Radix Sort

Analysis

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

1 handout: slides
Radix Sort

- Counting Sort
- $O(n)$
- $O(k)$ Example
- Stable Counting
- Radix Sort

Analysis
For $n$ numbers in the range 0 to $k$:

1. for $x$ from 0 to $k$
2. \( \text{count}[x] \leftarrow 0 \)
3. for each input number $x$
4. increment \( \text{count}[x] \)
5. for $x$ from 0 to $k$
6. do \( \text{count}[x] \) times
7. emit $x$
Counting Sort

For \( n \) numbers in the range 0 to \( k \):

1. for \( x \) from 0 to \( k \) \( \quad O(k) \)
2. \( \text{count}[x] \leftarrow 0 \)
3. for each input number \( x \) \( \quad O(n) \)
4. increment \( \text{count}[x] \)
5. for \( x \) from 0 to \( k \) \( \quad O(k) \) times around loop
6. do \( \text{count}[x] \) times \( \quad \) iterates \( O(n) \) times total
7. emit \( x \) \( \quad O(1) \) 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 \( 10n^2 - 5n = O(n^2) \)
\[ 10n^2 + 5n = \Theta(n^2) \]

\[ 10n \log \frac{n}{e} = O(n \log n) \]
Input array contains $n$ records with keys in the range 0 to $k$. 
Input array contains $n$ records with keys in the range 0 to $k$

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?
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?
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?
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):
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):

1. for \( i \) from 0 to \( d \)
2. stable sort on digit in place \( i \) from right
Analysis

<table>
<thead>
<tr>
<th>Correctness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complexity</td>
</tr>
<tr>
<td>Limitations</td>
</tr>
<tr>
<td>Break</td>
</tr>
<tr>
<td>Insertion Sort</td>
</tr>
<tr>
<td>Merge Sort</td>
</tr>
<tr>
<td>Quicksort</td>
</tr>
<tr>
<td>Partition</td>
</tr>
<tr>
<td>Lower Bounds</td>
</tr>
<tr>
<td>EOLQs</td>
</tr>
</tbody>
</table>

Analysis
What’s the invariant in radix sort?
What’s the space complexity?
What’s the time complexity?
Why not implemented more?
everyone receiving piazza notifications?
books available?
asst 1: agate, valgrind, happy Alison
for $i$ from 2 to $n$
move $A[i]$ earlier until in place

worse case?
better case?
‘divide and conquer’: divide, conquer, combine

**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, 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, 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 ($y:$) greater ($j:$) unknown ($x:$) pivot
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
Lower Bounds

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 \( \log(n!) \).
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 $\log(n!)$.

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

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

So:

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

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

So:

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

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

\[
= \Theta(\lg \sqrt{n} + n \lg \left(\frac{n}{e}\right)) + \lg(1 + \Theta(\frac{1}{n}))
\]
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:

$$\begin{align*}
\lg(n!) &= \lg\left(\sqrt{2\pi n} \left(\frac{n}{e}\right)^n (1 + \Theta\left(\frac{1}{n}\right))\right) \\
&= \lg \sqrt{2\pi} + \lg \sqrt{n} + \lg \left(\frac{n}{e}\right)^n + \lg(1 + \Theta\left(\frac{1}{n}\right)) \\
&= \Theta(\lg \sqrt{n} + n \lg \frac{n}{e}) + \lg(1 + \Theta\left(\frac{1}{n}\right)) \\
&= \Theta(n \lg n)
\end{align*}$$
Lower Bounds

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:

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

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