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

1 handout: slides
Radix Sort

- Counting Sort
- O()
- And Friends
- O() Example
- Stable Counting
- Radix Sort

Analysis

Radix 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

\[ O(k + n + k + n) = O(2n + 2k) = O(n + k) \neq O(n \lg n) \]
\[ O(g(n)) = \{ f(n) : \text{there exist positive constants } c, n_0 \]
\[ \text{such that } f(n) \leq cg(n) \text{ for all } n \geq n_0 \} \]

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 \( 10n^2 - 5n = O(n^2) \)
Upper bound ('order of'):
\[ O(g(n)) = \{ f(n) : \text{there exist positive constants } c, n_0 \]
\[ \text{such that } f(n) \leq cg(n) \text{ for all } n \geq n_o \} \]

Lower bound:
\[ \Omega(g(n)) = \{ f(n) : \text{there exist positive constants } c, n_0 \]
\[ \text{such that } cg(n) \leq f(n) \text{ for all } n \geq n_o \} \]

Tight bound:
\[ \Theta(g(n)) = \{ f(n) : \text{there exist positive constants } c_1, c_2, n_0 \]
\[ \text{such that } c_1g(n) \leq f(n) \leq c_2g(n) \text{ for all } n \geq n_o \} \]
10n^2 + 5n = \Theta(n^2)

10n \log \frac{n}{e} = O(n \log n)
Stable Counting Sort

Input array contains \( n \) records with keys in the range 0 to \( k \)

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

Complexity?

Invariants?
How to sort one million records?
Radix Sort

How to sort one million records?

How to sort one trillion \textit{4-bit integers}?
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?
What’s the space complexity?
What’s the time complexity?
Why not implemented more?
■ Break

Radix Sort
Analysis
■ Correctness
■ Complexity
■ Limitations
■ Break
■ Lower Bounds
■ Quick Sort
■ Insertion Sort
■ EOLQs

■ did everyone get piazza email?
■ trouble getting books?
■ asst 1: agate, valgrind, happy Bence
What is the minimum that a sorting algorithm must do?
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?

A 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: $n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n (1 + \Theta\left(\frac{1}{n}\right))$
What is the minimum that a sorting algorithm must do? How many possible outputs are there for sorting $n$ items?

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

**Stirling**:

\[ n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n (1 + \Theta\left(\frac{1}{n}\right)) \]

So:

\[
\lg(n!) = \lg\left(\sqrt{2\pi n} \left(\frac{n}{e}\right)^n (1 + \Theta\left(\frac{1}{n}\right))\right)
\]
What is the minimum that a sorting algorithm must do?

How many possible outputs are there for sorting $n$ items?

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

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

So:

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

$$= \lg \sqrt{2\pi} + \lg n + \lg \left(\frac{n}{e}\right)^n + \lg(1 + \Theta\left(\frac{1}{n}\right))$$
What is the minimum that a sorting algorithm must do? How many possible outputs are there for sorting \( n \) items? A binary tree with \( n! \) leaves has height at least \( \lg(n!) \).

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

so:

\[
\begin{align*}
\lg(n!) &= \lg\left(\sqrt{2\pi n} \left( \frac{n}{e} \right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right)\right) \\
&= \lg \sqrt{2\pi} + \lg \sqrt{n} + \lg\left( \frac{n}{e} \right)^n + \lg\left(1 + \Theta\left(\frac{1}{n}\right)\right) \\
&= \Theta\left(\lg \sqrt{n} + n \lg\left( \frac{n}{e} \right) + \lg(1 + \Theta\left(\frac{1}{n}\right))\right)
\end{align*}
\]
What is the minimum that a sorting algorithm must do?

How many possible outputs are there for sorting $n$ items?

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

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

So:

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

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

$$= \Theta(\lg n + n \lg \left(\frac{n}{e}\right)) + \lg \left(1 + \Theta\left(\frac{1}{n}\right)\right)$$

$$= \Theta(n \lg n)$$
What is the minimum that a sorting algorithm must do? How many possible outputs are there for sorting \( n \) items?

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

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

so:

\[
\lg(n!) = \lg(\sqrt{2\pi n} \left(\frac{n}{e}\right)^n (1 + \Theta(\frac{1}{n}))) \\
= \lg \sqrt{2\pi} + \lg \sqrt{n} + \lg \left(\frac{n}{e}\right)^n + \lg(1 + \Theta(\frac{1}{n})) \\
= \Theta(\lg \sqrt{n} + n \lg \left(\frac{n}{e}\right) + \lg(1 + \Theta(\frac{1}{n}))) \\
= \Theta(n \lg n)
\]

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

Radix Sort

Analysis
- Correctness
- Complexity
- Limitations
- Break
- Lower Bounds

Quick Sort
- Insertion Sort
- EOLQs

in-place
potentially less copying than mergesort
expected \( O(n \lg n) \)
needs tricks to avoid worst case
Insertion Sort

how fast if almost sorted?
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!