Homework 1 Solutions
Fundamental Algorithms, Spring 2025, Professor Yap, Section Leader Dr. Bingwei Zhang

Due: Thu Feb 6, 2025 (Upload to GradeScope by 11:30pm)
HOMEWORK with SOLUTION

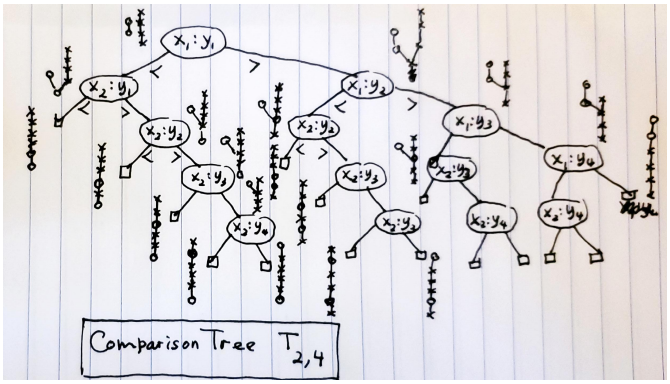
INSTRUCTIONS:

- We have a "no late homework" policy.
  Special permission must be obtained *in advance* if you have a valid reason.
- The exercises in this homework come from Chapters I and II (i.e., `l1-class.pdf` and `l2-class.pdf`).
- If you are not familiar with Gradescope, please try to practice uploading your solutions in advance. You can resubmit as many times as you like.

(Q1) (12 Points)
Exercise I.4.12, page 20.
Drawing the comparison tree $T_{2,4}$.

> **THE QUESTION** ("Unrolling" or "unwinding" a uniform algorithm into a comparison tree)
> (a) Draw the comparison tree $T_{2,4}$ obtained by unrolling our Merge Algorithm on input (x_1, x_2) and (y_1, y_2, y_3, y_4).
> (b) Draw the Hasse diagram associated with each node of the tree in Part (a).

SOLUTION FIGURE:



> **SOLUTION:** We follow the execution of the Merge Algorithm: the first comparison is $x_1 : y_1$. So the root of our tree program has the comparison $x_1 : y_1$. If $x_1 < y_1$, then the next comparison is $x_2 : y_1$. So the left child of the root has the comparison $x_2 : y_1$. Likewise, the right child of the root has the comparison $x_1 : y_2$. We proceed in this way to draw the entire comparison tree.
> **Comments:** This problem illustrates the difference between a uniform model of computation and a non-uniform model (like comparison trees).

> **SOLUTION:** We follow the execution of the Merge Algorithm on this input: the first comparison is $x_1 : y_1$. So the root of our tree program has the comparison $x_1 : y_1$. If $x_1 < y_1$, then the next comparison is $x_2 : y_1$. Thus the left child of the root has the comparison $x_2 : y_1$. Likewise, the right child of the root has the comparison $x_1 : y_2$. You can proceed in this way to draw the entire comparison tree.
> **Comments:** This problem illustrates the difference between a uniform model of computation and a

non-uniform model (like comparison trees).

(Q2) (10+10 Points)
Exercise I.5.5, page 27.
Bounds on $S(1000)$.

THE QUESTION The exact value of $S(1000)$ is unknown, so we seek to derive upper and lower bounds on $S(1000)$. For upper bound, please bound the complexity of any $O(n \log n)$ algorithm, say, MergeSort.

NOTE: we are asking for two numbers. State the two values and tell us how you obtain them. Your numbers must be *explicit* (in decimal notation like 1234), not an expression like $1000\lceil \lg 1000 \rceil$. You may use computer programs or calculators, etc, but tell us how you do it. If you use computers, discuss how you can be confident despite rounding errors in the computation. Stirling's formula in the text is useful to know.

SOLUTION We claim: $S(1000) \geqslant 8530$ and $S(1000) \leqslant 8977$.
JUSTIFICATION: The lower bound is given by the ITB: $S(1000) \geqslant \lg(1000!)$ (must be base 2). But we need an explicit number, not an expression like "$\lg(1000!)$". You can use a calculator or write a program (any language you like) to compute the numerical value of $\lg(1000!)$.
It seems best to use form of Stirling's formula found in my Lecture II: $n! = (n/e)^n \sqrt{2\pi n} e^{\alpha(n)}$ where $\frac{1}{12n+1} < \alpha(n) < \frac{1}{12n}$. Taking natural logs, I obtain

$$\begin{array}{lcl} \lg(n!) & = & n(\lg n - \lg e) + \frac{1}{2}\lg(2\pi n) + \alpha(n). \\ \lg(1000!) & \geqslant & 8523.0892 + 6.3086 + 0.00012 \\ & \geqslant & 8529.3979 \end{array}$$

Since I want a lower bound, I can truncate the value I get on my calculator $S(1000) \geqslant 8529$. However, because $S(1000)$ is an integer, I can actually round it up: $S(1000) \geqslant 8530$.
Justifying the upper bound on $S(1000)$. It must come from some algorithm. We know that the Mergesort algorithm makes at most the following number of comparisons, assuming n is a power of 2 (so when we divide by 2, we get another power of 2, etc):

$$T(n) = 2T(n/2) + n - 1. \tag{1}$$

But this only works when n is even. We can use the exact recurrence

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n - 1. \tag{2}$$

I wrote a Java program to evaluate (2) recursively and obtain

$$S(1000) \leqslant T(1000) = 8977.$$

Can you improve on upper bound of 8977? Use the fact that for $n \leqslant 31$, $S(n) = \lceil \lg(n!) \rceil$.
What if you want to cruder upper bound that can be done by hand? Pick n to be a power of two that is larger than 1000, say $n = 2^{10} = 1024$. If $n = 2^k$, this has an easily computed bound (replacing $n - 1$ by n in (1)):

$$T(n) = 2^k T(n/2^k) + kn$$

where $k = \lg n$. Since $T(n/2^k) = T(1) = 0$, we obtain the following (crude) upper bound:

$$\begin{array}{rcl} S(1000) & < & S(1024) = S(2^{10}) \\ & \leqslant & 10 \cdot 2^{10} = 10240. \end{array}$$

Be careful of some pitfalls.
(1) First of all, you must not confuse lg with ln or $\log_{10}$ (on calculators).
(2) If you tried to compute the factorial 1000! before taking logs, your value will overflow because 1000! needs over 8000 bits (and most calculators or computer programs uses 64 bits or at most 128 bits in their calculation.
(3) To get the upper bound, you can use any sorting algorithm, but to get full credit, use an $O(n \log n)$ algorithm. But it is not so simple – there is a hidden constant in the big-Oh notation. So you need to think through how to get the upper bound.
Another way to do a crude hand computation, as a student in our class did, is to use the "lazy" Mergesort recurrence $T(n) = 2T(\lceil n/2 \rceil) + n - 1$:

$$S(1000) \leqslant 2S(500) + 999 \leqslant 4S(250) + 999 + 2(499) \leqslant 8S(125) + 999 + 2(499) + 4(249) \leqslant \cdots \leqslant 9105.$$

This is much better upper bound than $S(1024) \leqslant 10240$ but not as good as $S(1000) \leqslant 8977$.

(Q3) (10 Points)
Exercise I.8.6, page 42.
Why is there is no "small-theta" notation?

THE QUESTION Our asymptotic notations fall under two groups: O, Ω, Θ and o, ω. In the first group, we have $\Theta(f) = O(f) \cap \Omega(f)$. This suggests the "small-theta" analogue for the second group, "$\theta(f) = o(f) \cap \omega(f)$". Why was this not done?

SOLUTION: Recall that $o(f)$ comprises those g such that for all $C > 0$, $Cf \gtrsim g \gtrsim 0$ (ev.). Likewise $\omega(f)$ comprise those g such that for all $Cg \gtrsim f \gtrsim 0$ (ev.). This means that if $g \in o(f) \cap \omega(f)$, then

$$(\forall C > 0)[Cf \gtrsim g \gtrsim f/C \text{ (ev.)}].$$

Now, if f is the 0 constant, then this statement implies $g = 0$ (ev.). But such g's are essentially equal to the 0 constant, and not very interesting. If f is non-zero *infinitely often*, then this statement can never be satisfied by any g. THE ONLY EXCEPTION is when $g = \uparrow$ (ev.) (eventually undefined). But this is NOT a uninteresting class of functions. Thus $o(f) \cap \omega(f)$ is not an interesting concept.

(Q4) (6×8 Points)
Exercise I.8.15, Parts (a)-(h), page 44.

THE QUESTION True or false: please provide a counter-example when false, and a proof when true. The base b of logarithms is arbitrary but fixed, and $b > 1$. Assume the functions f, g are arbitrary. Do not assume that f and g are $\gtrsim 0$ eventually – the follow-up question next make assume additional properties of f, g.

(a) $f = \mathcal{O}(g)$ implies $g = \mathcal{O}(f)$.
(b) $\max\{f, g\} = \Theta(f + g)$.
(c) If $g > 1$ and $f = \mathcal{O}(g)$ then $\ln f = \mathcal{O}(\ln g)$.
(d) $f = \mathcal{O}(g)$ implies $f \circ \log = \mathcal{O}(g \circ \log)$. Assume that $g \circ \log$ and $f \circ \log$ are complexity functions.
(e) $f = \mathcal{O}(g)$ implies $2^f = \mathcal{O}(2^g)$.
(f) $f = o(g)$ implies $2^f = \mathcal{O}(2^g)$.
(g) $f = \mathcal{O}(f^2)$.
(h) $f(n) = \Theta(f(n/2))$.

SOLUTION: Only two statements are true:

(a) False. $f = n$ and $g = n^2$.

(b) False. Take f any function that is eventually non-zero. Then take $g = -f$

(c) False. Take $f = 1/2$ and any function g such that $g > 1$. But $\log(f) < 0$ and so, by definition of the $\mathcal{O}$-notation, $f \notin \mathcal{O}(\lg g)$).

This solution takes advantage of a technical requirement in the definition of $\mathcal{O}$-notation. HERE is another solution which does not exploit this property: Let $f = 2$ and $g(x) = (x+1)/x = 1+(1/x)$. Then $\ln g > 0$ but $\lg g(n) \to 0$ as $x \to \infty$. [In fact, $\lg g(x) < 1/(2x)$, but you don't need to not know this]. Clearly, $\lg f = 1$ but there is no constant $C > 0$ such that $\lg f = 1 \leqslant C \lg g$.

(d) True. We have $f = \mathcal{O}(g)$ implies there is some $C > 0$ and x_0 such that for all $x > x_0$, $f(x) \leqslant Cg(x)$. Thus, $f(\log(x)) \leqslant Cg(\log(x))$ for all $x \geqslant e^{x_0}$.

(e) False. Let $f = 2n$ and $g = n$.

(f) True. $f = o(g)$ implies that for all $C > 0$, $0 \leqslant f \leqslant C * g$ (ev). Taking $C = 1$, we obtain that $0 \leqslant 2^f \leqslant 2^g$ or in other words $2^f \in \mathcal{O}(2^g)$.

(g) False. Let $f = 1/n$.

(h) False. $f = 2^n$.

Note: f=1/x is unbounded, but not unbounded eventually!

(Q5) (10 Points)
Exercise II.4.4, page 18.
Karatsuba recurrence and rote.

THE QUESTION Solve the Karatsuba recurrence (21) using the Rote Method. NOTE: do not forget to verify your inductive formula!

SOLUTION:

$$
\begin{aligned}
T(n) &= n + 3T(n/2) && \text{(Expand once)}\\
&= (n + (3n/2)) + 3^2T(n/2^2) && \text{(Expand twice, simplify)}\\
&\vdots\\
&= n\left(\textstyle\sum_{j=0}^{i-1}(3/2)^j\right) + 3^iT(n/2^i) && \text{(Guessed formula for } i \text{ expansions)}\\
&= n\left(\textstyle\sum_{j=0}^{i-1}(3/2)^j\right) + 3^i\boxed{3T(n/2^{i+1}) + n/2^i} && \text{(Expand } i+1 \text{ times)}\\
&= n\left(\textstyle\sum_{j=0}^{i}(3/2)^j\right) + 3^{i+1}T(n/2^{i+1}) && \text{(Simplify, and formula verified!)}\\
&= n\left(\textstyle\sum_{j=0}^{i_0}(3/2)^j\right) && \text{(stop at } = i_0 = \lfloor \lg n \rfloor\text{; by DIC, } T(n/2^{\lg n}) = 0 \text{ if } i > i_0)\\
&= n\Theta\left((3/2)^{\lg n}\right) && \text{(geometric sum formula)}\\
&= \Theta\left(3^{\lg n}\right)\\
&= \Theta\left(n^{\lg 3}\right).
\end{aligned}
$$

(Q6) (8+8+8+8+(6+8+10) Points)
Exercise II.7.3, page 45.
Growth types, parts (a)–(e).

THE QUESTION

Proving growth types of functions:

(a) Let $f(x) = 4x^2 - 10x$. Prove that $f(x)$ is polynomial-type.

Note that you must show two implicit constants, $C > 1$ and $x_0 > 0$. For full credit, *choose the smallest integer value of C, and subject to this, the smallest value of x_0.*

(b) Let $f(x) = 2^x - 10x$. Prove that $f(x)$ is exponentially increasing-type.
By definition, you must show 3 constants $C > 1$, $k > 0$ and $x_0 > 0$ such that

$$(\forall x > x_0)[f(x) > C \cdot f(x-k)] \tag{3}$$

Please choose $k = 1$, then choose C to be the smallest possible integer, and then choose x_0 to be the smallest possible integer.

(c) Let $f(x) = 2^x/(x!)$. Prove that $f(x)$ is exponentially decreasing-type.
Similar to part(b), we need 3 constants $C > 1$, $k > 0$ and $x_0 > 0$. Again, please set $k = 1$ and choose smallest possible integer values of C and x_0.

(d) Prove that polynomial-types are closed under multiplication: i.e., if $f(x) \geqslant 0$ and $g(x) \geqslant 0$ are polynomial-types, so is $h(x) := f(x)g(x)$.

(e) Please state the Θ-order of the following sums:

(i) $\sum_{i\geqslant 1}^{n} i^7 \log^3 i$

(ii) $\sum_{i\geqslant 1}^{n} i^2 \log^{-1} i$

(iii) $\sum_{i\geqslant 1}^{n} \frac{2^i}{i^5 \log i}$

Justify your derivations – you may quote any facts in these lecture notes (please be specific!). Otherwise you must prove it yourself!

SOLUTION: (a) We optimal choice is $C = 5$ and $x_0 = 15$. We must show

$$\begin{aligned}
f(x) &\leqslant C \cdot f(x/2) \\
&\textit{i.e.,} \\
4x^2 - 10x &\leqslant C \cdot [4(x/2)^2 - 10(x/2)] \\
4x^2 - 10x &\leqslant C \cdot [x^2 - 5x)] \\
4x - 10 &\leqslant C \cdot [x - 5] \qquad \text{(divide by } x > 0\text{)} \\
5C - 10 &\leqslant (C-4)x.
\end{aligned}$$

Choosing $C = 5$:

$$\begin{aligned}
25 - 10 &\leqslant x \\
15 &\leqslant x
\end{aligned}$$

The last inequality would hold if we choose $x_0 = 15$.
Comments: Notice how we delay the choices of C and x_0 until the algebraic manipulations tell us what is needed! Clearly, we could choose ANY C that is strictly larger than 4, say $C = 4.01$. But as $C - 4$ gets smaller and smaller, we are forced to larger and larger values of x_0. But if C is integer, then $C = 5$ is determined. The smallest value of x_0 follows.

SOLUTION: (b) Claim: $k = 1$, $C = 1.5$ and $x_0 = 3$ will verify (3).
Using the hint, let $k = 1$. So we must show for $x > x_0$, $f(x) > C \cdot f(x-1)$, i.e.,

$$\begin{aligned}
2^x - 10x &> C[2^{x-1} - 10(x-1)] \\
2^x - C2^{x-1} &> 10x - C[10(x-1)] \\
2^x(1 - \tfrac{1}{2}C) &> 10x(1-C) + 10C
\end{aligned}$$

We are now forced to choose $C = 1$:

$$\begin{aligned}
2^{x-1} &> 10 \\
2^x &> 20
\end{aligned}$$

The smallest integer value of x_0 is now determined to be $x_0 = 5$.

SOLUTION: (c) This amounts to showing that

$$(\exists C > 1)(\exists k > 0)(\exists x_0 > 0)(\forall x > x_0)[f(x) \leqslant f(x-k)/C] \tag{4}$$

CLAIM: We can choose $k = 1$, $C = 2$ and $x_0 = 4$. For $k = 1$, the constants C, x_0 must satisfy $(\forall x > x_0)[f(x) \leqslant f(x-1)/C]$

$$\frac{2^x}{x!} \leqslant \frac{2^{x-1}}{C(x-1)!}$$
$$2 \leqslant \frac{x}{C}$$

The smallest integer C is $C = 1$; then the smallest integer x_0 is $x_0 = 2$.

SOLUTION: (d) First note that $h(x)$ is non-decreasing (ev.): this follows from

$$\begin{array}{rcl} h(x) = f(x)g(x) & \leqslant & f(x')g(x') \\ & = & h(x'), \text{for } x \leqslant x' (\text{ev.}). \end{array}$$

We must then show that h is not growing too fast,

$$h(x) \leqslant C_h \cdot h(x/2)$$

for some $C > 1$ eventually. But we know that

$$h(x) = f(x)g(x) \leqslant (C_f \cdot f(x/2))(C_g \cdot g(x/2))$$

for $x \geqslant \max\{x_f, x_g\}$. Thus we can choose $C_h = C_f \cdot C_g$ and $x_h = \max\{x_f, x_g\}$.

SOLUTION: (e)

(i) $\sum_{i \geqslant 1}^{n} n^7 \log^3 n = \Theta(n^8 \log^3 n)$
Justification: n^7 and $\log^3 n$ are polynomial-types, so is their product.

(ii) $\sum_{i \geqslant 1}^{n} n^2 \log^{-1} n = \Theta(n^2 \log^{-1} n)$
Justification: The function $f(n) = n^2/\log n$ is polynomial-type. But nothing we have seen so far shows this fact! So we show it directly:

$$\begin{array}{rcll} f(n) = \frac{n^2}{\log n} & \leqslant & K\frac{(n/2)^2}{\log n} & \text{provided we choose } K \geqslant 4 \\ & < & K\frac{(n/2)^2}{\log(n/2)} & \\ & = & Kf(n/2). & \end{array}$$

(iii) $\sum_{i \geqslant 1}^{n} \frac{2^i}{i^5 \log i} = \Theta(\frac{2^n}{n^5 \log n})$
Justification: The function $f(n) = \frac{2^n}{n^5 \log n}$ is exponential increasing-type. But nothing we have seen so far shows this fact. So we must show it directly.
I will show that $f(n) > 2f(n-7)$ (ev.) using elementary arguments: but to simplify the details, let us assume that $\log n$ is actually $\ln n$ (natural log) in this problem.

(A) First, we claim

$$\ln n - \ln(n-k) < k/(n-k), \qquad (n > k \geqslant 1). \tag{5}$$

In proof:

$$\ln n - \ln(n-k) = \ln\left(\frac{n}{n-k}\right) = \ln\left(1 + \frac{k}{n-k}\right) < k/(n-k)$$

(the last inequality comes from $\ln(1+x) < x$).

(B) Second, we claim

$$k/(n-k) \leqslant \ln(n-k) \tag{6}$$

when n is big enough relative to k. We use the fact[a] that $H_n = \ln n + g(n)$ for some $0 < g(n) < 1$. Thus $\ln n > H_{n-k} - 1 > \frac{1}{2}$ (provided $n - k \geqslant 2$). But $\frac{1}{2} \geqslant k/(n-k)$ provided $n \geqslant 3k$. This proves (6).

(C) Third, we claim

$$\tfrac{1}{2}\ln n \leqslant \ln(n-7) \tag{7}$$

This is equivalent to $\ln n \leqslant 2\ln(n-7)$, and to $\ln n - \ln(n-7) \leqslant \ln(n-7)$. The last inequality follows from $\ln n - \ln(n-7) < 7/(n-7)$ (by (5)), and $7/(n-7) \leqslant \ln(n-7)$ (by (6)).

(D) Finally we prove that $f(n)$ is exponential increasing:

$$\begin{aligned} f(n) = \tfrac{2^n}{n^5 \ln n} &= 2^7 \tfrac{2^{n-7}}{n^5 \ln n} \\ &= 2\tfrac{2^{n-7}}{(n/2)^5(\frac{1}{2}\ln n)}) \\ &> 2\tfrac{2^{n-7}}{(n-7)^5(\frac{1}{2}\ln n)} \quad \text{(since } (n/2) < n-7 \text{ for } n > 14\text{)} \\ &\geqslant 2\tfrac{2^{n-7}}{(n-7)^5 \ln(n-7)} \quad \text{(by (7))} \\ &= 2f(n-7). \end{aligned}$$

[a] E.g., Chapter 2, §6 (Basic Sums) in my algorithms book; download through our wiki Schedule Page.

(Q7) ((6× 6)× 2 Points)
Exercise II.14.2, page 78.
Simplifying expressions, parts (a)–(f). **Do both Versions (i) and (ii)**.

THE QUESTION Simplify the following expressions (two versions):
Version (i): Please show your simplification steps.

(a) $n^{1/\lg n}$, (b) $2^{2^{(\lg\lg n)-1}}$, (c) $\sum_{i=0}^{k-1} 2^i$,
(d) $2^{(\lg n)^2}$, (e) $4^{\lg n}$, (f) $(\sqrt{2})^{\lg n}$.

Version (ii): Re-do version (i), replacing each explicit or implicit occurrence of "2" in the previous expressions by a constant $c > 1$. We view lg as $\log_2$, 4 as 2^2 and $\sqrt{n}$ as $n^{1/2}$. Thus these expressions becomes $\log_c$, c^c and $n^{1/c}$.

SOLUTION: Version (i):
(a) 2. Pf: $n^{1/\lg n} = (2^{\lg n})^{1/\lg n} = 2^1$.
(b) $\sqrt{n}$. Pf: $2^{2^{(\lg\lg n)-1}} = 2^{(2^{\lg\lg n})(2^{-1})} = 2^{(\lg n^{\lg 2})(1/2)} = 2^{(\lg n)(1/2)} = n^{1/2} = \sqrt{n}$.
(c) $2^k - 1$. Pf: geometric series.
(d) $n^{\lg n}$. Pf: $2^{(\lg n)^2} = (2^{\lg n})^{\lg n} = n^{\lg n}$. [In what sense is this a simplification?]
(e) n^2. Pf: $4^{\lg n} = 2^{2\lg n} = (2^{\lg n})^2 = n^2$.
(f) $\sqrt{n}$. Pf: $(\sqrt{2})^{\lg n} = 2^{(1/2)(\lg n)} = n^{1/2} = \sqrt{n}$.

SOLUTION: Version (ii):
(a) c
(b) $n^{1/c}$
(c) $(c^k - 1)/(c-1)$
(d) The expression to simplify is $c^{(\log_c n)^c}$. This simplification is rather special, but you can imitate the argument in (i)(d) above:

$$c^{\log_c n^c} = c^{(\log_c n)(\log_c n)^{c-1}} = (c^{\log_c n})^{(\log_c n)^{c-1}} = n^{(\log_c n)^{c-1}}.$$

NOTE: Beware of this standard mistake: a^{b^c} means $a^{(b^c)}$, NOT $(a^b)^c$. Some students made this mistake

to give the wrong answer

$$c^{\log_c n^c} = (c^{\log_c n})^c = n^c.$$

(e) The question might be ambiguous: if 4 is not viewed as 2^2, then $4^{\log_c n} = n^{\log_c 4}$. But we asked you to view 4 as 2^2. So the answer is $(c^c)^{\log_c n} = (c^{\log_c n})^c = n^c$.
(f) View $\sqrt{2}$ as $2^{1/2}$ and so the expression to simplify is $(c^{1/c})^{\log_c n} = (c^{\log_c n})^{1/c} = n^{1/c}$ (which may be written $\sqrt[c]{n}$).