Tutorial 4, Week 5 - Basic Algorithm Analysis - Answers

1 Before Your Tutorial

- Read the additional material on asymptotic notation (the proper name for "big-oh"). This material goes into a fair bit of detail, you don't need to understand every precise detail.
- For each pair of functions f and g given in the following table, fill out whether $f \in O, \Omega$ or Θ of g:

f	g	O	Ω	Θ	o	ω
$5n$	$3n+1$	Yes	Yes	Yes	No	No
$3n^2+n-23$	$8n^3$	Yes	No	No	Yes	No
$n \log n$	n^2	Yes	No	No	Yes	No
2^n	$34n^{100}$	No	Yes	No	No	Yes
2^{n+1}	2^n	Yes	Yes	Yes	No	No
$5n^3-n+3$	$3n^3$	Yes	Yes	Yes	No	No

2 During Your Tutorial

- Examine the implementations of your `intLinkedList` class. Express their running time for a list of size n in terms of big-oh notation.

  These will vary by implementation of course:

Function	$O(\cdot)$
Constructors	$O(1)$
`~intNode()`	$O(1)$
`~intLinkedList()`	$O(n)$
`prepend(int)`	$O(1)$
`append(int)`	$O(n)$
`tail()`	$O(n)$
everything else	$O(1)$

- Examine the functions in the file `Week5Tutorial.cpp`. Make a prediction for each function about it's asymptotic running time in terms of big-oh notation. Use the magnitude of the input parameter (*i.e.* the value of the number) as the variable for the analysis.

Function	$O(\cdot)$
`stars(int)`	$O(height^2)$
`square(int)`	$O(height^2)$
`line(int)`	$O(length)$
`printRabbit(int)`	$O(1)$
`bunnies(int)`	$O(2^{gen})$
`gcd(int, int)`	$O(\log(\min\{n, m\}))$

- Modify the functions in `Week5tutorial.cpp` to print out the total number of steps they take (you must make a choice about a reasonable definition of 'step'). Use this code along with multiple runs to produce data to confirm or disconfirm your predictions (it's probably easiest to put the data into Excel or similar to plot it – you can fit the function by hand if you want though...).
- Put the following list of functions in order of their big-oh inclusion:

  $$\log_2 n, n^3, n, 2^n, 2^{\log_2 n}, \sqrt{n}, 6, n \log_2 n, n^2, n\sqrt{n}$$

  In order:

  $$6, \log_2 n, \sqrt{n}, n = 2^{\log_2 n}, n \log_2 n, n\sqrt{n}, n^2, n^3, 2^n$$

3 Extensions

- Extend the table from the prework section to include o and ω.

  See table.

- Imagine we have two algorithms $\mathcal{A}$ and $\mathcal{B}$ with running times $T_{\mathcal{A}} \in \Theta(n^2)$ and $T_{\mathcal{B}} \in \Theta(n^3)$. Assuming we only care about the running time, is it necessarily true that algorithm $\mathcal{A}$ is preferable to algorithm $\mathcal{B}$?

  If we only have small instances, then perhaps $\mathcal{B}$ may be better than $\mathcal{A}$. Asymptotic notation only considers instances that are "large enough". So the early behaviour of the algorithms may be quite different to their asymptotic analysis.

- Imagine we have two algorithms $\mathcal{A}$ and $\mathcal{B}$ with running times $T_{\mathcal{A}} \in O(n^2)$ and $T_{\mathcal{B}} \in O(n^3)$. Is it possible that there is an implementation of $\mathcal{B}$ that is faster than an implementation of $\mathcal{A}$ on all instances? Why or why not?

  Yes, $O(\cdot)$ only gives an upper boud. A tighter analysis may show that $T_{\mathcal{B}} \in \Theta(\log n)$ – it is still $O(n^3)$.

- Prove that for any polynomial $f(n) = \sum_{i=0}^{k} a_i n^i$ of degree k, $f(n) \in O(n^k)$.

  $$\begin{aligned} \sum_{i=0}^{k} a_i n^i &\leq \sum_{i=0}^{k} a_i n^k \\ &= n^k \cdot \sum_{i=0}^{k} a_i \end{aligned}$$

  So if we take $c = \sum_{i=0}^{k} a_i$ and $N = 0$, we have $f(n) \leq cn^k$ and hence $f \in O(n^k)$.

- Prove that for any a and b, $\log_a n \in O(\log_b n)$.

  Recall the basic logarithm identity, for any a, b and n:

  $$\log_a n = \frac{\log_b n}{\log_b a}$$

  So if we take $c = \frac{1}{\log_b a}$ we have $\log_a n = c \cdot \log_b n$, and hence is in $O(\log_b n)$.

- Before, you used the *magnitude* of the input parameter to measure the size of the input. What happends to the analysis if you use the actual size of the input parameter? (First you will have to determine what the size is, approximately.)

  The amount of space (written $||n||$) required to encode an integer n is $O(\log n)$ (we need $\lfloor \log_2 n \rfloor + 1$ bits, plus maybe some small overhead). So $n = 2^{||n||}$, and substituting this into the running times of the functions gives a range of exponential results.