`http://www.cs.unh.edu/~ruml/cs758`

1 handout: slides

Previously On CS 758...

O()

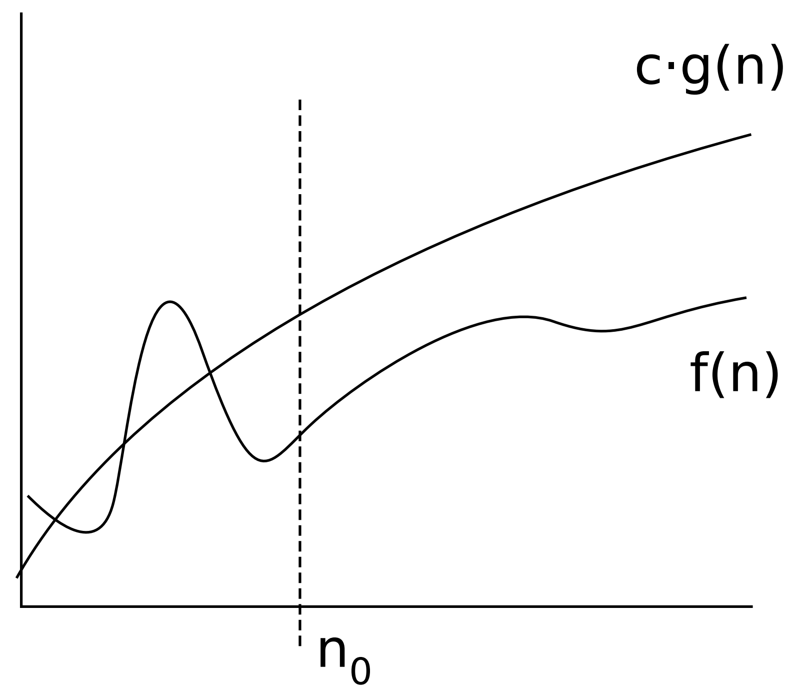
$$O(g(n)) = \{f(n) : \text{there exist positive constants } c, n_0 \text{ such that } f(n) \leq cg(n) \text{ for all } n \geq n_o\}$$

ignore constant factors
ignore 'start-up costs'
upper bound

We can upper-bound f (except perhaps at start) by scaling g by a constant.

eg, running time of
$10n^2 - 5n = O(n^2)$

f(n) = O(g(n))



O() Example

$$10n^2 + 5n = \Theta(n^2)$$

$$10n \lg \frac{n}{e} = O(n \lg n)$$

Counting Sort

For n numbers in the range 0 to k:

1. for x from 0 to k
2.   count$[x] \leftarrow 0$
3. for each input number x
4.   increment count$[x]$
5. for x from 0 to k
6.   do count$[x]$ times
7.   emit x

Counting Sort

For n numbers in the range 0 to k:

1.	for x from 0 to k	$O(k)$
2.	count$[x] \leftarrow 0$	
3.	for each input number x	$O(n)$
4.	increment count$[x]$	
5.	for x from 0 to k	$O(k)$ times around loop
6.	do count$[x]$ times	iterates $O(n)$ times total
7.	emit x	$O(1)$ each time

$$O(k + n + k + n) = O(2n + 2k) = O(n + k) \neq O(n \lg n)$$

Radix Sort

Stable Counting Sort

Input array contains n **records** with keys in the range 0 to $k - 1$

Stable Counting Sort

Input array contains n **records** with keys in the range 0 to $k-1$

1. set count[x] to number of items with key $= x$
2. set pos[x] to total number of keys $< x$
3. for each input record r (in order)
4. write r in output array at position pos[key of r]
5. increment pos[key of r]

Complexity?
Invariants?

Radix Sort

How to sort one million records?

How to sort one million records?

How to sort one trillion 4-bit integers?

Radix Sort

How to sort one million records?

How to sort one trillion 4-bit integers?

How to sort one billion 16-bit integers?

Radix Sort

How to sort one million records?

How to sort one trillion 4-bit integers?

How to sort one billion 16-bit integers?

How to sort one billion 64-bit integers?

Radix Sort

How to sort one million records?

How to sort one trillion 4-bit integers?

How to sort one billion 16-bit integers?

How to sort one billion 64-bit integers?

For n numbers with d digits (each digit has k values):

How to sort one million records?

How to sort one trillion 4-bit integers?

How to sort one billion 16-bit integers?

How to sort one billion 64-bit integers?

For n numbers with d digits (each digit has k values):

1. for i from 0 to *d*
2. stable sort on digit in place i from right

Analysis

What's the invariant in radix sort?

Complexity

What's the space complexity?

What's the time complexity?

Limitations

Why not implemented more?

Break

- everyone receiving `piazza` notifications?
- book access?

  see book for example proofs
- asst 1: `agate`, `valgrind`, submit, happy TA
- no hardcopy submission
- probabilistic grading
- schedule: asst 1, 2, 3

More Sorting Algorithms

Insertion Sort

for i from 2 to n
    move $A[i]$ earlier until in place

worse case?

best case?

'divide and conquer': divide, combine, and conquer

Mergesort(A, i, j)
1. if $i \geq j$, done
2. k $\leftarrow (i + j)/2$
3. Mergesort(A, i, k)
4. Mergesort$(A, k + 1, j)$
5. merge $A[i..k]$ and $A[k + 1..j]$ into $A[i..j]$

how does merge work?
running time?

divide, conquer, and combine

Quicksort(A, i, j)

1. choose pivot key x
2. partition $A[i..j]$ into $A[i..p-1]$ and $A[p+1..j]$
3. if $p-1 > i$ then Quicksort$(A, i, p-1)$
4. if $j > p+1$ then Quicksort$(A, p+1, j)$

divide, conquer, and combine

Quicksort(A, i, j)

1. choose pivot key x
2. partition $A[i..j]$ into $A[i..p-1]$ and $A[p+1..j]$
3. if $p-1 > i$ then Quicksort$(A, i, p-1)$
4. if $j > p+1$ then Quicksort$(A, p+1, j)$

+:

entirely in-place, no allocation
often less copying than merge sort

−:

expected $O(n \lg n)$
needs tricks to avoid worst case

Partition(A, i, j)

1. choose pivot key p and swap into $A[j]$
2. $x = i$
3. for $y = i$ to $j - 1$
4. if $A[y] \leq p$
5. swap $A[x]$ and $A[y]$
6. $x \leftarrow x + 1$
7. swap $A[x]$ and $A[j]$

A: $(i{:})$ less $(x{:})$ greater $(y{:})$ unknown $(j{:})$ pivot

Lower Bounds

What is the minimum that a sorting algorithm must do?

Lower Bounds

What is the minimum that a sorting algorithm must do?

How many possible outputs are there for sorting n items?

What is the minimum that a sorting algorithm must do?

How many possible outputs are there for sorting n items?

binary tree with $n!$ leaves

What is the minimum that a sorting algorithm must do?

How many possible outputs are there for sorting n items?

binary tree with $n!$ leaves has height at least $\lg(n!)$

What is the minimum that a sorting algorithm must do?

How many possible outputs are there for sorting n items?

binary tree with $n!$ leaves has height at least $\lg(n!)$

Stirling: $n! = \sqrt{2\pi n}(\frac{n}{e})^n(1 + \Theta(\frac{1}{n}))$

What is the minimum that a sorting algorithm must do?

How many possible outputs are there for sorting n items?

binary tree with $n!$ leaves has height at least $\lg(n!)$

Stirling: $n! = \sqrt{2\pi n}(\frac{n}{e})^n(1 + \Theta(\frac{1}{n}))$
so:

$$\lg(n!) = \lg(\sqrt{2\pi n}(\frac{n}{e})^n(1 + \Theta(\frac{1}{n})))$$

What is the minimum that a sorting algorithm must do?

How many possible outputs are there for sorting n items?

binary tree with $n!$ leaves has height at least $\lg(n!)$

Stirling: $n! = \sqrt{2\pi n}(\frac{n}{e})^n(1+\Theta(\frac{1}{n}))$
so:

$$\begin{aligned} \lg(n!) &= \lg(\sqrt{2\pi n}(\frac{n}{e})^n(1+\Theta(\frac{1}{n}))) \\ &= \lg\sqrt{2\pi} + \lg\sqrt{n} + \lg(\frac{n}{e})^n + lg(1+\Theta(\frac{1}{n})) \end{aligned}$$

What is the minimum that a sorting algorithm must do?

How many possible outputs are there for sorting n items?

binary tree with $n!$ leaves has height at least $\lg(n!)$

Stirling: $n! = \sqrt{2\pi n}(\frac{n}{e})^n(1+\Theta(\frac{1}{n}))$
so:

$$\begin{aligned}
\lg(n!) &= \lg(\sqrt{2\pi n}(\frac{n}{e})^n(1+\Theta(\frac{1}{n}))) \\
&= \lg\sqrt{2\pi} + \lg\sqrt{n} + \lg(\frac{n}{e})^n + lg(1+\Theta(\frac{1}{n})) \\
&= \Theta(\lg\sqrt{n} + n\lg(\frac{n}{e})) + lg(1+\Theta(\frac{1}{n}))
\end{aligned}$$

What is the minimum that a sorting algorithm must do?

How many possible outputs are there for sorting n items?

binary tree with $n!$ leaves has height at least $\lg(n!)$

Stirling: $n! = \sqrt{2\pi n}(\frac{n}{e})^n(1 + \Theta(\frac{1}{n}))$
so:

$$\begin{aligned}
\lg(n!) &= \lg(\sqrt{2\pi n}(\frac{n}{e})^n(1+\Theta(\frac{1}{n}))) \\
&= \lg\sqrt{2\pi} + \lg\sqrt{n} + \lg(\frac{n}{e})^n + lg(1+\Theta(\frac{1}{n})) \\
&= \Theta(\lg\sqrt{n} + n\lg(\frac{n}{e})) + lg(1+\Theta(\frac{1}{n})) \\
&= \Theta(\lg\sqrt{n} + n\lg n - n\lg e)) + lg(1+\Theta(\frac{1}{n}))
\end{aligned}$$

What is the minimum that a sorting algorithm must do?

How many possible outputs are there for sorting n items?

binary tree with $n!$ leaves has height at least $\lg(n!)$

Stirling: $n! = \sqrt{2\pi n}(\frac{n}{e})^n(1+\Theta(\frac{1}{n}))$
so:

$$\begin{aligned}
\lg(n!) &= \lg(\sqrt{2\pi n}(\frac{n}{e})^n(1+\Theta(\frac{1}{n}))) \\
&= \lg\sqrt{2\pi} + \lg\sqrt{n} + \lg(\frac{n}{e})^n + lg(1+\Theta(\frac{1}{n})) \\
&= \Theta(\lg\sqrt{n} + n\lg(\frac{n}{e})) + lg(1+\Theta(\frac{1}{n})) \\
&= \Theta(\lg\sqrt{n} + n\lg n - n\lg e)) + lg(1+\Theta(\frac{1}{n})) \\
&= \Theta(n\lg n)
\end{aligned}$$

What is the minimum that a sorting algorithm must do?

How many possible outputs are there for sorting n items?

binary tree with $n!$ leaves has height at least $\lg(n!)$

Stirling: $n! = \sqrt{2\pi n}(\frac{n}{e})^n(1+\Theta(\frac{1}{n}))$
so:

$$\begin{aligned}
\lg(n!) &= \lg(\sqrt{2\pi n}(\frac{n}{e})^n(1+\Theta(\frac{1}{n}))) \\
&= \lg\sqrt{2\pi} + \lg\sqrt{n} + \lg(\frac{n}{e})^n + lg(1+\Theta(\frac{1}{n})) \\
&= \Theta(\lg\sqrt{n} + n\lg(\frac{n}{e})) + lg(1+\Theta(\frac{1}{n})) \\
&= \Theta(\lg\sqrt{n} + n\lg n - n\lg e)) + lg(1+\Theta(\frac{1}{n})) \\
&= \Theta(n\lg n)
\end{aligned}$$

so comparison-based sorting takes $\Omega(n\lg n)$ time

EOLQs

- What's still confusing?
- What question didn't you get to ask today?
- What would you like to hear more about?

Please write down your most pressing question about algorithms and put it in the box on your way out.
Thanks!