Lecture 09: The Efficiency of Algorithms

CS 0445: Data Structures

Constantinos Costa

http://db.cs.pitt.edu/courses/cs0445/current.term/

Sep 23, 2019, 8:00-9:15
University of Pittsburgh, Pittsburgh, PA

Why Efficient Code?

- Computers are faster, have larger memories
  - So why worry about efficient code?
- And … how do we measure efficiency?

Importance of Efficiency

- **Consider the problem of summing**

$$\sum_{k=1}^{n} k = 1 + 2 + 3 + \ldots + n$$

Algorithm A	Algorithm B	Algorithm C

`long sum = 0;`
`for (long i = 1; i <= n; i++)`
`   sum = sum + i;`
	`sum = 0;`
	`for (long i = 1; i <= n; i++)`
	`{`
	`   for (long j = 1; j <= i; j++)`
	`      sum = sum + 1;`
	`} // end for`
		`sum = n * (n + 1) / 2;`

- **Three algorithms for computing the sum 1 + 2 + . . . + n for an integer n > 0**

What is “best”?

- An algorithm has both time and space constraints – that is complexity
  - Time complexity
  - Space complexity
- This study is called analysis of algorithms

Counting Basic Operations

- A basic operation of an algorithm
  - Most significant contributor to its total time requirement

	Algorithm A	Algorithm B	Algorithm C

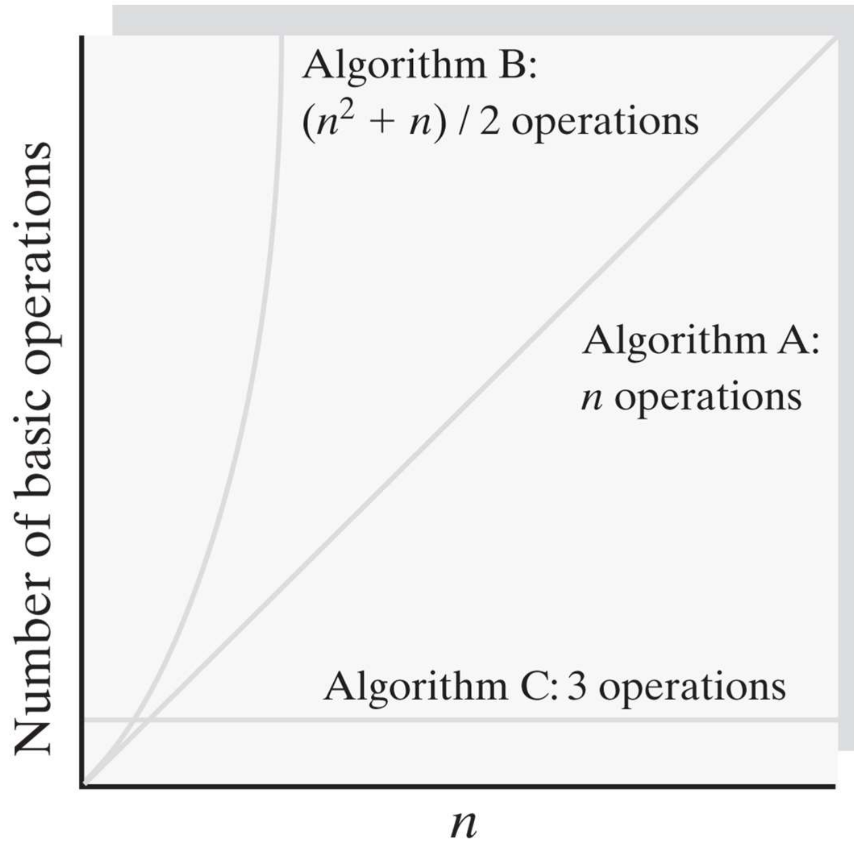
	`long sum = 0;`
	`for (long i = 1; i <= n; i++)`
	`   sum = sum + i;`
		`sum = 0;`
		`for (long i = 1; i <= n; i++)`
		`{`
		`   for (long j = 1; j <= i; j++)`
		`      sum = sum + 1;`
		`} // end for`
			`sum = n * (n + 1) / 2;`

Additons	n	$n(n + 1)/2$	1
Multiplications	0	0	1
Divisions	0	0	1
Total Basic Operations	n	$(n^2 + n)/2$	3

The number of basic operations required by the algorithms

Counting Basic Operations

- **Number of basic operations required by the algorithms as a function of *n***



Counting Basic Operations

- **Typical growth-rate functions evaluated at increasing values of *n***

n	(log(log n)	log n	$\log^2 n$	n	$n \log n$	n^2	n^3	2^n	$n!$
10	2	3	11	10	33	10^2	10^3	10^3	10^5
10^2	3	7	44	100	664	10^4	10^6	10^{30}	10^{94}
10^3	3	10	99	1,000	9,966	10^6	10^9	10^{301}	10^{1435}
10^4	4	13	177	10,000	132,877	10^8	10^{12}	10^{3010}	$10^{19,335}$
10^5	4	17	276	100,00	1,660,964	10^{10}	10^{15}	$10^{30,103}$	$10^{243,338}$
10^6	4	20	397	1,000,000	19,931,569	10^{12}	10^{18}	$10^{301,301}$	$10^{2,933,369}$

Best, Worst, and Average Cases

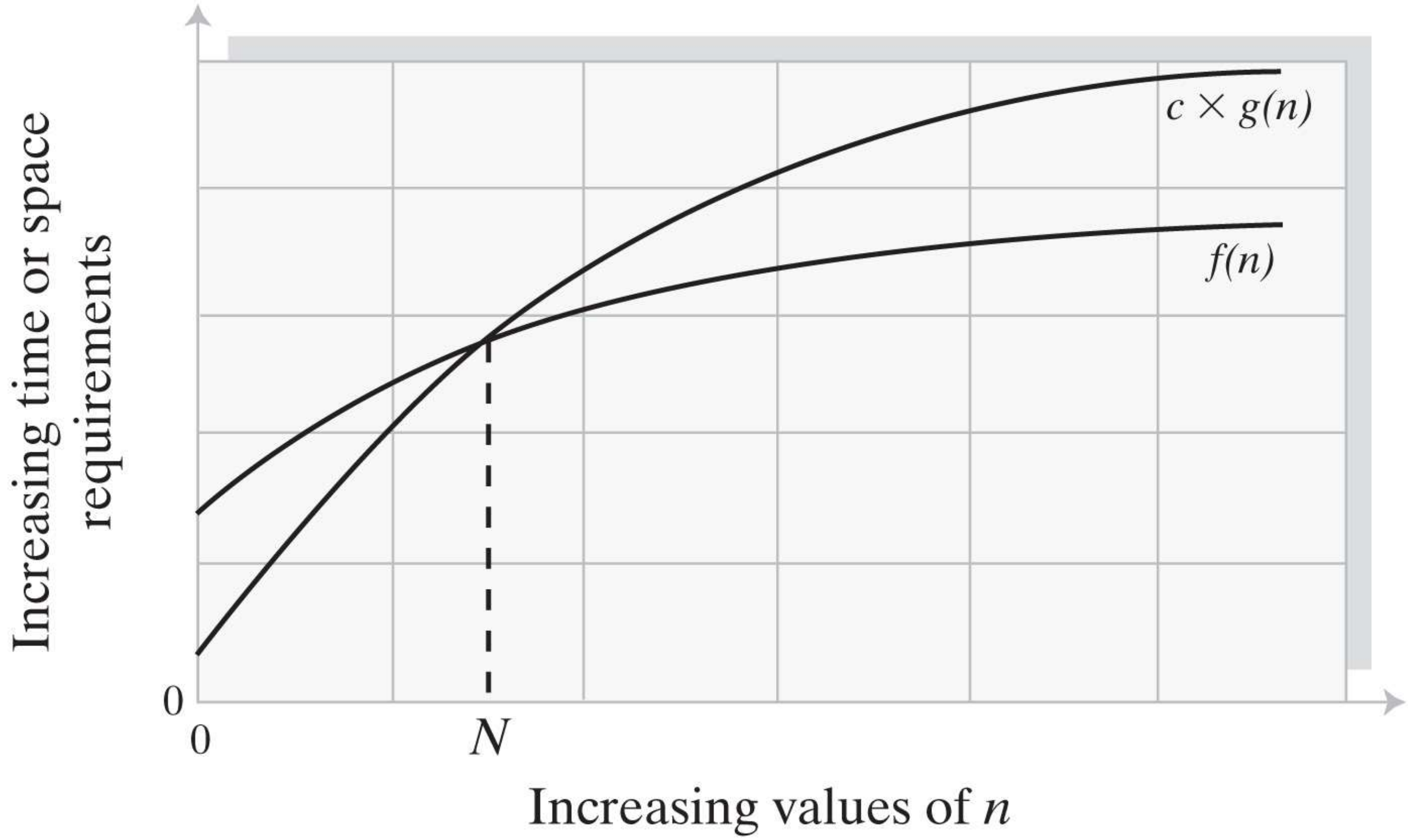
- For some algorithms, execution time depends only on size of data set
- Other algorithms depend on the nature of the data itself
  - Goal is to know best case, worst case, average case

Big Oh Notation

- A function $f(n)$ is of order at most $g(n)$
- That is, $f(n)$ is $O(g(n))$ — if
  - A positive real number c and positive integer N exist …
  - Such that $f(n) \leq c \times g(n)$ for all $n \geq N$
  - That is:
    - $c \times g(n)$ is an upper bound on $f(n)$ when n is sufficiently large

Big Oh Notation

- **An illustration of the values of two growth-rate functions**



Big Oh Notation

$O(k\ g(n)) = O(g(n))$ *for a constant k*

$O(g_1(n)) + O(g_2(n)) = O(g_1(n) + g_2(n))$

$O(g_1(n)) * O(g_2(n)) = O(g_1(n) * g_2(n))$

$$O(g_1(n) + g_2(n) + \ldots + g_m(n)) = O(\max(g_1(n), g_2(n), \ldots, g_m(n))$$

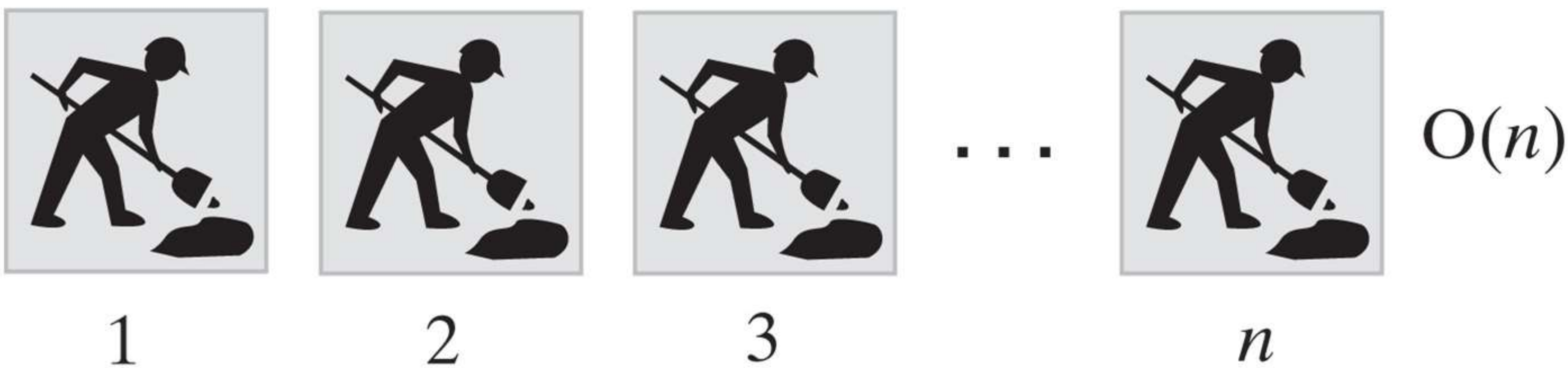
$$O(\max(g_1(n), g_2(n), \ldots, g_m(n)) = \max(O(g_1(n)), O(g_2(n)), \ldots, O(g_m(n)))$$

Identities for Big Oh Notation

Picturing Efficiency

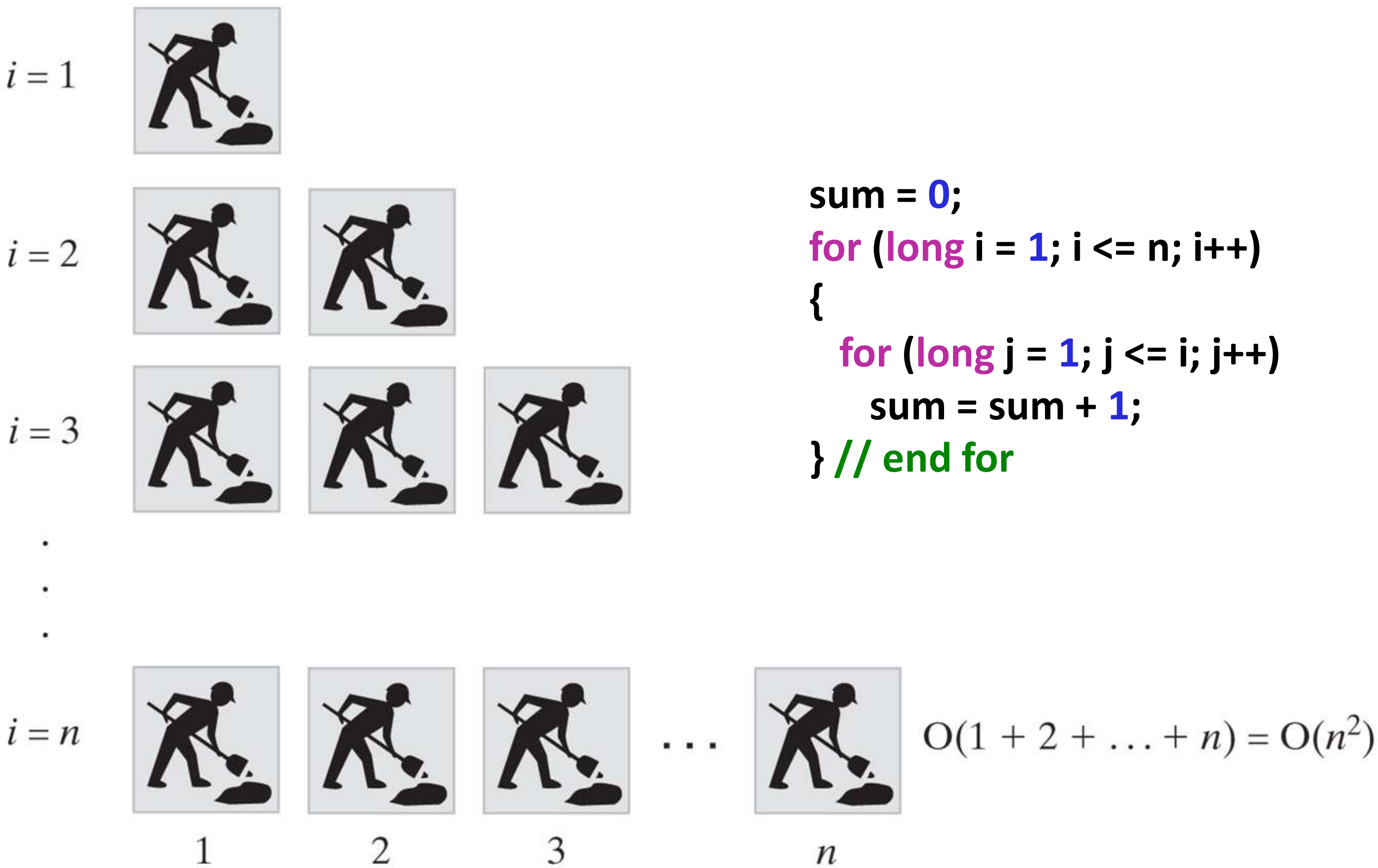
- **An O(n) algorithm**

```
long sum = 0;
for (long i = 1; i <= n; i++)
  sum = sum + i;
```



Picturing Efficiency

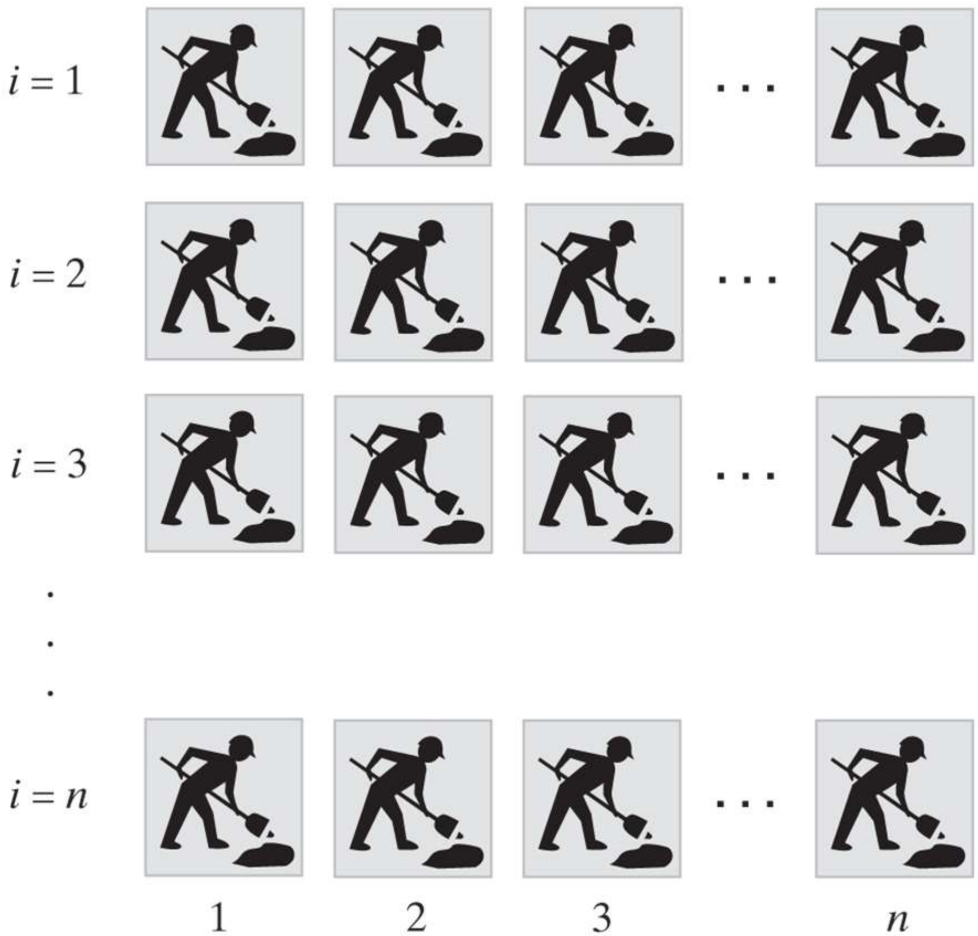
- **An *O(n²)* algorithm**




```
sum = 0;
for (long i = 1; i <= n; i++)
{
  for (long j = 1; j <= i; j++)
    sum = sum + 1;
} // end for
```

Picturing Efficiency

- **Another *O(n^2)* algorithm**




```
sum = 0;
for (long i = 1; i <= n; i++)
{
   for (long j = 1; j <= n; j++)
      sum = sum + 1;
} // end for
```

$$O(n \times n) = O(n^2)$$

Picturing Efficiency

The effect of doubling the problem size on an algorithm's time requirement

Growth-Rate Function for Size n Problems	Growth-Rate Function for Size $2n$ Problems	Effect on Time Requirement
1	1	None
$\log n$	$1 + \log n$	Negligible
n	$2n$	Doubles
$n \log n$	$2n \log n + 2n$	Doubles and then adds $2n$
n^2	$(2n)^2$	Quadruples
n^3	$(2n)^3$	Multiples by 8
2^n	2^{2n}	Squares

Picturing Efficiency

- **The time required to process one million items by algorithms of various orders at the rate of one million operations per second**

Growth-Rate Function g	$g(10^6) / 10^6$
$\log n$	0.0000199 seconds
n	1 second
$n \log n$	19.9 seconds
n^2	11.6 days
n^3	31,709.8 years
2^n	$10^{301,016}$ years

Efficiency of ADT Bag Implementations

- **The time efficiencies of the ADT bag operations for two implementations, expressed in Big Oh notation**

Operation	Fixed-Size Array	Linked
`add(newEntry)`	O(1)	O(1)
`remove()`	O(1)	O(1)
`remove(anEntry)`	O(1), O(n), O(n)	O(1), O(n), O(n)
`clear()`	O(n)	O(n)
`getFrequencyOf(anEntry)`	O(n)	O(n)
`contains(anEntry)`	O(1), O(n), O(n)	O(1), O(n), O(n)
`toArray()`	O(n)	O(n)
`getCurrentSize(), isEmpty()`	O(1)	O(1)