CS208 PART III: ALGORITHMS AND COMPLEXITY
EXERCISE SHEET FOR TUTORIAL 1

T. SELIG, 31 JAN 2016

1. **Big $\mathcal{O}$ notation**
   (a) Let $f(n) = 5n^2 + 2n + 4$. Is f $\mathcal{O}(n^3)$? Can this bound be improved, i.e. is there a function h such that f is $\mathcal{O}(h)$, where h is "smaller" than $g(n) = n^3$?
   (b) Give an example of a function which is $\mathcal{O}(n^7)$ but not $\mathcal{O}(n^5)$.
   (c) Show that $f(n) = 4n^2$ is not $\mathcal{O}(n)$.

   SOLUTION: Recall the definition that f is $\mathcal{O}(g)$: there exists an integer $N \geq 1$ and a real number $c > 0$ such that for all $n \geq N$, we have $f(n) \leq c.g(n)$.
   (a) For any $n \geq 1$ we have $1 \leq n \leq n^2 \leq n^3$, thus

   $$4 \leq 4n \leq 4n^2 \leq 4n^3, \quad 2n \leq 2n^2 \leq 2n^3, \quad 5n^2 \leq 5n^3.$$

   Summing these inequalities gives us that for any $n \geq 1$, we have

   $$f(n) = 5n^2 + 2n + 4 \leq 5n^3 + 2n^3 + 4n^3 = 11n^3.$$

   Thus setting $N = 1$ and $c = 11$ yields the desired result.
   (b) The idea is to use the result that a polynomial belongs to the complexity class of its largest exponent. So any polynomial of degree 7 (or indeed 6) will be $\mathcal{O}(n^7)$ but not $\mathcal{O}(n^5)$. I don't want them to actually prove this.
   (c) To show that a function f is not $\mathcal{O}(g)$ requires one to negate the above definition, i.e. show that for any integer $N \geq 1$ and real number $c > 0$ there exists $n \geq N$ such that $f(n) > c.g(n)$. Therefore fix $N \geq 1$ and $c > 0$.
   We want to show that there exists $n \geq N$ such that $4n^2 > c.n$, i.e. $4n > c$, i.e. $n > \frac{c}{4}$. If we define $n = \max\{N, \lceil c/4 + 1 \rceil\}$, then clearly $n \geq N$ and $n > \frac{c}{4}$, which yields the desired result.

2. For each of the algorithms below, and each of the pricing schemes (a) and (b), write down expressions for the worst case time complexity and the asymptotic complexity.

Function	Cost (a)	Cost (b)
`print`	1	c_1
`+` and `-`	1	c_2
`*`	1	c_3

Algorithm 1 AlgOne, input is integer n

```
1: for $i = 1$ to $n$ do
2:     print $i$
3:     print $3 + i$
4: end for
```

SOLUTION:

Costing (a):

$$\begin{aligned} T_{AlgOne}(n) &= n \times (T_{print\ i}(n) + T_{print\ 3+i}(n) \\ &= n \times (1 + 2) \\ &= 3n. \end{aligned}$$

Consequently $T_{AlgOne}(n)$ has asymptotic complexity $\mathcal{O}(n)$.

Costing (b):

$$\begin{aligned} T_{AlgOne}(n) &= n \times (T_{print\ i}(n) + T_{print\ 3+i}(n) \\ &= n \times (c_1 + (c_1 + c_2)) \\ &= (2c_1 + c_2)n. \end{aligned}$$

Consequently $T_{AlgOne}(n)$ has asymptotic complexity $\mathcal{O}(n)$.

Algorithm 2 AlgTwo, input is integer n

```
1: while $n - 5 > 0$ do
2:     print $(n * 2 + 4)$
3:     $n \leftarrow n - 1$
4: end while
5: for $i = 1$ to $6$ do
6:     print $i$
7: end for
```

SOLUTION:

Costing (a):

$$\begin{aligned} T_{AlgTwo}(n) &= (n - 5 + 1) \times T_{n-5>0}(n) + (n - 5) \times (T_{print\ n*2+4}(n) + T_{n \leftarrow n-1}(n)) \\ &\quad +6 * T_{print\ i}(n) \\ &= (n - 4) \times 1 + (n - 5) \times (3 + 1) \\ &\quad +6 * 1 \\ &= n - 4 + 4(n - 5) + 6 \\ &= 5n - 20 + 2 = 5n - 18. \end{aligned}$$

Costing (b):

$$\begin{aligned}
T_{AlgTwo}(n) &= (n-5+1)\times T_{n-5>0}(n) + (n-5)\times(T_{print\ n*2+4}(n) + T_{n\leftarrow n-1}(n)) \\
&\quad +6 * T_{print\ i}(n) \\
&= (n-4)\times c_2 + (n-5)\times(c_1+c_2+c_3+c_2) \\
&\quad +6 * c_1 \\
&= c_2(n-4) + (2c_2+c_1+c_3)(n-5) + 6c_1 \\
&= (3c_2+c_1+c_3)n - 4c_2 - 5(2c_2+c_1+c_3) + 6c_1 \\
&= (3c_2+c_1+c_3)n - 14c_2 + c_1 - 5c_3.
\end{aligned}$$

In either case, the asymptotic complexity of $T_{AlgTwo}(n)$ is $\mathcal{O}(n)$.

Algorithm 3 AlgThree, input is integer n

```
1: for $i = 1$ to $n$ do
2:     for $j = 1$ to $i$ do
3:         print $(i + j)$
4:     end for
5: end for
```

SOLUTION: A very exact way of doing it is as follows. If you use the rules for decomposing the 2 for loops in the lecture slides, then you must make sure you are using the worst case time complexity and that you end up with an answer that is quadratic. The method we now use shows the exact complexity:

Costing (a):

$$\begin{aligned}
T_{AlgThree}(n) &= 1 * T_{print\ i+j} + 2 * T_{print\ i+j} + \ldots + n * T_{print\ i+j} \\
&= (1+2+3+\ldots n) * T_{print\ i+j} \\
&= \frac{n(n+1)}{2}(1+1) \\
&= n(n+1) \\
&= n^2 + n.
\end{aligned}$$

Costing (b):

$$\begin{aligned}
T_{AlgThree}(n) &= 1 * T_{print\ i+j} + 2 * T_{print\ i+j} + \ldots + n * T_{print\ i+j} \\
&= (1+2+3+\ldots n) * T_{print\ i+j} \\
&= \frac{n(n+1)}{2}(c_1+c_2) \\
&= \frac{(c_1+c_2)}{2}n(n+1) \\
&= \frac{(c_1+c_2)}{2}n^2 + \frac{(c_1+c_2)}{2}n.
\end{aligned}$$

In either case, the asymptotic worst case time complexity is $\mathcal{O}(n^2)$.

(You may find it useful to know that $1 + 2 + 3 + \ldots + m = \frac{m\times(m+1)}{2}$.)

3. In the following algorithm, suppose that the primitive operations which contribute to the time complexity of the algorithm are addition and multiplication (where each contribute 1 unit).

Algorithm 4 card(non-negative integer n)

```
1: if $n$ is 0 then
2:         $answer \leftarrow 1$
3: else if $n$ is odd then
4:         $answer \leftarrow n + 3 * \mathsf{card}(n-1)$
5: else if $n$ is even and positive then
6:         $answer \leftarrow n + n + 3 * n * \mathsf{card}(n-1)$
7: end if
8: return $answer$
```

Write down an expression for the worst case time complexity $T_{card}(n)$ and comment on its asymptotic complexity.

SOLUTION: We find that

$$T_{card}(n) \;=\; \begin{cases} 0 & \text{if } n = 0 \\ 2 + T_{card}(n-1) & \text{if } n \text{ odd} \\ 4 + T_{card}(n-1) & \text{if } n \text{ even and positive.} \end{cases}$$

Therefore the worst case time complexity is given by the equation:

$$T_{card}(n) = 4 + T_{card}(n-1).$$

This corresponds to the case seen in the lectures where $T(n) = k + T(n-1)$ for some constant $k > 0$. As such, the asymptotic complexity of $T_{card}(n)$ is $\mathcal{O}(n)$.

4. Let f, g, F, and G be functions from $\mathbb{N} = \{1, 2, 3, \ldots\}$ to $\mathbb{N}$.

Suppose that f is $\mathcal{O}(F)$ and g is $\mathcal{O}(G)$.

Let h be the function defined by $h(n) = f(n) \times g(n)$ and let H function $H(n) = F(n) \times G(n)$.

(a) Write down what it means for f to be $\mathcal{O}(F)$.
[Hint: use the definition given in the slides]
SOLUTION: f is $\mathcal{O}(F)$ means there exists a number c_1 and a number N_1 such that for all $n \geq N_1$, we have $f(n) \leq c_1 \times F(n)$.

(b) Write down what it means for g to be $\mathcal{O}(G)$.
[Hint: do not use the same constants that were used in part (a)]
SOLUTION: g is $\mathcal{O}(G)$ means there exists a number c_2 and a number N_2 such that for all $n \geq N_2$, we have $g(n) \leq c_2 \times G(n)$.

(c) Give a value N such that both of the inequalities in parts (a) and (b) hold for all $n \geq N$.
SOLUTION: So long as $N \geq \max(N_1, N_2)$ both inequalities will hold.

(d) Multiply both sides of the inequalities in parts (a) and (b) to get a new inequality.
SOLUTION: $f(n) \times g(n) \leq c_1 \times c_2 \times F(n) \times G(n)$

(e) Use parts (c) and (d) to show that h is $\mathcal{O}(H)$.

SOLUTION: Combining the facts from parts (c) and (d) we have that $f(n) \times g(n) \leq c_1 \times c_2 \times F(n) \times G(n)$ for all $n \geq \max(N_1, N_2)$. Setting $h(n) = f(n) \times g(n)$ and $H(n) = F(n) \times G(n)$, we see the definition of big $\mathcal{O}$ is satisfied with $c = c_1 \times c_2$ and $N = \max(N_1, N_2)$.