National University of Singapore
School of Computing
CS1101S: Programming Methodology
Semester I, 2016/2017

Discussion Group Exercises Week 3

1 Abstraction

The first two problems in this Discussion Group sheet handle abstraction techniques. When you solve them, pay attention to the following:

- The specification of the problem: What do the English words in the problem mean?
- Abstractions: What are some useful abstractions that can help us in solving the problem?

Problems:

1. Define a function that takes three numbers as arguments and returns the sum of the squares of the two larger numbers.
2. Write a function, `is_leap_year`, which takes one integer parameter and decides whether it corresponds to a leap year, according to the Gregorian calendar.

2 Orders of Growth

Definitions

Theta (Θ) notation:

$f(n)$ has an order of growth of $\Theta(g(n)) \rightarrow$ There exist k_1, k_2, n_0 s.t.: $k_1 \cdot g(n) \leq f(n) \leq k_2 \cdot g(n), \text{for } n > n_0$

Big-O notation:

$f(n)$ has an order of growth of $O(g(n)) \rightarrow$ There exist k, n_0 s.t.: $f(n) \leq k \cdot g(n), \text{for } n > n_0$

Adversarial approach: For you to show that $f(n) = \Theta(g(n))$, you pick k_1, k_2, and n_0, then I (the adversary) try to pick an $n > n_0$ which doesn't satisfy $k_1 \cdot g(n) \leq f(n) \leq k_2 \cdot g(n)$.

Implications

Ignore constants. Ignore lower order terms. For a sum, take the larger term. For a product, multiply the two terms. Orders of growth are concerned with how the effort scales up as the size of the problem increases, rather than an exact measure of the cost.

Typical Orders of Growth

- $\Theta(1)$ - Constant growth. Simple, non-looping, non-decomposable operations have constant growth.
- $\Theta(\log n)$ - Logarithmic growth. At each iteration, the problem size is scaled down by a constant amount: `call_again(n / c)`.
- $\Theta(n)$ - Linear growth. At each iteration, the problem size is decremented by a constant amount: `call_again(n - c)`.
- $\Theta(n \log n)$ - Nifty growth. Nice recursive solution to normally $\Theta(n^2)$ problem.
- $\Theta(n^2)$ - Quadratic growth. Computing correspondence between a set of n things, or doing something of cost n to all n things both result in quadratic growth.
- $\Theta(2^n)$ - Exponential growth. Really bad. Searching all possibilities usually results in exponential growth.

What's n?

Order of growth is *always* in terms of the size of the problem. Without stating what aspect of the problem is growing, the order of growth (time or space) doesn't have any meaning.

Time and Space Consumption of Primitives

In our analysis, all primitive operations (arithmetics, built-in functions such as `Math.exp`, conditionals, function calls) are considered primitive. We assume that they take constant time. We also assume that all values (numbers, boolean values, strings and function values) take constant space.

Problems:

1. Assume $r_1(n) = 4n^2 - n$. Let us say we want to prove that r_1 has quadratic order of growth, i.e. $r_1(n)$ is $\Theta(n^2)$. Following the definition of Θ, give k_1, k_2, n_0 such that $k_1 \cdot n^2 \leq r_1(n) \leq k_2 \cdot n^2, \text{for } n > n_0$
2. Assume $r_2(n) = 10n \ \log n$. Ben Bitdiddle claims that $r_2(n)$ is $O(n^2)$. Following the definition of O, he gives $n_0 = 5$ and $k = 2$, and states that $r_2(n) \leq k \cdot n^2, \text{for } n > n_0$. What do you make of his claims?
3. Assume $r_3(n) = n^3$. Louis Reasoner claims that $r_3(n)$ is $O(2^n)$. Can you help him prove or disprove this claim?
4. For each of the following functions, find the simplest function that has the same order of growth:

   (a) $5n^2 + n$ has order of growth $\Theta($ $)$.

   (b) $\sqrt{n} + n$ has order of growth $\Theta($ $)$.

   (c) $3^n n^2$ has order of growth $\Theta($ $)$.

5. ```
   function factorial(n) {
       if(n === 0) {
           return 1;
       } else {
           return n * factorial(n-1);
       }
   }
   ```

   Use Θ notation to characterize the running time and space consumption of this function, as the argument n grows.

   Running time? Θ(   )   Space? Θ(   )

6. Write a version of `fact` that gives rise to an iterative process.

   Use Θ notation to characterize the running time and space consumption of your new version, as the argument n grows.
   ```