Algorithm Analysis and Data Structures
CS 5343.001: Homework #2

Due on September 12, 2016 at 11:59pm

Professor Greg Ozbirn

Lizhong Zhang(lxz160730)

Problem 1

Time complexity is asymptotic. So, considering when $n \to \infty$, we need require "$N \geq n_0$", which means time complexity is always stable for all values after n_0.

Problem 2

According to the definition of Big $\mathcal{O}$, $T(N) = \mathcal{O}(f(N))$ means that $T(N) \leq cf(N)$ for some constant c and for $N \geq n_0$, and $f_1(N) = 2N \leq cN$, $f_2(N) = 3N \leq cN$ for some constant c (e.g. $c = 4$) when $N \geq 1$. So, they are both $\mathcal{O}(N)$.

Problem 3

Part (a)

$f_1(5) = 10, f_2(5) = 15, f_1(10) = 20, f_2(10) = 30$, When N was doubled in each case, the result became double.

Part (b)

$f_1(5) = 50, f_2(5) = 75, f_1(10) = 200, f_2(10) = 300$, When N was doubled in each case, the result became quadruple.

Problem 4

Algorithm analysis is dedicated to understanding the complexity of algorithms that could be expressed by Big-$\mathcal{O}$. And Assume two functions $f(N)$ and $g(N)$ are considered as two algorithms, as the scale of the problem(N) increases, $\mathcal{O}(f(N))$ and $\mathcal{O}(g(N))$ are their own growth rates of algorithm execution time. In the meantime, if there exists $\mathcal{O}(f(N)) < \mathcal{O}(g(N))$, which means complexity of algorithm $g(N)$ is more than complexity of algorithm $f(N)$. So Big-$\mathcal{O}$ could be applicable to algorithms analysis.

Problem 5

$n!$ grows faster.
Proof:

Base Case: $n = 4, 2^4 = 16 < 4! = 24$, So, it is true for $n = 4$.

Induction step: Assume it is true for k, that $2^k < k!$.

Problem 5 continued on next page. . .

Show true for $k+1$:

$$2^{(k+1)} = 2^k \times 2$$
$$(k+1)! = k! \times (k+1)$$

Due to $2^k < k!$ and $2 < k+1$, so $2^{(k+1)} < (k+1)!$
Conclusion: by induction, the statement holds true for all $n \geq 4$. So $n!$ grows faster. $\square$

Problem 6

(a) $\mathcal{O}(n^5)$

(b) $\mathcal{O}(5^n)$

(c) $\mathcal{O}(n)$

(d) $\mathcal{O}(n\log(n))$

(e) $\mathcal{O}(n^2)$

Problem 7

i=0, i < numItems; i++; // $1+(n+1)+n$
$1+(n+1)+n = 2n+2$, so the result is $\mathcal{O}(n)$.

Problem 8

i=0; i<numItems; i++; // $1+(n+1)+n$
j=0; j<numItems; j++; // $n(1+(n+1)+n)$
(i+1) * (j+1); // $n \times n \times 3$
// $1+(n+1)+n+n(1+(n+1)+n)+n \times n \times 3 = 5n^2+4n+2$, so the result is $\mathcal{O}(n^2)$.

Problem 9

i=0; i<numItems+1; i++; // $1+(n+2)+(n+1)$
j=0; j<2*numItems; j++; // $(n+1)(1+(2n+1)+2n)$
(i+1) * (j+1); // $(n+1) \times 2n \times 3$
// $1+(n+2)+(n+1)+(n+1)(1+(2n+1)+2n)+(n+1) \times 2n \times 3 = 10n^2+14n+6$, so the result is $\mathcal{O}(n^2)$.

Problem 10

When num < numItems:
num < numItems; // 1
int i=0; i < numItems; i++; // $1 + (n + 1) + n$
System.out.println(i); // n
$1 + 1 + (n + 1) + n + n = 3n + 3$, so the result is $\mathcal{O}(n)$.
When num > numItems:
“too many”; // 1
So the result is $\mathcal{O}(1)$.

Problem 11

i = numItems; // 1
i > 0; // $\lg n + 1$
i = i / 2; // $\lg n$
$1 + \lg n + 1 + \lg n = 2 + 2 \lg n$, so the result is $\mathcal{O}(\lg n)$.

Problem 12

numItems == 0; // $1 + \lg n$
return 0 // 1
numItems%2 + div(numItems/2) // $2 \lg n$
$1 + \lg n + 1 + 2 \lg n = 3 \lg n + 2$, so the result is $\mathcal{O}(\lg n)$.