Courtney Wright
260608954

COMP 250 ASSIGNMENT #2

1) Mathematically, for $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ to be true, $g(n)$ must grow at a faster rate than $f(n)$. Knowing that $O(g(n))$ is an upper bound of $g(n)$ and $f(n)$ grows slower than $g(n)$, we know $f(n)$ is $O(g(n))$.

$$f(n) < g(n) \in O(g(n)) \Rightarrow f(n) \in O(g(n))$$

But, since we know $f(n)$ grows slower than $g(n)$, it is not possible for $f(n)$ to be both $O(g(n))$ and $\Omega(g(n))$ because $f(n)$ will never grow as quickly as $g(n)$, therefore it must be that $f(n)$ is also $O(f(n))$ for which $g(n)$ is not $O(f(n))$. This proves that $O(g(n))$ cannot be $\Omega(f(n))$. We know that for $g(n)$ to be $\Theta(g(n))$, $g(n)$ must be $O(g(n))$ and $\Omega(g(n))$. For $f(n)$ to be $\Theta(g(n))$, $f(n)$ must be $O(g(n))$ and $\Omega(f(n))$, but we now know that $O(g(n))$ cannot be $\Omega(f(n))$, therefore, $f(n)$ cannot be $\Theta(g(n))$.

2) a) i) $99(2n) = 198n \Rightarrow$ two times slower

ii) $99(n+1) = 99n + 99 \Rightarrow \frac{n+1}{n}$ times slower

b) i) $(2n)^2 = 4n^2 \Rightarrow$ four times slower

ii) $(n+1)^2 = n^2 + 2n + 1 \Rightarrow (1 + \frac{1}{n})^2$ times slower

c) i) $(2n)^4 = 256n^4 \Rightarrow 128$ times slower

ii) $(n+1)^4 = (n^2+2n+1)(n^2+2n+1) = n^4 + 4n^3 + 6n^2 + 4n + 1$

$\Rightarrow (1 + \frac{1}{n})^4$ times slower

d) i) $(2n)\, 2^{(2n)} \Rightarrow 2^{n+1}$ times slower

ii) $(n+1)2^{n+1} = n2^{n+1} + 2^{n+1} \Rightarrow 2(1 + \frac{1}{n})$ times slower

e) i) $3^{2n} \Rightarrow 3^n$ times slower

ii) $3^{n+1} \Rightarrow 3$ times slower

2 cont.) slowest growth rate g) $f_2(n) = 2^{\log(n)}$

f) $f_1(n) = 99n^2$

h) $f_3(n) = n^2 \log(\log(n))$

i) $f_4(n) = n2^n$

fastest growth rate j) $f_5(n) = 3^n$

3) a) $\log_2(f(n))$ is not $O(\log_2(g(n)))$

counter example: $f(n) = 2(1+\frac{1}{n})$, $g(n) = (1+\frac{1}{n})$

$\log_2(f(n)) = \log_2 2 + \log_2(1+\frac{1}{n})$

$\log_2(g(n)) = \log_2(1+\frac{1}{n})$.

$f(n) \in O(g(n))$ but $\log_2(f(n)) \not\in O(\log_2(g(n)))$.

b) $2^{f(n)}$ is not $O(2^{g(n)})$

counter example: $f(n) = 2n$, $g(n) = n$.

$2^{f(n)} = 2^{2n}$, $2^{g(n)} = 2^n$

$f(n) \in O(g(n))$ but $2^{f(n)} \not\in O(2^{g(n)})$.

c) $f(n)^2$ is $O(g(n)^2)$

Proof: it is known that squaring is order preserving for all positive values. This property on its own ensures that $f(n)^2$ must <u>ALWAYS</u> be $O(g(n)^2)$.

4) algo1(n) is $O(n)$

algo2(n) is $O(n^2)$

algo3(n) is $O(\log(n))$

algo4(n) is $O(1)$

5) $\sum_{i=0}^{n} i^k$ can be represented by a mathematical geometric series: $\sum_{n=0}^{N} x^n$ for which we know how to manipulate.

Specifically as $\lim_{x\to 1}\left(x\cdot\frac{d}{dx}\right)^k\left(\sum_{n=0}^{N} x^n\right) = \sum_{i=0}^{n} i^k$.

If we expand this expression into terms we have

$$\lim_{x\to 1}\left(x\cdot\frac{d}{dx}\right)^k\left(1+x+x^2+x^3+\ldots+x^N\right) = \left(0+1^k+2^k+3^k+\ldots+N^k\right)$$

Taking the partial sum of $\sum_{n=0}^{N} x^n$ we have

$S_N = \frac{x^{N+1}-1}{x-1}$. Deriving again and multiplying by x we get

$$S_N'\cdot x = \left((N+1)x^n + \frac{1}{x-1}\right)\cdot x.$$

If this is then repeated k times we have $\left(x\cdot\frac{d}{dx}\right)^k\left(\frac{x^{N+1}-1}{x-1}\right)$. We now take the limit of this as $x\to 1$ and get

$$\lim_{x\to 1}\left(x\cdot\frac{d}{dx}\right)^k\left(\frac{x^{N+1}-1}{x-1}\right) = \left(c_0N^{k+1}+c_1N^k+c_2N^{k-1}+\ldots+c_N\right)$$

Where the highest power of N we have is $(k+1)$. From this we know that for some constant a and some constant b such that $b \geq a$ we have

$$a\cdot n^{k+1} \leq \lim_{x\to 1}\left(x\cdot\frac{d}{dx}\right)^k\left(\frac{x^{N+1}-1}{x-1}\right) \leq b\cdot n^{k+1}$$

proving the original statement.

6) $f(x) = x$
$g(x) = x^{(1+\sin(x))}$

7) a) tortoise == hare at the blue circle.

b) tortoise == hare at the green pentagon.

c) let f = length from first node to connecting node.

ℓ = length of loop.

d = length from connecting node to node at which tortoise & hare first meet.

c = constant

When tortoise and hare meet, the distance tortoise has traveled from the start is

turtle_steps = $f + c_t \ell + d$

and the distance the hare has traveled is

hare_steps = $f + c_h \ell + d$

where we know $c_h > c_t$ because (speed_tortoise) < (speed_hare). Since we know (speed_tortoise) = $\frac{1}{2}$(speed_hare), we have

2(turtle_steps) = (hare_steps) so

$2(f + c_t \ell + d) = f + c_h \ell + d$ and

$f + d = (c_h - 2c_t)\ell$

which implies that $f+d$ is an integer multiple of the length of the loop.

After hare is sent back to the begining of the list, it travels f steps (or nodes) to get to the connecting node. At the same time, the tortoise will move the same number of steps: f. Since the tortoise starts at d, it will have traveled $m+k$ steps from the connecting node. This implies that the connecting node (we had assumed it was the connecting node), MUST indeed be where the list loops back to because we know (from before) that $f+d$ is an integer multiple of the length of the loop.

7 (cont.) d)

```
mylist ← turtle                 // set turtle to head
mylist ← bunny                  // set bunny to head

turtle.next                     // make first move for each
bunny.next.next                 // so they aren't on same node

while bunny != turtle           // both keep taking steps
    turtle.next                 // until they meet
    bunny.next.next

mylist ← bunny                  // send bunny back to head
bunny ← NR                      // make NR point to head

while bunny != turtle           // move each one at a
    turtle.next                 // time until they meet
    bunny.next                  // again

bunny ← R                       // make R point to connecting node
NR ← counter                    // initialize counter at head

while counter.next != bunny     // increment counter until it's
    counter.next                // two before connecting node

NULL ← counter.next             // remove pointer for node before
                                // connecting node
while turtle.next != bunny      // increment turtle until it's two
    turtle.next                 // before connecting node

NULL ← turtle.next              // remove pointer from end of loop node

return NR, R
```

7 cont.) e) Let n= number of cells in list

algorithm (part d) is $O(n)$
space used by algorithm is $O(n)$

f) $f(s) \leftarrow$ turtle // use turtle & bunny, set both to
$f(s) \leftarrow$ bunny // f(s) to start.

while bunny != turtle // increment turtle by one
    $f(f(f(s))) \leftarrow$ bunny // and bunny by two until
    $f(f(s)) \leftarrow$ turtle // they meet. Increment n & k
    $n+1 \leftarrow n$ // each by one until turtle and
    $k+1 \leftarrow k$ // bunny meet.

$f(s) \leftarrow$ bunny //send bunny back to head
turtle ← meet_one // instantiate meet_one to where they meet

while bunny != turtle // increment each by 1
    $f(f(s)) \leftarrow$ bunny // until they meet again
    $f(f(s)) \leftarrow$ turtle

$f(\text{turtle}) \leftarrow n$
n − meet_one ← k

g) Let n= number of cells in list
algorithm (part f) is $O(n)$
space used by algorithm is $O(1)$

h) Correlation lies in repeating decimal section of Assignment #1 and finding where the repeat starts.