Name:_____

Date: February _____, 2018

CS320: Complexity and Logic

Print these pages, **staple** them, and fill in your answers.

Bounding complexities[1]

The worst case complexity of an algorithm (the maximum number of basic operations it might perform) is a function of the size of its input. So when comparing the complexity of algorithms, we are comparing functions. If we have two functions, $f(n)\ g(n)$, we can determine their relationship as their input sizes grow by looking at the ratio $f(n) / g(n)$.

For example, $f(n) = 5n,\ \ g(n) = 2n,\ \ f(n)/g(n) = 5/2$. The ratio of the two functions does *not* depend on the input size. So the algorithm whose complexity is $f(n)$ requires 2.5 times more operations than the algorithm for $g(n)$ no matter the size of n. Their complexities are within a constant factor of each other, even as n grows toward infinity.

Another example, $f(n) = nlog_2 n,\ \ g(n) = n,\ \ f(n)/g(n) = log_2 n$. Here function $f(n)$ grows faster. The "gap" between them gets larger and larger as n grows. The "gap" in the complexities for the two algorithms with these complexities grows logarithmically with the input. As n grows toward infinity, so does the gap, and $f(n)$ dwarfs $g(n)$.

The relationships between functions we want to identify are:

- $f(n) = O(g(n))$ meaning that $g(n)$ is an asymptotic *upper* bound on $f(n)$, within a constant factor. It is read as "f of n is order g of n" or "f is big-oh of g of n".
- $f(n) = \Omega(g(n))$ meaning that $g(n)$ is an asymptotic *lower* bound on $f(n)$, within a constant factor. It is read as "f is big-omega of g of n".
- $f(n) = \Theta(g(n))$ meaning that $g(n)$ is an asymptotically tight bound on $f(n)$, bounding it above and below within a constant factor. It is read as "f is big-theta of g of n". If $f(n) = O(g(n))$ *and* $f(n) = \Omega(g(n())$, then $f(n) = \Theta(g(n))$. Hence if $f(n) = \Theta(g(n))$, then there exist positive constants $c_1,\ c_2,\ n_0$ such that for all $n \geq n_0 :\ \ c_1 \leq \frac{f(n)}{g(n)} \leq c_2$

Additionally[2] there are:

- $f(n) = o(g(n))$ meaning that $g(n)$ is an upper bound on $f(n)$ that is *not* asymptotically tight. It is read as "f is little-oh of g of n". For example, $2n = o(n^2)$, but $2n^2 \neq o(n^2)$. Where $O(g(n))$ is analogous to $\leq$, $o(g(n))$ is analogous to $<$. Hence if $f(n) = o(g(n))$, then $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.
- $f(n) = \omega(g(n))$ meaning that $g(n)$ is a lower bound on $f(n)$ that is *not* asymptotically tight. It is read as "f is little-omega of g of n". For example, $\frac{n^2}{2} = \omega(n)$, but $\frac{n^2}{2} \neq \omega(n^2)$. Where

[1] Read more details in Cormen Ch 3

[2] Some functions are not asymptotically comparable at all

$\Omega(g(n))$ is analogous to $\geq$, $\omega(g(n))$ is analogous to $>$. Hence if $f(n) = \omega(g(n))$, then $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$.

For the following questions, some **blanks should be filled** with one of the five symbols to indicate the relationship between two functions. In situations where more than one of these symbols accurately describes their relationship, there are more than one blank.

1. If $f(n) = O(g(n))$, then $g(n) =$ $(f(n))$
2. If $f(n) = o(g(n))$, then $g(n) =$ $(f(n))$
3. Since $f(n) = \Theta(f(n))$, it is also true that $f(n) =$ $(f(n))$ and $f(n) =$ $(f(n))$
4. If $f(n) = nlog_2 n$ $g(n) = n$, then $f(n)/g(n) =$, which means that $f(n) =$ $(g(n))$ *and* $f(n) =$ $(g(n))$
5. If $f(n) = n^{1/2}$ $g(n) = 75\sqrt{n}$, then $f(n)/g(n) =$, which means that $f(n) =$ $(g(n))$

Facts about exponents:

- $x^{-y} = 1/x^y$
- $(x^y)^z = x^{yz} = (x^z)^y$
- $x^y/x^z = x^{y-z}$ and $x^y x^z = x^{y+z}$
- $\lim_{n\to\infty} n^y = \infty$ if $y > 0$
- $\lim_{n\to\infty} \frac{n^y}{logn} = \infty$ if $y > 0$
    - The polynomial may start slow, but will overtake the log eventually. See the approximate solutions for $x^{0.1} = log_2 x$ (try changing it to $x^{0.01}$)

6. If $f(n) = n^{1.5}$ $g(n) = n^{1.6}$, then $f(n)/g(n) =$, which means that $f(n) =$ $(g(n))$ *and* $f(n) =$ $(g(n))$
7. If $f(n) = n^{1.2}$ $g(n) = nlog_2 n$, then $f(n)/g(n) =$, which means that $f(n) =$ $(g(n))$ *and* $f(n) =$ $(g(n))$
8. If $f(n) = (log_2 n)^2$ $g(n) = n^{0.001} log_2 n$, then $f(n)/g(n) =$, which means that $f(n) =$ $(g(n))$ *and* $f(n) =$ $(g(n))$
9. If $f(n) = n^2$ $g(n) = (\sqrt{n})^4$, then $f(n)/g(n) =$, which means that $f(n) =$ $(g(n))$

Facts about exponential functions:

- $\lim_{n\to\infty} \frac{c^n}{n^d} = \infty$, if c and d are constants such that $c > 1,\ d > 0$
    - Check out the solutions to $1.1^x = x^2$, $1.01^x = x^2$, $1.001^x = x^2$
- $\frac{c^n}{d^n} = \left(\frac{c}{d}\right)^n$

10. If $f(n) = 3^n$ $g(n) = n^3$, then $f(n) =$ $(g(n))$ *and* $f(n) =$ $(g(n))$
11. If $f(n) = 2^n$ $g(n) = 3^n$, then $f(n)/g(n) =$, which means that $f(n) =$ $(g(n))$ *and* $f(n) =$ $(g(n))$

Fact about comparing sums of functions:

- If $h(n) = \omega(j(n))$, then $\lim_{n\to\infty} \frac{h(n)+j(n)}{g(n)} = \lim_{n\to\infty} \frac{h(n)}{g(n)}$
- Essentially, we can ignore all but the largest term
- If $f(n) = 3n^2 + 9n + 4 \quad g(n) = n^2$, then $f(n)/g(n) = 3 + 9/n + 6/n^2$. As $n \to \infty$, those last two terms approach 0. Check out the graph, try changing the 9 to 900. See the limit for $f(n)/g(n)$ and $g(n)/f(n)$.

12. If $f(n) = \sum_{i=1}^{n} i \quad g(n) = n^2$, then $f(n)/g(n) =$, which means that $f(n) =$ $(g(n))$
13. If $f(n) = \sum_{i=0}^{n} 2^i \quad g(n) = 2^n$, then $f(n)/g(n) =$, which means that $f(n) =$ $(g(n))$

---

a. Hint 1: what would the binary representation of the values of $f(n)$ look like?
b. Hint 2: try punching the summation into wolfram alpha

Fact about logarithm bases:

- $log_b x = \frac{log_c x}{log_c b}$
- Check out these log functions of different bases and their relationships. Notice that their ratio is constant. Contrast that with exponentials, where the bases do matter.

14. If $f(n) = nlog_2 n \quad g(n) = nlog_8 n$, then $f(n)/g(n) =$, which means that $f(n) =$ $(g(n))$
15. If $f(n) = n! \quad g(n) = (n+1)!$, then $f(n)/g(n) =$, which means that $f(n) =$ $(g(n))$ *and* $f(n) =$ $(g(n))$

Greedy algorithms[3]

Algorithms using a greedy strategy try to solve problems with optimal substructure by making locally optimal choices that lead to a globally optimal solution.

Activity Selection

Remember that the activity selection problem takes a collection of start and end times for each activity in a group, and seeks to find a maximum sized subset of them that do not conflict with each other in time. We find that choosing the activity with the earliest finishing time, ignoring the conflicting activities, and recursing on what's left will always lead to an optimal solution. We can avoid recursion by operating on a list of activities, sorted by earliest finishing time, moving from left to right. If we have:

i	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
start	1	3	0	5	3	5	6	8	8	2	12
finish	4	5	6	7	9	9	10	11	12	14	16

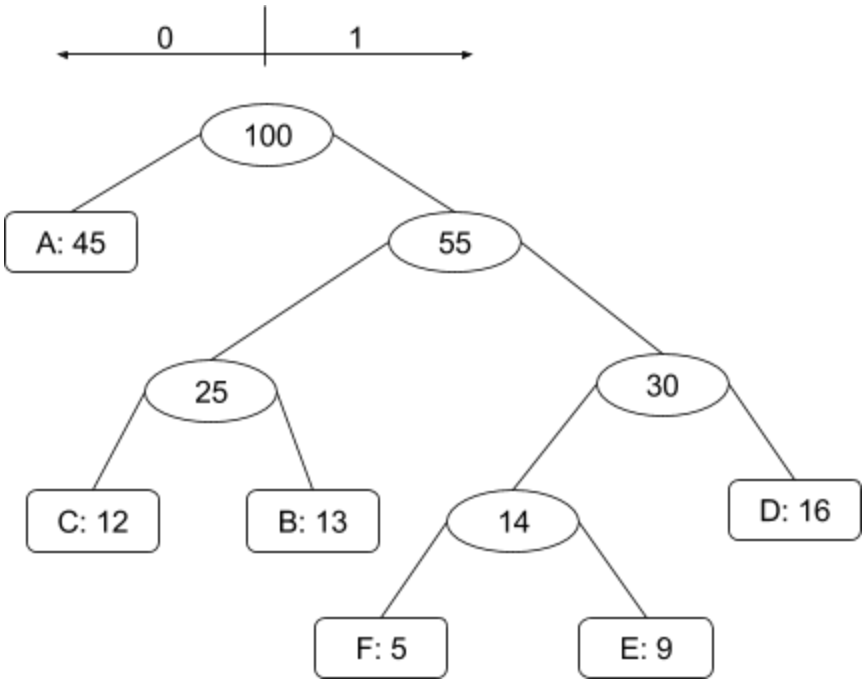
[3] Read more details in Cormen Ch 16

We would start by choosing activity 1 for our solution. That precludes activities 2 and 3 because they start before activity 1 finishes. Activity 4 has the next earliest finishing time of the activities that don't conflict. We'll choose it too.

1. Find the remaining activities that would be chosen by this algorithm:
   _____

Huffman Codes

When choosing a binary representation for data to be stored or transmitted across a network, the size of the resulting message is a consideration. A Huffman code works by assigning shorter bit sequences to represent more frequent symbols (characters or words typically), and assigning longer bit sequences to rarer symbols. When storing/sending data, we use the assigned bit code in place of the symbols' normal representation. And when loading/receiving data, we use a lookup tree to find the symbol that corresponds to each bit code. A tree will look like:



When decoding, you would start at the root, and take a left if you see a 0, and a right if you see a 1. So the code 100 corresponds to a "C", while a 0 corresponds to an "A". We construct this tree using a greedy algorithm that makes its choice based on the frequencies of nodes in a priority queue. The frequencies are the numbers next to the letters. Also, the ovals (non-leaves) have a frequency which is simply the sum of the frequencies of their children.

The algorithm goes like this. Put all symbols in a min-priority queue by their frequency. Then in a loop, until there is only one item in the queue, extract the two lowest items and make them the children of a new internal node whose frequency is the sum of the children's. Then put this root of a new tree into the queue. In the tree above, F then E were the two lowest frequency items in the first round. They were joined into a 14 weight node. Then C and B were joined into a 25

weight node. Then the combined 14 node and D were joined into the 30 weight node, and so on.

1. These frequencies depend on the type of the data, and even on the content of an individual message. Take the string “badbanana”, using characters as symbols, and create a frequency chart:

a	b	d	n
			2

2. Next, write out these four original nodes and start joining them with internal nodes according to the Huffman algorithm.

3. You should now have a bit pattern determined for each of the four symbols, based on how to navigate to that node from the root. How many bits does the original message take to represent with your Huffman code? _____

Logic and Reasoning

For many of the following questions, you should draw a truth table to help you in reasoning about how they work. For example, if the problem asks about implication relationships, we should draw a truth table describing the relationship among the input boolean variables, let’s say A and B, and the boolean expression $A => B$ (A implies B). You might draw it like this:

A	B	A => B
T	T	T
T	F	F

F	T	T
F	F	T

This truth table reminds us that the implication is false only when A is true and B is false. In particular, if the premise (A) is false, the implication is true no matter what the value of B.

Recall this about logic: Suppose there is no life on Mars. Then the statement: "All Martians are green" is a true statement. Any statement in logic is true if no counterexample can exist. If there is no example of it either, it is what is called vacuously true.

1. Suppose you know that A ⇒ B is ***true*** and that A is ***true***. What do these facts tell you about the value of B, if anything?

2. Suppose you know that A ⇒ B is ***true*** and that A is ***false***. What do these facts tell you about the value of B, if anything?

3. Suppose you know that A ⇒ B is true and B is false. What do these facts tell you about the value of A, if anything?

4. Tell **whether the following implication** statement is true **given** each of the following scenarios, or state that it can't be determined. The implication statement is:
   (All politicians are dishonest) ⇒ (The country is in trouble)
   The scenarios follow, consider each separately:
   a. *All politicians are dishonest* is a true statement.

   b. *There exists an honest politician* is a true statement

   c. *The country is in trouble* is a true statement.

5. Suppose: *(All politicians are dishonest) ⇒ (The country is in trouble)* is a true statement, but the country is in trouble is a false statement. Precisely what do these facts imply about politicians?

6. Draw out a full truth table in the space below to solve for the values of A and B, given the following statements are true: !A ⇒ B, !B ⇒ !C, C. Below the truth table, state what we know about the values of A and B after completing the table.

7. Here are some statements related to the book's proof of correctness of Gale-Shapley.

- A: The set returned by every execution G-S is a stable matching.
- B: The set returned by an execution of G-S is never a stable matching.
- C: A set S returned by some execution of G-S is not a stable matching.
- D: In S, there exist pairs (m,w) and (m',w') such that m prefers w' to w and w' prefers m to m'
- E: In S, there exist pairs (m,w) and (m',w') such that m prefers w' to w and w prefers m' to m.
- F: w' rejects m, and traded up to higher and higher men in her list, culminating in m'.

Which of the following best reflects the logical structure of the proof?

a. Using logic, it shows that C⇒D, D⇒F and F⇒!(D) are true statements, without yet resolving the values of C, D, and F. ((D⇒F) AND (F⇒!(D)) ⇒ !(D) is true. Therefore !(D) is true. Since C⇒D, !(D)⇒!(C), so !(C) is true. Because $C \Leftrightarrow \ !A$, A is true.

b. Using logic, it shows that B⇒D, D⇒F and F⇒!(D) are true statements, without yet resolving the values of B, D, and F. ((D⇒F)AND(F⇒!(D))⇒!(D)is true. B=!(A), so A is true.

c. Using logic, it shows that C⇒D, D⇒E, and E⇒!(D) are true statements, without yet resolving the values of C, D, and E. ((D⇒E) AND (E⇒!(D)) ⇒!(D) is true. Therefore !(D) is true. Since C⇒D, !(D)⇒!(C), so !(C) is true. C=!(A), so A is true.

8. In the stable matching problem, let a pair of lovebirds be a man and a woman who are each first on the other's list. Let a pair of last resorts denote a man and a woman that are each last on the other's list.

For each of the following, either prove that the statement is true or prove that it is false. To prove a statement is false, come up with a counterexample. (For such a proof, it is desirable to

find the smallest counterexample you can think of.) If you want to prove the statement true, use proof by contradiction: assume that it is false and derive a contradiction. You should not use an overly formal style here, but do write a logical and coherent line of reasoning.

a. Claim: No stable matching contains a last resort.

b. Claim: Every stable matching contains a pair of lovebirds. A pair of lovebirds is two people, not two couples.

c. Claim: Whenever there is a pair of lovebirds. They are married in every stable matching. (If you would like to prove this true, do **not** refer to the mechanics of the Gale-Shapley algorithm. It has nothing to do with the definition of stable matchings or lovebirds, and it is not needed here.)

Fantastic work!