CS 457, Data Structures and Algorithms I
First Problem Set Solutions

October 13, 2019

1. (16 pts) Find two functions $f(n)$ and $g(n)$ that satisfy the following relationship. If no such f and g exist, then try to explain why this is the case.

   a) $f(n) = o(g(n))$ and $f(n) \notin \Theta(g(n))$

      Solution: $f(n) = n$ **and** $g(n) = n^2$

   b) $f(n) = \Theta(g(n))$ and $f(n) = o(g(n))$

      Solution: None exist. If $f(n) = \Theta(g(n))$ then $g(n) = \Theta(f(n))$. Thus, there exists some constant c' and some constant n_0 for which $g(n) \leq c' f(n) \quad \forall n > n_0$. However, $f(n) = o(g(n))$ implies that for all constants c there exists a constant n_0 for which $f(n) < cg(n) \quad \forall n > n_0$. Allowing $c = c'$ shows that these two statements are contradictory.

   c) $f(n) = \Theta(g(n))$ and $f(n) \notin O(g(n))$

      Solution: None exist. $f(n) = \Theta(g(n)) \Leftrightarrow f(n) = O(g(n)) \wedge f(n) = \Omega(g(n))$. It is impossible to have the left hand side be true without implying the right hand side.

   d) $f(n) = \Omega(g(n))$ and $f(n) \notin O(g(n))$

      Solution: $f(n) = n^2$ **and** $g(n) = n$

2. (16 pts) For each of the following questions, briefly explain your answer.

   a) If I prove that an algorithm takes $O(n^2)$ worst-case time, is it possible that it takes $O(n)$ on some inputs?

      Solution: Yes, this is exactly what we proved in class with insertion sort. Insertion sort is $O(n^2)$ for the worst-case input but $O(n)$ for the best-case input.

   b) If I prove that an algorithm takes $O(n^2)$ worst-case time, is it possible that it takes $O(n)$ time on all inputs?

Solution: Yes, this is possible. Recall that $O(n^2)$ is not a tight upper bound. Without proving the lower bound that the algorithm takes $\Omega(n^2)$ worst-case time, it is possible that the true tight bound is lower than $O(n^2)$.

c) If I prove that an algorithm takes $\Theta(n^2)$ worst-case time, is it possible that it takes $O(n)$ time on some inputs?

Solution: Yes, this is exactly what we proved in class for insertion sort. The worst-case time for insertion sort is $\Theta(n^2)$ but the best-case time is $\Theta(n)$ implying that there is some input that the algorithm takes $O(n)$ time for.

d) If I prove that an algorithm takes $\Theta(n^2)$ worst-case time, is it possible that it takes $O(n)$ time on all inputs?

Solution: No, this is not possible. Recall that a time bound defined as $\Theta(f(n))$ is a tight bound. Therefore, this implies that, for the worst-case input, the algorithm has a lower (and upper) bound of n^2 on the time. Thus, for the worst-case input, the upper bound of $O(n)$ is not correct. More succinctly, in the worst-case time we know the algorithm is $\Theta(n^2) \Rightarrow \Omega(n^2)$. $\Omega(n^2)$ implies that a time bound of $O(n)$ is invalid for the worst-case input.

3. (10 pts) Let $f(n)$ and $g(n)$ be any two functions with $f(n) > 1$ and $g(n) > 1$ for all n. Is it true that $15f(n)^2 + 23g(n)^2 = \Theta((f(n) + g(n))^2)$? Give a formal justification for your answer.

Solution: The statement is true. To show this, recall that we need to find c_1, c_2, and n_0 such that for all $n > n_0$, $0 \le c_1(f(n) + g(n))^2 \le 15f(n)^2 + 23g(n)^2 \le c_2(f(n) + g(n))^2$. We first will show the right hand inequality. We may expand $(f(n) + g(n))^2$ as $f(n)^2 + g(n)^2 + 2f(n)g(n)$. Since both $f(n)$ and $g(n)$ are positive, it must be the case that $15f(n)^2 + 23g(n)^2 \le 23(f(n)^2 + g(n)^2 + 2f(n)g(n))$. Therefore, the right hand inequality holds for $c_2 = 23$ and $n_0 = 1$. We now move to show the left hand inequality. Observe that if $f(n) \ge g(n)$ then we have $2f(n)g(n) \le 2f(n)^2$. Similarly, if $g(n) \ge f(n)$ then we would have $2f(n)g(n) \le 2g(n)^2$. Therefore, it must be true that $2f(n)g(n) \le 2f(n)^2 + 2g(n)^2$. From this we may conclude that $(f(n) + g(n))^2 \le 3f(n)^2 + 3g(n)^2$. This means that for $c_1 = 1$ and $n_0 = 1$ the left hand inequality holds, concluding the proof.

4. (15 pts) Is the following statement **true** or **false**, and why? (provide a formal argument either way)

$$\text{If } f(n) = O(g(n)) \text{ then } 2^{f(n)} = O(2^{g(n)}).$$

Solution: False. Leading constants don't matter, but constants in the exponent certainly do. For example, let $f(n) = 2n$ and $g(n) = n$. Then $f(n) = O(g(n))$, but $2^{2n} = 4^n$ is not $O(2^n)$, because $2^{2n}/2^n = 2^n$ is not bounded by any constant.

5. (20 pts) What value is returned by the following function? Express your answer as a function of n. From this, deduce the worst-case running time using Big Oh notation. For full credit, provide a lower bound as well, thus leading to a bound with $\Theta(\cdot)$ notation. Then, also deduce the worst-case running time if we change the while loop clause to $j \leq i$ instead of $j \leq n$.

```
function foo1($n$)
$r := 0$
for $i := 1$ to $n$ do
    $j := 1$
    while $j \leq n$ do
        $r := r + 1$
        $j := 2 \cdot j$
return $r$
```

Solution: For the first portion of the question, we observe that the inner while loop executes $\lfloor \log n \rfloor + 1$ times on each for loop iteration. Therefore, to find the resulting output of the function we may construct the summation

$$\sum_{i=1}^{n} \sum_{k=1}^{\lfloor \log n \rfloor + 1} 1 = \sum_{i=1}^{n} \lfloor \log n \rfloor + 1 = n \lfloor \log n \rfloor + n.$$

We can observe that on every loop iteration we perform a constant number of constant time operations, so the value of r directly implies the asymptotic worst-case running of $\Theta(n \log n)$. Alternatively, we may observe that if the outer loop ran from 1 to $2n$, the total running time would be less than $2n \log n + n$ which is $O(n \log n)$. Similarly, if the outer loop ran from 1 to $\frac{n}{2}$, the total running time would be greater than $\frac{n}{2} \log n$ which is $\Omega(n \log n)$.

We now move to consider the second portion of the question and allow j to range from 1 to i. The return value of r will instead be the summation $\sum_{i=1}^{n} \lfloor \log i \rfloor + 1$. For the asymptotic worst-case running time, observe that the same upper bound from the previous portion holds since the previous portion had moor loop iterations in the inner loop. It remains to give a lower bound. Observe that we may construct a smaller loop by varying i from $\frac{n}{2}$ to n. Since the arithmetic and branching operations take constant time, this would give us a running time of a constant times the summation

$$\sum_{i=n/2}^{n} \lfloor \log i \rfloor + 1 \geq \sum_{i=n/2}^{n} \log i - 1 + 1 = \sum_{i=n/2}^{n} (\log i).$$

But this summation is at least $\frac{n}{2} \log \frac{n}{2}$ since we add $\frac{n}{2} + 1$ copies of terms greater than or equal to $\log \frac{n}{2}$. By log laws, this is exactly $\frac{n}{2} \log n - \frac{n}{2} \log 2 = \frac{n}{2} \log n - \log n$. This quantity is lower bounded by $\Omega(n \log n)$, therefore the running time of the loop where i varies from 1 to n is also lower bounded by $\Omega(n \log n)$. This means that the running time of the loop where j ranges from 1 to i is still $\Theta(n \log n)$.

6. (14 pts) What value is returned by the following function? Express your answer as a function of n. From this, deduce the worst-case running time using Big Oh notation. For full credit, provide a lower bound as well, thus leading to a bound with $\Theta(\cdot)$ notation.

```
function foo2($n$)
$r := 0$
$j := n$
while $j \geq 1$ do
    for $i := 1$ to $j$ do
        $r := r + 1$
    $j := j/2$
return $r$
```

Solution: Although this question seems quite similar to the previous one, it leads to a very different running time (and return value of r). For the value of r, observe that the inner loop runs from 1 to j and j is all of the powers of 2 from n down to 1. Therefore, we may write the value of r as a summation as $\sum_{i=0}^{\lfloor \log n \rfloor + 1} \lfloor \frac{n}{2^i} \rfloor$. If we remove the floor operations, this is a geometric series with common factor n and recurring ratio 1/2. The running time of this code block is $\Theta(n)$. To see that the running time of this code block is $\Omega(n)$, observe that removing the while loop makes the process shorter. This would still take linear time since j is initially equal to n and run a for loop from 1 to j. Therefore, our restricted code block would be $\Omega(n)$ and thus our initial code block is also $\Omega(n)$. To see that the running time of this code block is $O(n)$ observe that the value returned for r multiplied by constants (for the branching and arithmetic operations) is the running time. This value is upper bounded by the infinite geometric summation $\sum_{i=0}^{\infty} \frac{n}{2^i} = 2n$. Since the running time is both $\Omega(n)$ and $O(n)$, we have that it is $\Theta(n)$, completing the proof.

7. (9 pts) Show that, if c is a positive real number, then $g(n) = 1 + c + c^2 + \cdots + c^n$ is

   a) $\Theta(1)$ if $c < 1$

   Solution: We begin by rewriting $g(n)$ as a familiar geometric series:

$$g(n) = \sum_{k=0}^{n} c^k.$$

   We know that the sum of a geometric series with ratio $r \neq 1$ has the value:

$$\sum_{k=0}^{n} c^k = \frac{c^{n+1} - 1}{c - 1}.$$

   To show that this value is in the class $\Theta(1)$, we must find constants n_0, k_0 and k_1 such that

$$k_0 \leq \frac{c^{n+1} - 1}{c - 1} \leq k_1 \qquad \forall n > n_0.$$

   It is clear that setting $k_0 = 1$ allows the left hand side of the inequality to hold since all terms in the summation are positive and the first one is 1. We also know that since this summation is finite and all the terms in the equivalent infinite summation are positive:

$$\frac{c^{n+1} - 1}{c - 1} \leq \sum_{k=0}^{\infty} c^k = \frac{1}{1 - c}.$$

Thus, $k_1 = \frac{1}{1-c}$ is an upper bound for the summation and we have that our summation is $\Theta(1)$.

b) $\Theta(n)$ if $c = 1$

Solution: We begin by noting that $1^k = 1$ for any k. Thus, we have in our sum:

$$g(n) = 1 + 1 + ... + 1 = \sum_{k=0}^{n} 1.$$

This sum is clearly equal to $n+1$. To show that $n+1$ is in the class $\Theta(n)$, we must find constants n_0,k_0 and k_1 such that

$$k_0 n \leq n + 1 \leq k_1 n \qquad \forall n > n_0.$$

Selecting $k_0 = \frac{1}{2}$ and $k_1 = 2$ and $n_0 = 5$ satisfies the inequalities.

c) $\Theta(c^n)$ if $c > 1$

Solution: We begin by rewriting $g(n)$ as a familiar geometric series:

$$g(n) = \sum_{k=0}^{n} c^k.$$

We know that the sum of a geometric series with ratio $r \neq 1$ has the value:

$$\sum_{k=0}^{n} c^k = \frac{c^{n+1} - 1}{c - 1}.$$

To show that this value is in the class $\Theta(c^n)$, we must find constants n_0,k_0 and k_1 such that

$$k_0 c^n \leq \frac{c^{n+1} - 1}{c - 1} \leq k_1 c^n \qquad \forall n > n_0.$$

We observe that all terms in the series are positive since $c > 1$. We also have that the last term is c^n so it is clear that allowing $k_0 = 1$ will make the left side of the inequality hold.
We now examine the right hand side of our original inequality.

$$\begin{array}{ll} \frac{c^{n+1}-1}{c-1} & \leq k_1 c^n \\ c^{n+1} & \leq k_1 c^n (c-1) + 1 \\ c & \leq k_1 (c-1) + \frac{1}{c^n} \end{array}$$

We observe that $\frac{1}{c^n}$ is positive so if $c \leq k_1(c-1)$ holds, so does our above inequality. Dividing through gives $\frac{c}{c-1} \leq k_1$ which implies we may select $k_1 = \frac{c}{c-1}$ which holds for our above selection of $n_0 = 1$.