Algorithm Analysis

Jonathan Windle

University of East Anglia

J.Windle@uea.ac.uk

May 20, 2017

Overview I

1. Algorithm Design
   - Intro
   - Developing Algorithms
   - Algorithm Differences
   - Recursion
   - Tail Recursion

2. Algorithm Analysis
   - Run-time analysis
   - Characterising $t(n)$
   - Useful Summations
   - Notations
   - Analysis Example

Intro

- An algorithm is a step by step process for solving a problem. It consits of a finite sequence of instructions that when carried out, always terminates.
- Algorithms manipulate data structures.
- They are developed through a process of refinement, from an informal description to a formal description.
- Formal description is written in pseudocode.

Developing Algorithms

Steps to designing algorithm

1. Express in general terms how the algorithm works.
2. Give more detailed, but still informal description of the algorithm, identifying subproblems.
3. May be necessary to treat subproblems in the same way, known as step-wise refinement.
4. Give detailed, unambiguous description in pseudo-code.

Algorithm Differences

- Differeneces between algorithms can impact dramatically the speed of execution.
  - This is measured by run-time efficiency
- Differences can also mean they have different memory requirements
  - They are said to have different space efficiency
- There is often a trade-off between the two.

Recursion

- A recursive definition is something that is defined in terms of itself. (A method calling itself).
- Example is definition of factorial n:

$$\begin{aligned} 1! &= 1 \\ n! &= n \times (n-1)!, \quad \textit{for} \quad n > 1 \end{aligned} \tag{1}$$

- Rules for a recursive algorithm:
  1. Must have at least one base case and one recursive case.
  2. The recursive case should ensure that the base case is eventually reached.

Tail Recursion

- An algorithm is tail recursive if there is nothing to do after the return (except return its value).
- For example, this return is **NOT** tail recursive because it has to be multiplied by n before return: `return n` $\times$ `factorial(n-1);`
- This, however **IS** tail recursive: `return gcd(y,x%y);`
- Tail recursive algorithms can easily be turne into iterative algorithms.
- Iterative is usually more efficient to use.

Run-time analysis

- Strategy:
  1. Decide on fundamental operation.
  2. Decide on the case.
  3. Count the number of occurances given case (Usually the worst-case).
  4. Form the run-time complexity function $t(n)$ where n is the problem size.
  5. Characterise the order.

Characterising $t(n)$

In order of best to worst:

Constant	$t(n) = c$
Logarithmic	$t(n) = logn$
Linear	$t(n) = n$
Log Linear	$t(n) = nlogn$
Quadratic	$t(n) = n^2$
Polynomial Degree p	$t(n) = n^p$
Exponential	$t(n) = 2^n$

Useful Summations

- $$\sum_{i=1}^{n}(c) = c \times n$$
- $$\sum_{i=1}^{n}(i) = \frac{1}{2}n(n+1)$$
- $$\sum_{i=1}^{n}(c \times f(i)) = c \times \sum_{i=1}^{n}(f(i))$$
- $$\sum_{i=1}^{n}(f(i) + g(i)) = \sum_{i=1}^{n}(f(i)) + \sum_{i=1}^{n}(g(i))$$
- $$\sum_{i=m}^{n}(f(i)) = \sum_{i=1}^{n}(f(i)) - \sum_{i=1}^{m-1}(f(i))$$

Notations

- $O(f(n))$: *BigO*
    - For all n sufficiently large, $t(n)$ grows at no greater rate than $f(n)$.
    - This gives the upper-bound for the rate of growth.
    - $\Omega(f(n))$: *Omegan*
        - This denotes the lower-bound for the rate of growth.
    - When an algorithm is both $\Omega(f(n))$ and $O(f(n))$ then it is said to be $\Theta(f(n))$.

Analysis Example - Algorithm

- A matrix is Upper-triangular if all the elements below the diagonal are 0.

Algorithm 1 Output true if M is upper-triangular, false otherwise

Require: N/A

function IsUpperTriangular(M, n)
    for $i \leftarrow 2ton$ **do**
        for $j \leftarrow 1toi - 1$ **do**
            if $M[i, j] \neq 0$ **then**
                return *false*
            end if
        end for
    end for
    return *true*
end function

Analysis Example - Analysis

1. Determine fundamental operation:
   - Comparisoon: $M[i,j] \neq 0$
2. Decide on the case (Worst/Best)
   - worst-case, the worst case is that the matrix is upper-triangular.
3. Determine run-time complexity function $t(n)$ by counting fundamental operations

$$
\begin{aligned}
t(n) &= \sum_{i=2}^{n} \sum_{j=1}^{i-1} (1) \\
&= \sum_{i=2}^{n} (i-1) \\
&= \sum_{i=2}^{n} (i) - \sum_{i=2}^{n} (1) \\
&= \sum_{i=1}^{n-1} (i+1) - (n-1) \\
&= \sum_{i=1}^{n-1} (i) + \sum_{i=1}^{n-1} (1) - (n-1) \\
&= \frac{1}{2} n(n-1) + (n-1) - (n-1) \\
&= \frac{1}{2} n(n-1)
\end{aligned}
\tag{2}
$$

4. Give the order:
   - The worst case time complexity of this is both $O(n^2)$ and $\Omega(n^2)$ and is therefore $\Theta(n^2)$

The End