ALGORITHMS

Loyola University Chicago

Antony Drew

COMP 460 HOMEWORK 1

January 28, 2014

Contents

Problem 1

This is a **quadratic** function (e.g., of the **second** order) when reduced most simply (via "**threshold**" rule):

- $f:N \rightarrow R^{\geq 0}$ such as $\mathbf{f(n) = n^2}$ **(**quadratic solution in simplest form)
- *Derivation*: By definition according to "threshold" rule, the solution must satisfy: $O(f(n)) \rightarrow \mathbf{t(n)} \leq cf(n)$ for all $n \geq n_0$ …
  - First, convert equation into **same** units or **milliseconds** here:
    - $t(n) = 3000 - 18n + 0.027n^2$
  - By definition, **t(n)** must be smaller than maximum degree (or highest order (e.g., "worst case") of the above equation) , so we can simply **substitute** here simplifying the equation to:
    - $t(n) \leq 3000n^2 - 18n^2 + 0.027n^2$
    - $t(n) \leq 2982.027n^2 \rightarrow \mathbf{2982.027\ f(n)}$ where $\mathbf{c} = 2982.027$ & $\mathbf{n_0} = 1$
    - Since some integer, "**n**" (or 1), is smaller than constant, "**c**" (or 2982.027), the "threshold" rule **holds true** and this quadratic is an acceptable solution to the problem…

Problem 2

In the **real** world, A is preferable to B. But, in the **ideal** world, A is **equivalent** to B. To see this, we can **simplify** these two algorithms and describe this **general** problem as follows:

- $\Theta(f(n)) = \mathbf{n^2 = \Theta(n^3)}$ or $n^2 \in \Theta(n^3)$ ...so, this simplified equation basically means that ***A and B are equal*** in terms of **generalized**, "Big-oh" notation – **ideally**, B can be **"no worse"** or "**no better**" than A (due to "**theta**" notation where both "threshold" and "maximization" rules hold true)…
- However, in the **practical** world, algorithm A and B **are** different so algorithm selection can have consequences

    - Even though “space” and “programming” complexity has been nullified (as given in the problem), “**time**” complexity also involves computing time (e.g., run-time computation).
    - For example, in the **real** world (all things being equal), it is **always faster** to multiply **2** identical **arrays** (or $O(n^2)$) by each other rather than **3** identical **arrays** (or $O(n^3)$) even though “*n^2 is in the order of n^3*”.

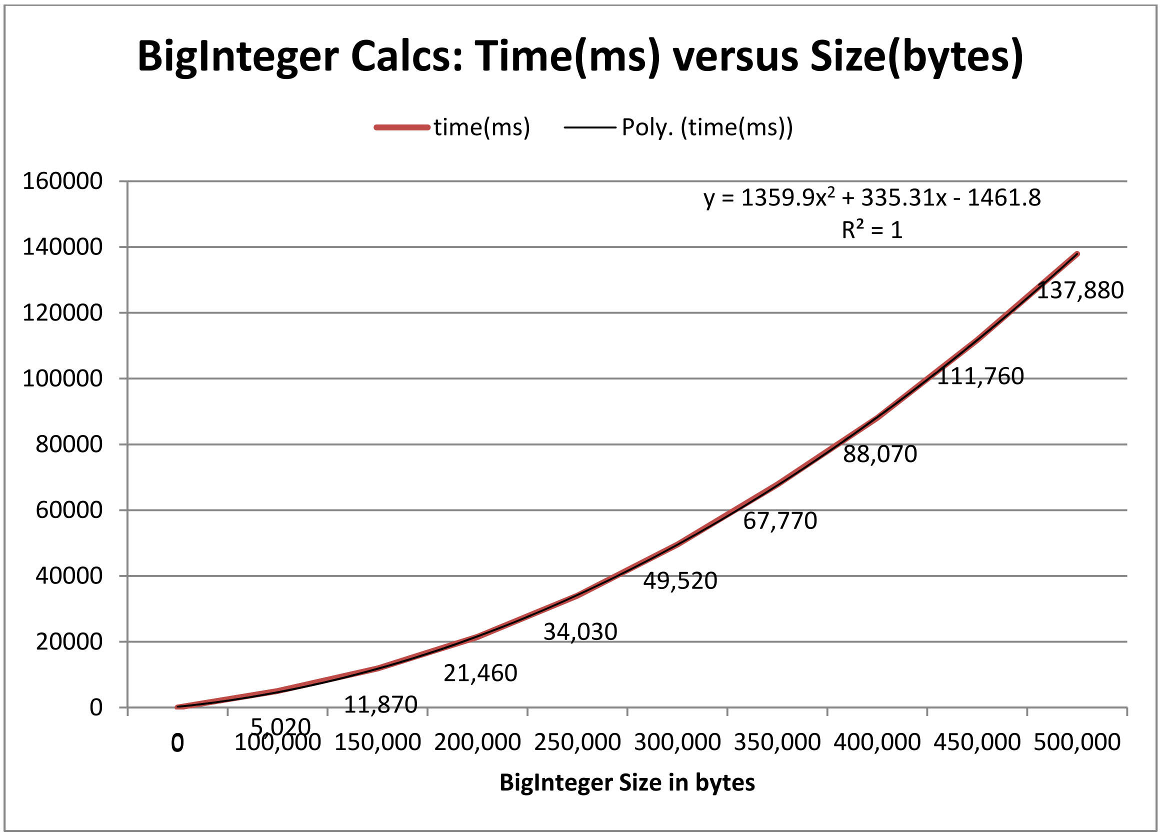
Problem 3

All things being equal, A should be faster than B in the real world (though extreme, unbalanced scenarios are possible, albeit, unlikely):

- For example, in the **real** world (all things being equal), it is **always faster** to multiply **2** identical **arrays** (or $O(n^2)$) by each other rather than **3** identical **arrays** (or $O(n^3)$) even though “*n^2 is in the order of n^3*”.
    - However, imagine a situation involving **unbalanced** circumstances:
        - If A has to multiply 2 large arrays (e.g., **int[100] * int[100]**), while B has to multiply 3 tiny arrays (e.g., **int[2] * int[2] * int[2]**), then obviously B will be faster in this case (since we have 100^2 versus 2^3 calculations). Keep in mind that this is a unfair example in that the conditions for each algorithm vary (in terms of array sizes) – so, in this case, **not** all things are equal
    - **Secondly**, imagine a quicker solution for the multiplication of **two** integers. In the textbook, there is a solution called “***divide-and-conquer***” (in Chapter 7). In this extreme example, it is actually **faster** to break-and-transpose the 2 integers **into 3 operations or multiples**. So, now we have gone from **$O(n^2)$ to $O(n^3)$** in terms of time complexity and yet the latter is still faster – this is a unique, non-intuitive example, however.

Problem 4

We have a **quadratic** operation or solution here. As the byte size or number of digits grow, so does the calculation time by a power of **2** (e.g., on the order of 2). So, we have a **convex** function here for "**BigInteger**" operations. Or, $\mathbf{f(n) = n^2}$ (or $\mathbf{O(n^2)}$) in generalized form. This is like multiplying 2 identical arrays by each other and has the same time complexity. It is easier to see a **picture** here to prove out this observation:



More **specifically**, if we try to fit a "non-linear" **regression** line through this chart (akin to a **trend** function), we can see the more specific equation of:

- **time = 1359.9*size^2 + 335.31*size – 1461.8** or in "Big-oh" notation:
  - $t(n) \leq 1359.9n^2 + 335.31n^2 - 1461.6n^2$
  - $t(n) \leq 233.41n^2 \rightarrow$ **233.41 f(n)** where **c** = 233.41 & $\mathbf{n_0} = 1$

- Since some integer, “**n**” (or 1), is smaller than constant, “**c**” (or 233.41), the “threshold” rule **holds true** and this quadratic is an acceptable solution to the problem…

Note that as we move further along the x-axis, the relative **change in time** becomes less severe (or grows more slowly). Note that this regression renders a **perfect fit** of 100% (or $\mathbf{R^2}$ of 1). This statistic is absolute proof that this function is quadratic and we have the right solution here.