Lecture 9: Describing and Evaluating Algorithms

Core 109S IDWT?, Spring 2017

Michael Hay

Algorithms

- Algorithm: a sequence of steps to accomplish a task
  - Making a sandwich
  - Multiplying two large numbers 1402 x 345
- Algorithm properties
  - Correctness
  - Resource usage: time, space, as a function of the *size* of the *input*

Terminology

- Procedure
  - Parameters to a procedure
  - Procedures are *called*; procedures *returns a* value
- Array
  - Index i
  - A[i] entry of array A at index i
  - Time to access entry i is the same for all i (if A in memory)

Algorithm 1 An algorithm for searching an array

procedure LinearSearch(A, n, x) ▷ Find x in array A of length n
    Set *answer* to Not-Found.
    Set i to 1.
    while $i \leq n$ **do**
        If $A[i] = x$, then set *answer* to the value of i.
        Increment i.
    end while
    return the value of *answer* as the output.
end procedure

Algorithm 2 A more efficient algorithm for searching an array

```
procedure SentinelLinearSearch(A, n, x)          ▷ Find x in array A of length n
    Save A[n] into last and then put x into A[n].
    Set i to 1.
    while A[i] ≠ x do
        Increment i.
    end while
    Restore A[n] from last.
    If i < n or A[n] = x, then return the vlaue of i as the output.
    Otherwise, return Not-Found as the output.
end procedure
```

Algorithm 1 An algorithm for searching an array

procedure LINEARSEARCH(A, n, x) $\triangleright$ Find x in array A of length n
    Set *answer* to NOT-FOUND.
    Set i to 1.
    while $i \leq n$ **do**
        If $A[i] = x$, then set *answer* to the value of i.
        Increment i.
    end while
    return the value of *answer* as the output.
end procedure

When is A2 more efficient?

Algorithm 2 A more efficient algorithm for searching an array

procedure SENTINELLINEARSEARCH(A, n, x) $\triangleright$ Find x in array A of length n
    Save $A[n]$ into *last* and then put x into $A[n]$.
    Set i to 1.
    while $A[i] \neq x$ **do**
        Increment i.
    end while
    Restore $A[n]$ from *last*.
    If $i < n$ or $A[n] = x$, then return the vlaue of i as the output.
    Otherwise, return NOT-FOUND as the output.
end procedure

Exercise 1

Instructions: Work in small groups of 3-4.

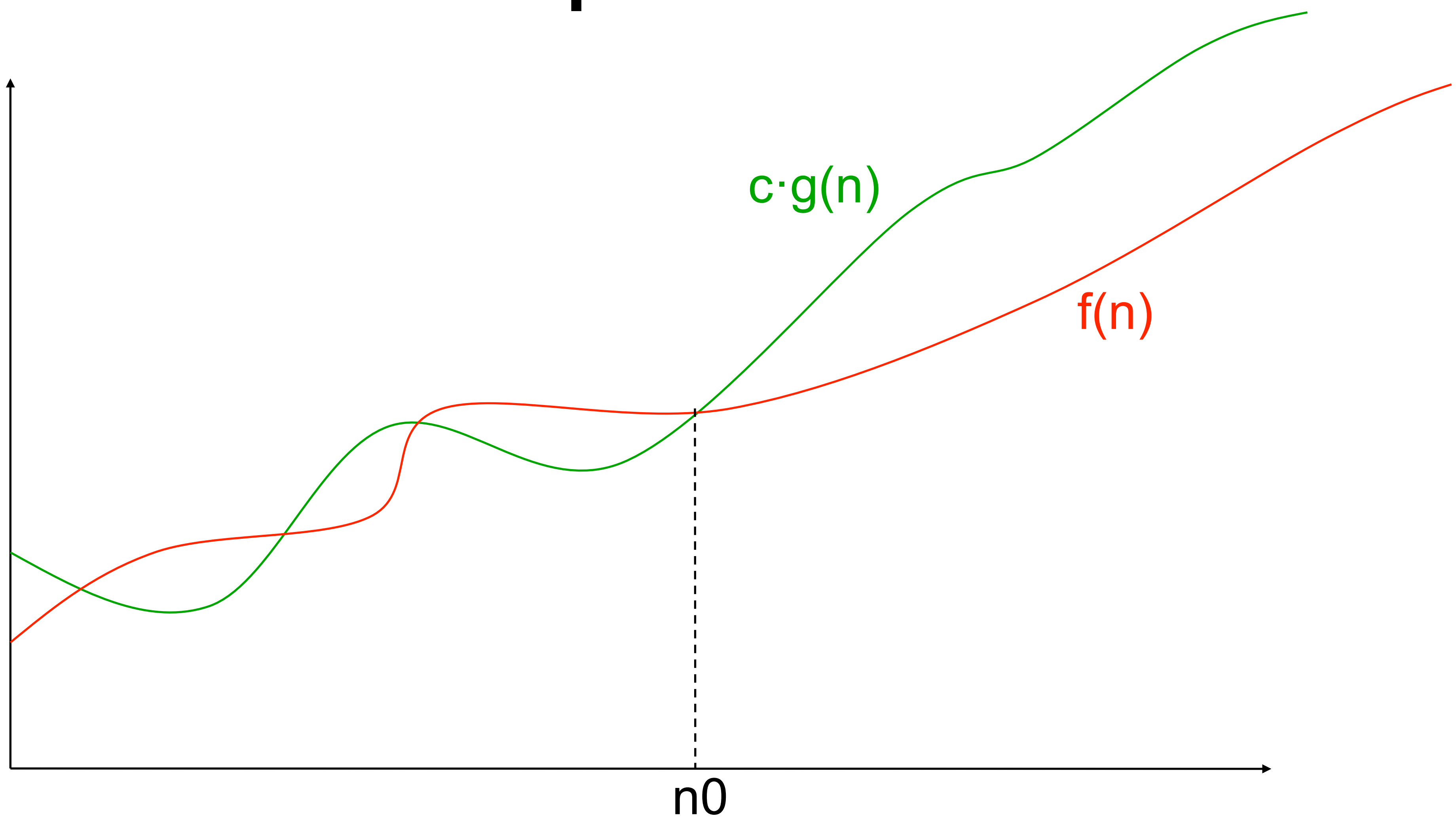
- Complete exercise 1 on your worksheet.

Using Big-O to Hide Constants

- We say f(n) is order of g(n) if f(n) is bounded by a constant times g(n)
- Notation: f(n) is O(g(n))
- Roughly, f(n) is O(g(n)) means that f(n) grows like g(n) or slower, to within a constant factor
- "Constant" means fixed and independent of n

We say that $f(n) = O(g(n))$ if there exist positive constants c and n_0 such that $0 \leq f(n) \leq c \cdot g(n)$ for all $n \geq n_0$.

A Graphical View



To prove that f(n) is O(g(n)):

Find an n0 and c such that f(n0) ≤ c g(n0) for all n ≥ n0

We call the pair (c, N) a *witness pair* for proving that f(n) is O(g(n))

Big-O Example

- **Claim**: Let $f(n) = 100\ n + lg\ n$ and $g(n) = n$. Claim is $f(n) = O(g(n))$
  - We know $\lg n \leq n$ for $n \geq 1$
  - So $100\ n + \lg n \leq 101\ n$ for $n \geq 1$
  - So by definition, $100\ n + \lg n$ is $O(n)$ for $c = 101$ and $N = 1$

Exercise 2.a.

Instructions: ~2 minute to think/answer on your own; then discuss with neighbors; then I will call on one of you

- Complete exercise 2.a. on your worksheet.

Exercise 2

Instructions: Work in small groups of 3-4.

- Working in groups, finish exercise 2 on your worksheet.

Problem-Size Examples

Suppose we have a computing device that can execute 1000 operations per second; how large a problem can we solve?

	1 second	1 minute	1 hour
n	1000	60,000	3,600,000
n log n	140	4893	200,000
n^2	31	244	1897
$3n^2$	18	144	1096
n^3	10	39	153
2^n	9	15	21

Commonly Seen Time Bounds

$O(1)$	constant	excellent
$O(\log n)$	logarithmic	excellent
$O(n)$	linear	good
$O(n \log n)$	n log n	pretty good
$O(n^2)$	quadratic	often OK
$O(n^3)$	cubic	maybe OK
$O(2^n)$	exponential	too slow

Algorithm 3 An algorithm for computing the cumulative sum

```
procedure CumulativeSum(A, n)    ▷ Constructs an array B where B[i] is the sum of
    the first i elements of the array A which has length n.
    Initialize B to an n-length array of zeros.
    Set i to 1.
    while i ≤ n do
        Set j to 1.
        Set total to 0.
        while j ≤ i do
            Add A[i] to total.
            Increment j.
        end while
        Set B[i] equal to total.
    end while
    return the array B.
end procedure
```

Exercise 3

Instructions: Work in small groups of 3-4.

- Working in groups, work on exercises 3 and 4 on your worksheet.

Improving search

- If we knew the array was sorted, how might we speed up search?