CSCI 3104 Spring 2017 Problem Set 2
Matthew Niemiec (06/24)
Mehmet Karaoglu (07/30)

---

1. *(15 pts total) Prove or disprove each of the following claims, where f(n) and g(n) are functions on positive values.*

   (a) $f(n) = O(g(n))$ *implies* $g(n) = \Omega(f(n))$
   The definition of O is that there exists a scalar constant c such that $0 \leq cf(n) \leq g(n)$ for all $n \geq n_0$. Given that the functions are non-negative, we can say that $f(n) = O(g(n))$ means that $cf(n) \leq g(n)$ and $g(n) = \Omega(f(n))$ means that $f(n) \geq g(n)$. In this case, if $f(n) = O(g(n))$, then $cf(n) \leq g(n)$ and $(\frac{1}{c})g(n) \geq f(n)$, and by definition, $g(n) = \Omega(f(n))$ since $\frac{1}{c}$ is still a constant.

   (b) $f(n) = O(g(n))$ *implies* $2^{f(n)} = O(2^{g(n)})$
   The statement is not true. Take $f(n) = 2n$ and $g(n) = n$. $2n = \Theta(n)$ and therefore $2n = O(n)$. However

$$\lim_{n\to\infty} \frac{2^n}{2^{2n}} = 0 \tag{1}$$

   which is 0, so $2^{2n} \neq O(2^n)$

   (c) $f(n) = O((f(n))^2)$
   This one is also false. Take $f(n) = \frac{1}{n}$

$$\lim_{n\to\infty} \frac{\frac{1}{n^2}}{\frac{1}{n}} = \frac{n}{n^2} = \frac{1}{n} = 0 \tag{2}$$

   so it is clear that $\frac{1}{n} \neq O(\frac{1}{n^{(2)}})$ since $\frac{1}{n}$ is asymptotically larger than $\frac{1}{n^2}$.

2. *(10 pts total) Consider the following Python function:*

```
def find_max (L):
    max = 0
    for x in L:
        if x > max:
            max = x
    return max
```

Suppose list L has n elements.

(a) *In asymptotic notation, determine the best case running time as function of n.*

The best case run-time happens when the max number is the first number in the array because the line $max = x$ will only run once. Initialization is constant, and there are 2 atomic operations within the loop that runs $n = L.length$ times, so it will have $2n + 4$ operations (2 in the loop, initialization of L, $max = 0$, $max = x$, and $return\ max$), so it has $O(n)$ runtime.

(b) *In asymptotic notation, determine the worst case running time as function of n.*

The worst case is when the array is sorted, because then the line $max = x$ will run n times, causing the array to have $3n + 3$ atomic operations, so it will still have $O(n)$ runtime. Thus we can say that the algorithm has $\Theta(n)$ runtime.

(c) *Now, assume L is sorted. Give an algorithm that takes asymptotically less time than the above algorithm, but performs the same function. Prove that your algorithm is correct, and explain why this algorithm is useless.*

The most efficient algorithm provided that the algorithm is sorted is

```
def find_max (L):
    return L[L.length]
```

As one can see, there are only 3 atomic operations (set L, find L[L.length], and return). Thus, it has $O(1)$ runtime. Unfortunately this algorithm is useless because it takes at best $nlog(n)$ time to sort an array, and since sorted arrays don't just appear, it is much faster to take the $\Omega(n)$ time needed to find the maximum value without sorting.

3. *(30 pts total) Solve the following recurrence relations using the method specified. Show your work.*

   (a) $T(n) = T(n-2) + n$ *by unrolling (tail recursion)*

$$\begin{aligned} T(n) &= T(n-2) + n \\ &= T(n-4) + (n-2) + n \\ &= T(n-6) + (n-4) + (n-2) + n \\ &= O(1) + (n-n) + (n-n+2) + ... + (n-6) + (n-4) + (n-2) + n \\ &= O(1) + \sum_{i=0}^{n/2} n - 2i \\ &= O(1) + n \cdot \left(\frac{n}{2} + 1\right) - 2(n/2)(n/2+1)/2 \\ &= O(1) + n^2/2 + n - n^2/4 - n/2 \\ &= O(1) + n^2/4 + n/2 \\ &= O(1) + \Theta(n^2) \\ &= \Theta(n^2) \end{aligned}$$

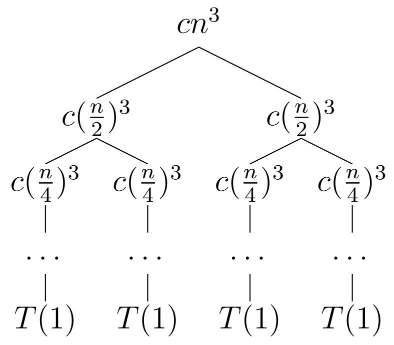
   (b) $T(n) = 2T(\frac{n}{2}) + n^3$ *by the Master method*

   There exists an epsilon $\epsilon = 1$ such that $n^3 = \Omega(n^{log_2 2+\epsilon}) = \Omega(n^2)$ because $\lim_{n\to\infty} \frac{n^3}{n^2} = \infty$. Our recurrence also satisfies $af(\frac{n}{b}) \leq cf(n)$ for $a = 2, b = 2$ and $c = \frac{1}{2}$ and $f(n) = n^3$. $(\frac{n^3}{4}) \leq (\frac{n^3}{2})$ for all $n \geq 0$.

   Our recurrence satisfies every condition for case 3 of the master method, so we can conclude that $T(n) = \Theta(f(n)) = \Theta(n^3)$

   (c) $T(n) = 2T(\frac{n}{2}) + n^3$ *by the recurrence tree method; include the tree diagram.*

   $T(n)$ here consists of the time taken to solve two smaller sub-problems $(2T(\frac{n}{2}))$ as well as the time taken to divide each problem and combine the parts back together (n^3). We can represent the recurrence by using the following tree diagram:



At the top level of recursion, we have only one node that has a running time of a constant c times n^3. Each problem thereafter gets split into two sub-problems that have half as many elements. This recursion continues until we reach the trivial case where $n = 1$. Each level i of the tree has 2^i nodes and each node at level i has size $\frac{n}{2^i}$. The bottom level is $i = \log_2 n$ and it contains $2^{\log_2 n} = n^{\log_2 2} = n$ nodes. The total running time of the last level is $nT(1)$, which is $\Theta(n)$ since $T(1)$ is a constant. The running time at level i is $2^i c(\frac{n}{2^i})^3 = c(\frac{n}{4^i})^3$.

$$\begin{aligned}
T(n) &= cn^3 + (\frac{1}{4})cn^3 + (\frac{1}{4})^2cn^3 + \ldots + (\frac{1}{4})^{\log_2 n-1}cn^3 + \Theta(n) \\
&= \sum_{i=0}^{\log_2 n-1} (\frac{1}{4})^i cn^3 + \Theta(n) \;<\; \sum_{i=0}^{\infty}(\frac{1}{4})^i cn^3 + \Theta(n) \\
&= (\frac{1}{1-\frac{1}{4}})cn^3 + \Theta(n) \;=\; \frac{4}{3}cn^3 + \Theta(n) \;=\; \mathcal{O}(n^3)
\end{aligned}$$

An infinite decreasing geometric series is used to give an upper bound to $T(n)$ and we have found that $O(n^3)$ is an upper bound for $T(n)$. However, we can say that $\Omega(n^3)$ is also a lower bound for $T(n)$ because we know the first recurrence already contributes $\Theta(n^3)$ to the entire running time.

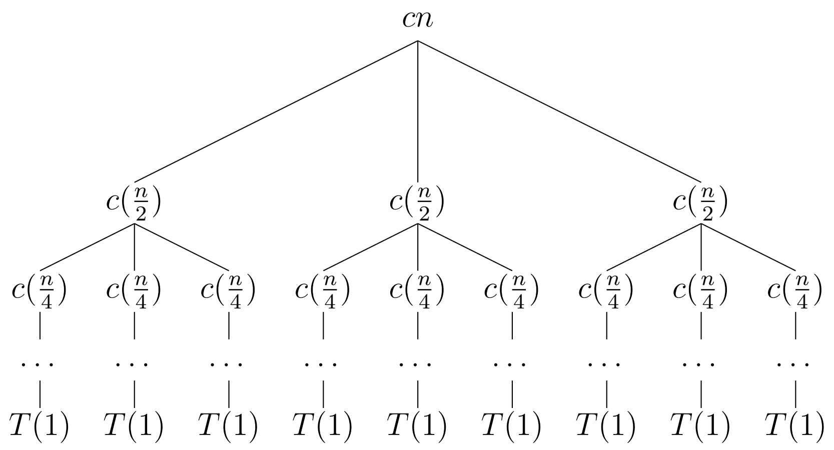
Thus, $T(n) = \Theta(n^3)$.

(d) $T(n) = 2T(\frac{n}{4}) + \sqrt{n}$ *by the Master method.*

Here, $a = 2$ and $b = 4$. We have $n^{log_4 2} = n^2$. $n^{1/2} = O(n^{2-\frac{3}{2}})$. We have an $\epsilon > 0$ so $T(n) = \Theta(n^2)$

(e) *$T(n) = 3T(\frac{n}{2}) + n$ by the recurrence tree method; include the tree diagram.*

For this problem, $T(n)$ consists of the time taken to solve three smaller subproblems that each have a size of $\frac{n}{2}$ and a linear factor n. The tree diagram for this recurrence is as follows:



At level i of the tree, there are 3^i nodes which have a size of $\frac{n}{2^i}$. The running time of each level is $3^i\frac{cn}{2^i}$. The last level is $i = \log_2 n$ and it has $3^{\log_2 n} = n^{\log_2 3}$ nodes, each with a running time of $T(1)$ because there is only one element contained in each node at this level. The running time of the bottom level is given by $T(1)n^{\log_2 3}$ which is $\Theta(n^{\log_2 3})$ because $T(1)$ is a constant. We can equate $T(n)$ as follows:

$$\begin{aligned} T(n) &= cn + (\frac{3}{2})cn + (\frac{3}{2})^2 cn + \ldots + (\frac{3}{2})^{\log_2 n - 1} cn + \Theta(n^{log_2 3}) \\ &= \sum_{i=0}^{\log_2 n-1} (\frac{3}{2})^i cn + \Theta(n^{log_2 3}) \ = \ \frac{\frac{3}{2}^{(\log_2 n-1)+1} - 1}{\frac{3}{2} - \frac{2}{2}} + \Theta(n^{log_2 3}) \\ &= 2(\frac{3}{2}^{log_2 n} - 1) + \Theta(n^{log_2 3}) \ = \ 3\log_3 n - 2 + \Theta(n^{log_2 3}) \end{aligned}$$

Getting rid of the lower order term $(3\log_3 n - 2)$, we're left with $T(n) = \Theta(n^{log_2 3})$

4. *(15 pts) Exercise* $4.5-4$ *in CLRS*

In this recurrence, $a = 4$ and $b = 2$. When we try to apply the master method, we see that $n^{log_b a} = n^{log_2 4} = n^2$. $n^2 log_2 n \neq O(n^{2-\epsilon})$. When we look at $\lim_{x\to\infty} \frac{n^2 log_2 n}{n^{2-\epsilon}}$, we can see that we cannot find an $\epsilon > 0$ such that the limit will converge to zero. So that means n^2 is asymptotically larger but not polynomicaly larger than $n^2 log_2 n$. Also, we know that $n^2 log_2 n \neq \Omega(n^{2+\epsilon})$ because there doesn't exist an $\epsilon > 0$ such that $\lim_{x\to\infty} \frac{n^{2+\epsilon}}{n^2 log_2 n}$ will converge to zero. These indicate that our recurrence doesn't fit the first and third (and second) rules for the master method, which means we can't use it.

We can solve this recurrence with a recurrence tree. We know that each level i in this tree will have 4^i nodes and each level will take $c(\frac{n}{2^i})^2 log_2(\frac{n}{2^i})$ running time. Using these, we can say the following:

$$\begin{aligned}
T(n) &= \sum_{i=0}^{\log_2 n} c4^i(\frac{n}{2^i})^2 \log_2(\frac{n}{2^i}) \\
&= \sum_{i=0}^{\log_2 n} cn^2 \log_2(\frac{n}{2^i}) \\
&= cn^2 \sum_{i=0}^{\log_2 n} log_2 n - log_2 2^i \\
&= cn^2(\sum_{i=0}^{\log_2 n} log_2 n - \sum_{i=0}^{\log_2 n} \log_2 2^i) \\
&= cn^2(\log_2 n \log_2 n - \frac{\log_2 n(log_2 n + 1)}{2}) \qquad = cn^2(\frac{(\log_2 n)^2}{2} - cn^2\frac{log_2 n}{2}) \\
&\leq cn^2(log_2 n)^2 \\
&= c(nlog_2 n)^2 \\
&= \Theta(nlog_2 n)^2
\end{aligned}$$

5. *(30 pts total) Professor Snape has n magical widgets that are supposedly both identical and capable of testing each others correctness. Snape's test apparatus can hold two widgets at a time. When it is loaded, each widget tests the other and reports whether it is good or bad. A good widget always reports accurately whether the other widget is good or bad, but the answer of a bad widget cannot be trusted. Thus, the four possible outcomes of a test are as follows:*

Widget A says	Widget B says	Conclusion
B is good	A is good	both are good, or both are bad
B is good	A is bad	at least one is bad
B is bad	A is good	at least one is bad
B is bad	A is bad	at least one is bad

(a) *Prove that if $n/2$ or more widgets are bad, Snape cannot necessarily determine which widgets are good using any strategy based on this kind of pairwise test. Assume a worst-case scenario in which the bad widgets are intelligent and conspire to fool Snape.*

If there are an even number of widgets, and half were bad, then half could for a group validating each other and saying the good widgets are bad. Of course, the good widgets would do the same thing, so you wouldn't be able to tell which group of $n/2$ widgets are actually good or bad. If there were an odd number and $(n-1)/2$ were good, then there could be two groups of $(n-1)/2$ widgets that confirm each other, and it doesn't really matter what the last bot does. However, if $(n+1)/2$ bots are good or more, then there will be a group of at least $(n+1)/2$ bots that confirm each other, and because that's more than half, there can't be another group of equal size that does the same.

(b) *(Divide) Consider the problem of finding a single good widget from among the n widgets, assuming that more than $n/2$ of the widgets are good. Prove that $n/2$ pairwise tests are sufficient to reduce the problem to one of nearly half the size.*

In order to find one good widget, we pair up each of the widgets, and then compare them. Since we know that more than $n/2$ of the widgets are good, then we know from the pigeon-hole principle that there will be at least one good-good pair, which actually is good. If we get only one of these pairs, then we know that each of the rest of them are all good-bad pairs, so we choose one from the good-good pair and we have a widget. In the worst case there will be an odd number, so we can't pair up every widget, and every pair is good-good. In that case we know that each of the widgets in the pairs are of the same identity, then we can eliminate $ceil(n/2)$ widgets and begin the process again until we have just one good-good pair.

(c) *(And Conquer!) Prove that the good widgets can be identified with $\Theta(n)$ pairwise tests, assuming that more than $n/2$ of the widgets are good. Give and solve the recurrence that describes the number of tests.*

Let's say it takes $\Theta(1)$ time to compare the widgets. Then for every round we are making $n/2$ (or $(n-1)/2$, which is the same asymptotically) comparisons. We also know that at each stage of the process, we reduce the problem by $n/2$. Finally, at each stage, we make just 1 recursive call deeper. Thus, by the Master Theorem, our recurrence relation is $T(n) = (1)T(n/2) + \frac{n}{2}$. Now let $a = 1$, $b = 2$, and $f(n) = \frac{n}{2}$. Now moving to the Master Theorem we see that $n^{log_b a} = n^{log_2 1} = n^0 = 1$. Thus we can readily see that $n/2 = \Omega(1)$. Also note that $af(n/b) \leq c(f(n)) \rightarrow n/4 \leq n/2$. Thus $T(n) = \Theta(f(n)) = \Theta(n/2) = \Theta(n)$.

6. *(15 pts extra credit)*

Asymptotic relations like O, Ω, and Θ represent relationships between functions, and these relationships are transitive. That is, if some f(n) $= \Omega(g(n))$, and $g(n) = \Omega(h(n))$, then it is also true that $f(n) = \Omega(h(n))$. This means that we can sort functions by their asymptotic growth.

Sort the following functions by order of asymptotic growth such that the final arrangement of functions $g_1, g_2, ..., g_{12}$ satisfies the ordering constraint $g_1 = \Omega(g_2), g_2 = \Omega(g_3), ..., g_{11} = \Omega(g_{12})$.

n	n^2	$(\sqrt{2})^{\lg n}$	$2^{(lg^* n)}$	$n!$	$(lg(n))!$	$(\frac{3}{2})^n$	$n \lg n$	$lg(n!)$	e^n	1

Give the final sorted list and identify which pair(s) functions $f(n), g(n)$, if any, are in the same equivalence class, i.e., $f(n) = \Theta(g(n))$.

Here is the list:

$n!$	e^n	$(\frac{3}{2})^n$	$lg(n)!$	n^2	$n \lg n$	$lg(n!)$	n	$(\sqrt{2})^{\lg n}$	$2^{(lg^* n)}$	$n^{1/\lg n}$	1

The limit cancels out because as n gets larger, they far outweigh the constant values of e below

$$\lim_{n\to\infty} \frac{n!}{e^n} = \lim_{n\to\infty} \frac{n(n-1)\cdots 1}{e * e * \cdots * e} = \infty \tag{3}$$

$$\tag{4}$$

For the next comparison all we need is a simple limit

$$\lim_{n\to\infty} \frac{e^n}{(1.5)^n} = \lim_{n\to\infty} (\frac{e}{1.5})^n \geq \lim_{n\to\infty} 1.218^n = \infty \tag{5}$$

$$\tag{6}$$

We can apply L'Hopital's rule to the following limit:

$$\lim_{n\to\infty} \frac{1.5^n}{n^2} = \lim_{n\to\infty} \frac{\frac{d}{dn}1.5^n}{\frac{d}{dn}n^2} = \lim_{n\to\infty} \frac{ln(1.5)1.5^n}{2n} = \tag{7}$$

$$\lim_{n\to\infty} \frac{\frac{d}{dn}ln(1.5)1.5^n}{\frac{d}{dn}2n} = \lim_{n\to\infty} \frac{ln(1.5)^2 1.5^n}{2} = \infty \tag{8}$$

$$\tag{9}$$

It is a common fact that $n^2 = O(n\lg(n))$, but here's the limit:

$$\lim_{n\to\infty} \frac{n^2}{n\lg(n)} = \lim_{n\to\infty} \frac{n}{\lg(n)} = \infty \tag{10}$$

$$\tag{11}$$

We can see that

$$\lg(n!) = \lg(n(n-1)(n-2)\cdots 1) = \lg(n) + \lg(n-1) + \lg(n-2) + \cdots + \lg(1) \tag{12}$$

Even though $\lg(n-k) \leq \lg(n)$, $\lg(n-k) = \Omega(\lg(n)$, so in the limit

$$\lg(n!) = n\lg(n) = \Theta(n\lg(n)) \tag{13}$$

$$\tag{14}$$

Using exponential properties of logarithms, we can deduce that

$$(\sqrt{2})^{lg(n)} = 2^{0.5\lg(n)} = 2^{\lg(\sqrt{n})} = \sqrt{n} = \Theta(\sqrt{n}) \tag{15}$$

Also note that $\sqrt{2}^{\lg(n)}$ is $O(n\lg(n))$.

Here is a chart comparing n and $2^{\lg * n}$

n	$2^{\lg * n}$
2	2
4	4
16	8
65536	16

So we can see that for every time 2^{lg*n} doubles, n is raised to the $n-1$ power. Thus we can see that the function will never be linear, but it also is not constant.

We know from properties of logarithms that

$$\lim_{n\to\infty} n^{1/\lg(n)} = \lim_{n\to\infty} 2^{\lg(n^{1/lg(n)})} = \lim_{n\to\infty} 2^{lg(n)/lg(n)} = 2^1 = 2 = \Theta(1) \tag{16}$$