that $f(n) \leq C \cdot g(n)$ for all $n \geq n_0$, therefore $f = \mathcal{O}(g)$.

   Next, $n \leq \frac{1}{2} \cdot 2n$ for all $n \geq 1$. There exist positive constants $n_0' = 1$ and $C' = \frac{1}{2}$ such that $g(n) \leq C' \cdot f(n)$ for all $n \geq n_0'$, therefore $g = \mathcal{O}(f)$.

   Of course, $2n > n$ for all n, therefore $f(n) > g(n)$ for all n.

   Assume, for the sake of contradiction, that $f - g = \mathcal{O}(1)$. Then there exist positive constants n_0'' and C'' such that $f(n) - g(n) \leq C'' \cdot 1$ for all $n \geq n_0''$. This implies that there exists a constant $\lim_{n\to\infty} [f(n) - g(n)]$. However, substituting f and g gives $\lim_{n\to\infty} [2n - n]$, which is unbounded. This contradicts the hypothesis. We conclude that for $f(n) = 2n$ and $g(n) = n$, we have $f - g \neq \mathcal{O}(n)$.

   Hence, the claim that if $f = \mathcal{O}(g)$ and $g = \mathcal{O}(f)$, and for all n, we have $f(n) > g(n)$, then $f - g = \mathcal{O}(1)$, is false. □

3. If $f = \mathcal{O}(g)$ and $g = \mathcal{O}(f)$ then $\frac{f}{g} = \mathcal{O}(1)$.

   Proof. Suppose $f = \mathcal{O}(g)$ and $g = \mathcal{O}(f)$. Then there exist positive constants n_0, n_0', C and C' such that $f(n) \leq C \cdot g(n)$ for $n \geq n_0$ and $g(n) \leq C' \cdot f(n)$ for $n \geq n_0'$. Observe

   $$f(n) \leq C \cdot g(n).$$

   $$\frac{f(n)}{g(n)} \leq C \cdot \frac{g(n)}{g(n)} \leq C.$$

   There exist positive constants n_0 and C for which $\frac{f(n)}{g(n)} \leq C \cdot 1$ for all $n \geq n_0$. Therefore, $\frac{f}{g} = \mathcal{O}(1)$. □

Question 2

Consider the following functions:

a) $\left(\frac{1}{2}\right)^n + 3n^{17}$

b) $2^{n+\log_2 n}$

c) 2^{4n}

d) $\sqrt{n^2+n^4}$

e) $\frac{n^8-n^7}{24}$

1. For each of the above functions $f(n)$, find a "canonical" function $g(n)$ such that $f(n) = \Theta(g(n))$. By canonical we mean that g should be of the form $g(n) = a^n n^b \log^c n$ for constants $a \geq 1$ and $b, c \geq 0$. For example, $3200n + 2n^2 \log^{24}(n^2) = \Theta(n^2 \log^{24} n)$.

Solution:

(a) $\left(\frac{1}{2}\right)^n + 3n^{17} = \Theta(n^{17})$. As $n \to \infty$, we have $\left(\frac{1}{2}\right)^n \to 0$ and $3n^{17} \to \infty$. We have the asymptotically dominant term $3n^{17}$; ignoring constants gives $\Theta(n^{17})$.

(b) $2^{n+\log_2 n} = \Theta(n \cdot 2^n)$. We have $2^{n+\log_2 n} = 2^n \cdot 2^{\log_2 n} = n \cdot 2^n$.

(c) $2^{4n} = \Theta(16^n)$. We have $2^{4n} = (2^4)^n = 16^n$.

(d) $\sqrt{n^2+n^4} = \Theta(n^2)$. As $n \to \infty$, the n^4 term dominates n^2. Ignoring the dominated term, we have $\sqrt{n^4} = n^2$, giving $\Theta(n^2)$.

(e) $\frac{n^8-n^7}{24} = \Theta(n^8)$. As $n \to \infty$, the n^8 term dominates n^7; ignoring constants gives $\Theta(n^8)$.

2. Based on your answers from above, sort the functions above in asymptotically increasing order. Are there any two functions with the same order of growth? If yes, which ones?

Solution: In increasing order of asymptotic growth, we have (d), (e), (a), (b), and (c). None of the functions have the same order of growth. We know (b) is greater than (c) because $\lim_{n\to\infty} \frac{16^n}{n \cdot 2^n}$ is unbounded.

Question 3

In this problem, we will solve the following recurrence using the **substitution method** (i.e., induction). Assume $T(1) = 0,\ T(2) = 1$ and that the recurrences below define $T(n)$ for $n > 2$:

$$T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + 1\ . \tag{1}$$

You do not need to worry about the $n/3$ and $2n/3$ terms in (1) being fractions.

1. Try $T(n) = \sqrt{n}$. Does the recurrence hold? You may ignore the base cases ($n = 1$ and 2) and only look at whether $T(n)$ satisfies Eq. (1). Is the RHS bigger than the LHS? Show your calculations.
Counterexample. Consider $T(n) = \sqrt{n}$.

$$\begin{aligned}\sqrt{n} &\stackrel{?}{=} \sqrt{\frac{n}{3}} + \sqrt{\frac{2n}{3}} + 1 \\ &\stackrel{?}{=} \frac{1+\sqrt{2}}{3}\sqrt{n} + 1 \\ &< \frac{1+\sqrt{2}}{3}\sqrt{n} + 1.\end{aligned}$$

Taking $n = 3$ as a counterexample, we have $2 + \sqrt{2} \neq \sqrt{3}$.

The right-hand side is greater than the left-hand side, so the recurrence does not hold. □

2. Try $T(n) = n$. Does the recurrence hold? You may ignore the base cases ($n = 1$ and 2) and only look at whether $T(n)$ satisfies Eq. (1). Is the RHS bigger than the LHS? Show your calculations.
Counterexample. Consider $T(n) = n$.

$$\begin{aligned}n &\stackrel{?}{=} \left(\frac{n}{3}\right) + \left(\frac{2n}{3}\right) + 1 \\ n &\stackrel{?}{=} n + 1 \\ n &< n + 1.\end{aligned}$$

Taking $n = 3$ as a counterexample, we have $3 < 4$.

The right-hand side is greater than the left-hand side, so the recurrence does not hold. □

3. Try $T(n) = n^2$. Does the recurrence hold? You may ignore the base cases ($n = 1$ and 2) and only look at whether $T(n)$ satisfies Eq. (1). Is the RHS bigger than the LHS? Show your calculations.
Counterexample. Consider $T(n) = n^2$.

$$\begin{aligned}n^2 &\stackrel{?}{=} \left(\frac{n}{3}\right)^2 + \left(\frac{2n}{3}\right)^2 + 1 \\ n^2 &\stackrel{?}{=} \frac{5}{9}n^2 + 1 \\ n^2 &> \frac{5}{9}n^2 + 1.\end{aligned}$$

Taking $n = 3$ as a counterexample, we have $9 > 6$.

The left-hand side is greater than the right-hand side, so the recurrence does not hold. □

Among parts 1 to 3, which choice of $T(n)$ "almost" satisfied recurrence (1)? That is, for which choice of $T(n)$ was the left-hand side *almost equal* to the right-hand side of (1)? Reflect on this question because it would help solve the next part.

Among parts 1 to 3, the choice $T(n) = n$ almost satisfies the recurrence.

4. Let $T(n) = n^p + c$ for some $c \in \mathbb{R}$, and $p > 0$ and solve for p and c. Based on the p and c you obtained, prove your result formally using the substitution method. (Note that parts 1-3 correspond to $c = 0$ and $p = 1/2, p = 1, p = 2$, respectively.)

Proof. Let $T(n) = n^p + c$ for $c \in \mathbb{R}$ and $p > 0$. We will show that $T(n) = n - 1$ satisfies the recurrence (that is, we will demonstrate that the recurrence holds for $c = -1$ and $p = 1$). Observe

$$\begin{aligned} T(n) &= T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + 1 \\ &= \left(\frac{n}{3} - 1\right) + \left(\frac{2n}{3} - 1\right) + 1 \\ &= n - 1 \\ &= n^1 + (-1). \end{aligned}$$

Therefore $T(n) = n - 1$ satisfies the recurrence. □

Question 4: Merge sort

Recall Merge Sort, in which a list is sorted by first sorting the left and right halves, and then merging the two lists. We define the *3-Merge Sort* algorithm, in which the input list is split into 3 equal length parts (or as equal as possible), each is sorted recursively, and then the three lists are merged to create a final sorted list.

1. Write a recurrence for $T(n)$, the worst-case run time for 3-Merge Sort on any input containing n elements.

   Let $f(n)$ represent the worst-case running time of the merge operation with an n-element input array, where $f(n) = \Theta(n)$. We can write a recurrence for $T(n)$, the worst-case running time for 3-merge sort with an n-element input array:

   $$T(1) = 1,$$

   $$T(2) = 1,$$

   $$T(n) = 3T\left(\frac{n}{3}\right) + f(n).$$

2. Solve this recurrence for $T(n)$. Prove your result formally using the substitution method (i.e., induction). For full credit, you'll need to show matching upper and lower bounds.

   Solution.

   Let $f(n)$ represent the worst-case running time of the merge operation with an n-element input array, where $f(n) = \Theta(n)$. We can write a recurrence for $T(n)$, the worst-case running time for 3-merge sort with an n-element input array.

   Based on our understanding of merge sort, we can guess that $T(n) = \Theta(n \log n)$.

   Lemma I.

   Claim. $T(n) = O(n \log n)$ for $n \geq 3$.

   We want to show that there exists a positive constant c for which $T(n) \leq cn \log n$, and we can prove this result for $n \geq 3$ using induction on n.

   Basis. Consider $n = 3$. We have $T(3) = f(3) = \Theta(1)$. We can choose c such that $f(3) \leq 3c \log(3)$. There exists a positive constant c where $c \geq \frac{f(3)}{3 \log(3)}$. Therefore the argument holds in the basis case.

   Hypothesis. Consider $3 < n \leq k$. Assume $T(k) \leq ck \log k$.

   Inductive step. Consider $n = k + 1$. Observe

   $$\begin{aligned} T(k) &= 3T\left(\frac{k+1}{3}\right) + f(k+1) \\ &= 3T\left(\frac{k+1}{3}\right) + \Theta(k+1). \end{aligned}$$

Since $\frac{k+1}{3} < k$ for all $k > 3$, the strong induction hypothesis holds for $n = \frac{k+1}{3}$.

$$\begin{aligned} T(k) &\leq 3c\left(\frac{k+1}{3}\right)\log\left(\frac{k+1}{3}\right) + \Theta(k+1) \\ &\leq 3c\left(\frac{k+1}{3}\right)\log\left(\frac{k+1}{3}\right) + c'(k+1) \\ &\leq c(k+1)(\log(k+1) - \log(3)) + c'(k+1) \\ &\leq c(k+1)\log(k+1) - c(k+1)\log(3) + c'(k+1) \\ &\leq c(k+1)\log(k+1) - (k+1)(c\log(3) - c'). \end{aligned}$$

Choose $c\log(3) - c' \geq 0$ such that c satisfies the basis case. That is, choose c such that $c \geq \frac{f(3)}{3\log(3)}$ and $c \geq \frac{c'}{\log(3)}$. For $n = k+1$, there exists a positive constant c for which $T(k+1) \leq c(k+1)\log(k+1)$. Hence, $T(n) = O(n\log n)$ for $n \geq 3$, by the principle of mathematical induction.

Lemma II.

Claim. $T(n) = \Omega(n\log n)$ for $n \geq 3$.

Similarly, we want to show that there exists a positive constant c for which $T(n) \geq cn\log n$, and we can prove this result for $n \geq 3$ using induction on n.

Basis. Consider $n = 3$. We can choose c such that $T(3) \geq 3c\log(3)$. We have $T(3) = f(3) = \Theta(1)$. We can choose c such that $f(3) \geq 3c\log(3)$. There exists a positive constant c where $c \leq \frac{f(3)}{3\log(3)}$. Therefore the argument holds in the basis case.

Hypothesis. Consider $3 < n \leq k$. Assume $T(k) \geq ck\log k$.

Inductive step. Consider $n = k+1$. Observe

$$\begin{aligned} T(k) &= 3T\left(\frac{k+1}{3}\right) + f(k+1) \\ &= 3T\left(\frac{k+1}{3}\right) + \Theta(k+1). \end{aligned}$$

Since $\frac{k+1}{3} < k$ for all $k > 3$, the strong induction hypothesis holds for $n = \frac{k+1}{3}$.

$$\begin{aligned} T(k) &\geq 3c\left(\frac{k+1}{3}\right)\log\left(\frac{k+1}{3}\right) + \Theta(k+1) \\ &\geq 3c\left(\frac{k+1}{3}\right)\log\left(\frac{k+1}{3}\right) + c'(k+1) \\ &\geq c(k+1)(\log(k+1) - \log(3)) + c'(k+1) \\ &\geq c(k+1)\log(k+1) - c(k+1)\log(3) + c'(k+1) \\ &\geq c(k+1)\log(k+1) - (k+1)(c\log(3) - c'). \end{aligned}$$

Choose $c\log(3) - c' \leq 0$. That is, choose c such that $c \leq \frac{c'}{\log(3)}$ and $c \leq \frac{f(3)}{3\log(3)}$. For $n = k+1$, there exists a positive constant c for which $T(k+1) \leq c(k+1)\log(k+1)$.

Hence, $T(n) = \Omega(n \log n)$ for $n \geq 3$, by the principle of mathematical induction.

Proof.

Claim. $T(n) = \Theta(n \log n)$.

From Lemma I, we have $T(n) = O(n \log n)$ for $n \geq 3$.

From Lemma II, we have $T(n) = \Omega(n \log n)$ for $n \geq 3$.

Ergo $T(n) = \Theta(n \log n)$ for $n \geq 3$. $\square$

Honors Question 1

(**) Solve the following recurrences (assume $T(1) = T(2) = 1$ and that the recurrences below define $T(n)$ for $n > 2$). In all cases, you must prove the correctness of your solution. It is good enough to get an answer $f(n)$ such that $T(n) = \Theta(f(n))$.

1. $T(n) = 2T(\lceil\sqrt{n}\rceil) + \log_2 n$.

   Claim. $T(n) = 2T(\lceil\sqrt{n}\rceil) + \log_2 n = \Theta(\log n \log^2 n)$.

   Proof. Let $S(n) = T(2^n)$ for $n > 0$. Observe

   $$\begin{aligned} S(n) &= T(2^n) \\ &= 2T\left\lceil\sqrt{2^n}\right\rceil + \log_2 2^n \\ &= 2T\left\lceil\sqrt{2^n}\right\rceil + n \\ &= 2S\left(\frac{n}{2}\right) + n. \end{aligned}$$

   We can apply the master theorem for divide-and-conquer recurrences. We have a recurrence S of the form $S(n) = 2S\left(\frac{n}{2}\right) + n$. The critical exponent is $c^* = \log_2(2) = 1$. Note $n = \Theta(n^{c^*} \log_2^0 n)$. Therefore, Case II of the master theorem gives

   $$S(n) = \Theta(n^{c^*} \log_2^{0+1} n) = \Theta(n \log_2 n).$$

   By construction, $S(n) = T(2^n)$, therefore

   $$\begin{aligned} T(n) &= S(\log_2 n) \\ &= \Theta(\log_2 n \log_2 \log_2 n) \\ &= \Theta(\log_2 n \log_2^2 n). \end{aligned}$$

   Ergo $T(n) = \Theta(\log n \log^2 n)$. $\square$

2. $T(n) = \lceil\sqrt{n}\rceil\, T(\lceil\sqrt{n}\rceil) + n$.

   Claim. $T(n) = \lceil\sqrt{n}\rceil\, T(\lceil\sqrt{n}\rceil) + n = \Theta(n \log\log n)$.

   Proof. Let $S(n) = T(2^n)$ for $n > 0$. Observe

   $$\begin{aligned} S(n) &= T(2^n) \\ &= \left\lceil\sqrt{2^n}\right\rceil T\left(\left\lceil\sqrt{2^n}\right\rceil\right) + 2^n \\ &= 2^{\frac{n}{2}} T\left(2^{\frac{n}{2}}\right) + 2^n \\ \frac{S(n)}{2^n} &= \frac{2^{\frac{n}{2}} T\left(2^{\frac{n}{2}}\right)}{2^n} + 1. \\ &= \frac{T\left(2^{\frac{n}{2}}\right)}{2^{\frac{n}{2}}} + 1. \end{aligned}$$

Let $R(n) = \frac{S(n)}{2^n} = \frac{T\left(2^{\frac{n}{2}}\right)}{2^{\frac{n}{2}}} + 1$ for $n > 0$. Observe

$$R(n) = R\left(\frac{n}{2}\right) + 1.$$

We can apply the master theorem for divide-and-conquer recurrences. We have a recurrence R of the form $R(n) = 1 \cdot R\left(\frac{n}{2}\right) + 1$. The critical exponent is $c^* = \log_2(1) = 0$. Note $1 = \Theta(n^{c^*} \log_1^0 n)$. Therefore, Case II of the master theorem gives

$$R(n) = \Theta(n^{c^*} \log_2^{0+1} n) = \Theta(\log_2 n).$$

By construction, $R(n) = \frac{S(n)}{2^n}$, therefore $S(n) = 2^n R(n)$. By construction $S(n) = T(2^n)$, therefore

$$\begin{aligned} T(n) &= 2^{\log_2 n} R(\log_2 n) \\ &= nR(\log_2 n) \\ &= n\Theta(\log_2 n) \\ &= \Theta(n \log_2 \log_2 n). \end{aligned}$$

Ergo $T(n) = \Theta(n \log \log n)$. $\square$