Big-O and Sorting

April 2–3, 2012

CS 60: Principles of Computer Science

Assignment 8 (Full Unicalc) due Monday, April 2.

RECALL: $O(\cdots)$ NOTATION

Upper bound

Ignores constant factors

Assumes input gets arbitrarily big

In theory, $O()$ tells us very little.

In practice, the hidden constants are often small, and so we can draw reasonable conclusions about scalability.

Running Times

(or why people worry about algorithm complexity)

problem size →

complexity ↓	**n =** 10	100	1,000	10,000	100,000	1,000,000
log n	3.3219	6.6438	9.9658	13.287	16.609	19.931
$\log^2 n$	10.361	44.140	99.317	176.54	275.85	397.24
sqrt n	3.162	10	31.622	100	316.22	1000
n	**10**	**100**	**1000**	**10000**	**100000**	**1000000**
n log n	33.219					
$n^{1.5}$	31.6					
n^2	100					
n^3	1000					
2^n	1024					

O

imagine the time units ~ nanoseconds: 10**(-9)

Running Times

(or why people worry about algorithm complexity)

problem size →

	complexity	n = 10	100	1,000	10,000	100,000	1,000,000
Polynomial	log n	3.3219	6.6438	9.9658	13.287	16.609	19.931
	$\log^2 n$	10.361	44.140	99.317	176.54	275.85	397.24
	sqrt n	3.162	10	31.622	100	316.22	1000
	n	**10**	**100**	**1000**	**10000**	**100000**	**1000000**
	n log n	33.219	664.38	9965.8	132877	$1.66*10^6$	$1.99*10^7$
	$n^{1.5}$	31.6	10^3	$31.6*10^4$	10^6	$31.6*10^7$	10^9
	n^2	100	10^4	10^6	10^8	10^{10}	10^{12}
	n^3	1000	10^6	10^9	10^{12}	10^{15}	10^{18}
Exp.	2^n	1024	10^{30}	10^{301}	10^{3010}	10^{30103}	10^{301030}

factorial? don't even ask!

O

imagine the time units ~ nanoseconds: 10**(-9)

Common Series in $O()$-Land

How many terms are here? What's the $O()$ sum?

$$1 + 2 + 4 + 8 + \cdots + 2^{n-2} + 2^{n-1} + 2^{n}$$

How many terms are here? What's the $O()$ sum?

$$1 + 2 + 4 + 8 + \cdots + n/4 + n/2 + n$$

How many terms are here? What's the $O()$ sum?

$$1 + 2 + 3 + 4 + \cdots + (n-2) + (n-1) + n$$

Common Series in O()-Land

How many terms are here? What's the O() sum?

$$1 + 2 + 4 + 8 + \cdots + 2^{n-2} + 2^{n-1} + 2^n = 2^{n+1} - 1 = O(2^n)$$

How many terms are here? What's the O() sum?

$$1 + 2 + 4 + 8 + \cdots + n/4 + n/2 + n$$

How many terms are here? What's the O() sum?

$$1 + 2 + 3 + 4 + \cdots + (n-2) + (n-1) + n$$

Common Series in O()-Land

How many terms are here? What's the $O()$ sum?

$$1 + 2 + 4 + 8 + \cdots + 2^{n-2} + 2^{n-1} + 2^n = 2^{n+1} - 1 = O(2^n)$$

How many terms are here? What's the $O()$ sum?

$$1 + 2 + 4 + 8 + \cdots + n/4 + n/2 + n = 2n - 1 = O(n)$$

How many terms are here? What's the $O()$ sum?

$$1 + 2 + 3 + 4 + \cdots + (n-2) + (n-1) + n$$

Common Series in $O()$-Land

How many terms are here? What's the $O()$ sum?

$$1 + 2 + 4 + 8 + \cdots + 2^{n-2} + 2^{n-1} + 2^n = 2^{n+1} - 1 = O(2^n)$$

How many terms are here? What's the $O()$ sum?

$$1 + 2 + 4 + 8 + \cdots + n/4 + n/2 + n = 2n - 1 = O(n)$$

How many terms are here? What's the $O()$ sum?

$$1 + 2 + 3 + 4 + \cdots + (n-2) + (n-1) + n = \frac{n(n+1)}{2} = O(n^2)$$

Loops and O()

- ✓ What are the possible values of the loop index?
- ✓ For each, how much work is being done inside?
- ✓ What is the O() running time of each loop?

```
for (int i = 0; i < N; ++i)
  ...1 step of work here...
```

```
for (int i = N; i > 0; i /= 2)
  ...1 step of work here...
```

Loops and O()

- ✓ What are the possible values of the outer loop index?
- ✓ For each, how much work is being done inside?
- ✓ What is the O() running time of each loop?

```
for ( int i=0; i<N; ++i )
  for ( int j=0; j<N; ++j )
    ...1 time step of work here...
```

```
for ( int i=1; i<=N; ++i )
  for ( int j=1; j<=i; ++j )
    ...1 time step of work here...
```

```
for ( int i=1; i<=N; i*=2 )
  for ( int j=0; j<i; j++ )
    ...1 time step of work here ...
```

NAME: "QUIZ"

Compute the asymptotic running time of:

```
for (int i=1; i<=N; i*=2 )
  for (int j=1; j<i; j*=2 )
    ... O(1) work here ...
```

Sorting Algs: The "Fruit flies" of Complexity

In the worst-case, how many *comparisons* might the following sorting algorithms perform, given an array of n inputs?

A `minsort` repeatedly find the minimum; assemble sorted list

B `mergesort` mergesort each half; merge sorted halves.

C `cs5sort`

```
def cs5sort(L):
   if len(L) < 2:
      return L
   if L[0] == min(L):
      return [L[0]] + cs5sort(L[1:])
   return cs5sort(L[1:] + L[0])
```

Sorting Algs: The "Fruit flies" of Complexity

In the worst-case, how many *comparisons* might the following sorting algorithms perform, given an array of n inputs?

A	`minsort`	repeatedly find the minimum; assemble sorted list	$O(n^2)$
B	`mergesort`	mergesort each half; merge sorted halves.	
C	`cs5sort`		

```
def cs5sort(L):
  if len(L) < 2:
     return L
  if L[0] == min(L):
     return [L[0]] + cs5sort(L[1:])
  return cs5sort(L[1:] + L[0])
```

Sorting Algs: The "Fruit flies" of Complexity

In the worst-case, how many *comparisons* might the following sorting algorithms perform, given an array of n inputs?

A	`minsort`	repeatedly find the minimum; assemble sorted list	$O(n^2)$
B	`mergesort`	mergesort each half; merge sorted halves.	$O(n \log n)$
C	`cs5sort`		

```
def cs5sort(L):
  if len(L) < 2:
     return L
  if L[0] == min(L):
     return [L[0]] + cs5sort(L[1:])
  return cs5sort(L[1:] + L[0])
```

Sorting Algs: The "Fruit flies" of Complexity

In the worst-case, how many *comparisons* might the following sorting algorithms perform, given an array of n inputs?

- A `minsort` — repeatedly find the minimum; assemble sorted list — $O(n^2)$
- B `mergesort` — mergesort each half; merge sorted halves. — $O(n \log n)$
- C `cs5sort` — (n^3)

```
def cs5sort(L):
  if len(L) < 2:
     return L
  if L[0] == min(L):
     return [L[0]] + cs5sort(L[1:])
  return cs5sort(L[1:] + L[0])
```

Sorting Algs: The "fruit flies" of Complexity

In the worst-case, how many *comparisons* might the following sorting algorithms perform, given an array of n inputs?

D	`prologsort`	try all permutations in order; check each for being sorted.
E	`bogosort`	repeatedly pick random permutations; check each for being sorted.
F	`stoogesort`	recurse on first 2/3; recurse on last 2/3; recurse on first 2/3.

Sorting Algs: The "fruit flies" of Complexity

In the worst-case, how many *comparisons* might the following sorting algorithms perform, given an array of n inputs?

D	`prologsort`	*try all permutations in order; check each for being sorted.*	$O(n!)$
E	`bogosort`	*repeatedly pick random permutations; check each for being sorted.*	
F	`stoogesort`	*recurse on first 2/3; recurse on last 2/3; recurse on first 2/3.*	

Sorting Algs: The "fruit flies" of Complexity

In the worst-case, how many *comparisons* might the following sorting algorithms perform, given an array of n inputs?

D	`prologsort`	try all permutations in order; check each for being sorted.	$O(n!)$
E	`bogosort`	repeatedly pick random permutations; check each for being sorted.	$O(n!)$
F	`stoogesort`	recurse on first 2/3; recurse on last 2/3; recurse on first 2/3.	

Sorting Algs: The "fruit flies" of Complexity

In the worst-case, how many *comparisons* might the following sorting algorithms perform, given an array of n inputs?

D	`prologsort`	try all permutations in order; check each for being sorted.	$O(n!)$
E	`bogosort`	repeatedly pick random permutations; check each for being sorted.	$O(n!)$
F	`stoogesort`	recurse on first 2/3; recurse on last 2/3; recurse on first 2/3.	$\approx O(n^{2.7})$

Insertion Sort

4 7 5 2 1 3 0 6

Idea:

1. Maintain a "sorted part" at the start.
2. Repeatedly extend this sorted part by moving one more number to its proper location.

Worst-case running time? Best-case running time?