

Big-Oh Rules

Prof. Dianne Foreback

Information on slides originating from *Algorithm Design & Applications* by Michael T. Goodrich and Roberto Tamassia, © 2015 John Wiley & Sons, Inc. Goodrich and Tamassia, ISBN: 978-1118335918.

Reading Material

- ***Algorithm Design & Applications* by Michael T. Goodrich and Roberto Tamassia**
 - **Section 1.1.5**

Big-Oh Rules

Theorem 1.7: *Let $d(n)$, $e(n)$, $f(n)$, and $g(n)$ be functions mapping nonnegative integers to nonnegative reals.*

1. *If $d(n)$ is $O(f(n))$, then $ad(n)$ is $O(f(n))$, for any constant $a > 0$.*
2. *If $d(n)$ is $O(f(n))$ and $e(n)$ is $O(g(n))$, then $d(n) + e(n)$ is $O(f(n) + g(n))$.*
3. *If $d(n)$ is $O(f(n))$ and $e(n)$ is $O(g(n))$, then $d(n)e(n)$ is $O(f(n)g(n))$.*
4. *If $d(n)$ is $O(f(n))$ and $f(n)$ is $O(g(n))$, then $d(n)$ is $O(g(n))$.*
5. *If $f(n)$ is a polynomial of degree d (that is, $f(n) = a_0 + a_1n + \cdots + a_dn^d$), then $f(n)$ is $O(n^d)$.*
6. *n^x is $O(a^n)$ for any fixed $x > 0$ and $a > 1$.*
7. *$\log n^x$ is $O(\log n)$ for any fixed $x > 0$.*
8. *$\log^x n$ is $O(n^y)$ for any fixed constants $x > 0$ and $y > 0$.*

It is considered poor taste to include constant factors and lower order terms in the big-Oh notation. For example, it is not fashionable to say that the function $2n^2$ is $O(4n^2 + 6n \log n)$, although this is completely correct. We should strive instead to describe the function in the big-Oh in *simplest terms*.

Stylistic Note

- Do NOT write $f(N) \leq O(g(N))$ -- the inequality is implied by the definition
- Do NOT write $f(N) \geq O(g(N))$ -- the does not make sense
- For constant time, write $O(1)$. Do NOT put some other constant in the notation, e.g. $O(6)$ or $O(25)$

Runtime Analysis – For Loops

The running time of a for loop is at most the running time of the statements inside the for loop (including tests) times the number of iterations

```
for i ← 0 to n-1 do  
    k ← k+1
```

Counting Primitive Operations

initialize i to 0	1 primitive operation done once
i=i+1	2 primitive operations done n times in total
compare i < n	1 primitive operation done n+1 times in total

k ← k+1	2 primitive operations done n times in total
---------	--

$$1 + 2n + 1(n + 1) + 2n = 5n + 2 = O(n)$$

Approximation: for loop is executed about n times and there is one statement inside the for loop that runs in $O(1)$. Thus $O(n) * O(1) = O(n)$

Runtime Analysis – Nested Loops

Analyze these inside out. The total running time of a statement inside a group of nested loops is the running time of the statement multiplied by the product of the sizes of all the loops.

Example that is $O(N^2)$

```
for i ← 0 to n-1
    for j ← 0 to n-1
        some statements that run in constant time
```

Approximation

The inner “j-loop” executes n times

The outer “i-loop” executes n times

Some constant runtime statements execute in $O(1)$

$$n * n * O(1) = O(n^2)$$

Runtime Analysis – Consecutive Statements

These just add (which means that the maximum is the one that counts)

A rule of Big-Oh notation

If $T_1(N) = O(f(N))$ and $T_2(N) = O(g(N))$, then

(a) $T_1(N) + T_2(N) = O(f(N) + g(N))$ (intuitively and less formally it is $O(\max(f(N), g(N)))$),

```
for i ← 0 to n-1 do            $O(N)$ 
    k ← k+1
for i ← 0 to n-1
    for j ← 0 to n-1            $O(N^2)$ 
        some statements that run in constant time
```

Runtime is $O(N + N^2) = O(N^2)$

Runtime Analysis – If/Else

Running time of an if/else statement is never more than the running time of the test plus the larger of the running times of S1 and S2

```
if( condition ) then
    S1
else
    S2
```

Runtime Analysis – If/Else

Running time of an if/else statement is never more than the running time of the test plus the larger of the running times of S1 and S2

```
if( condition ) then
    x=1
else
    for i ← 1 to n
        x ← x+1
```

7

take $\max(O(S1) + O(S2) = O(1 + n) = O(n))$

Thank You !



Questions ?