CS 2230
CS II: Data structures
Limits of comparison sorting, beyond comparison sorting
Brandon Myers
University of Iowa
ACE
Today’s learning objectives

• Calculate the number of comparisons needed to find a sort
• Execute sorting algorithms that are not comparison-based
• Calculate running time of a radix sort
Are $O(n^2)$ sorts like insertion sort/selection sort useless?

We’ve got $O(n \log n)$ sorts heap sort and merge sort.
Are $O(n^2)$ sorts like insertion sort/selection sort useless?

We’ve got $O(n\log n)$ sorts heap sort and merge sort.

No!
we saw insertion sort was best case $O(n)$
and, a $O(n\log n)$ implementation with a large constant is slower than a $O(n^2)$ implementation for small $n$
Lower bound on running time of comparison-based sorting
Peer instruction

You have three mystery blocks that you need to sort from lightest to heaviest.

A  B  C

The twist: you can only weigh two at a time on the scales

How many weighs do you need to be sure?

https://b.socrative.com/login/student/
room CS2230X ids 1000-2999
room CS2230Y ids 3000+
Lower bound on running time of comparison-based sorting?

we can abstract all comparison sorting algorithms to a tree of comparisons (ignore all other work the algorithm does)

Each leaf in the tree represents one of the possible initial orders (permutations) of the collection
Example comparison sorting tree

Consider the collection of numbers 1, 2, 3. There are $3! = 6$ permutations.

At each internal node, I ask the question is $a < b$?
Example comparison sorting tree

or, weighing three opaque boxes two at a time to sort them from light to heavy

A < B?
  yes
  no

B < C?
  yes
  no

C < A?
  yes
  no

A < B?
  yes
  no

B < C?
  yes
  no

C < A?
  yes
  no

A < B?
  yes
  no
Lower bound on running time of comparison-based sorting?

The height of this tree is the shortest path an algorithm could take to comparison sort the permutation.

The number of possible initial orders (permutations) is $n!$.

Therefore, the minimum height of such a tree is $\log_2(n!)$.
Lower bound on running time of comparison-based sorting?

Minimum height of such a tree is \( \log_2(n!) \)

\[
\log_2(n!) = \log_2(n \cdot (n - 1) \cdot \ldots \cdot 2 \cdot 1) = \log_2 n + \log_2 (n - 1) + \ldots + \log_2 1
\]

\[
= \sum_{i=1}^{n} \log_2(i)
\]

\[
\geq \frac{n}{2} \log_2 \frac{n}{2}
\]

\[
\in \Omega(n \log n)
\]

at least \( n/2 \) integers greater than \( n/2 \)
Beyond comparison-based sorting

Q: when are pairwise comparisons not necessary?

A: when the data you are sorting give you information about their absolute position in the ordering

for example...integers
Radix sort

for d in 0...D-1
    stable sort the items by digit d by partitioning into buckets 0-9
Radix sort

constraint: *your items are in the range* \([0, 10^D)\)

for \(d\) in \(0\ldots D-1\)
   stable sort the items by digit \(d\) by partitioning into buckets 0-9

Not limited to one digit at a time and base 10
- any base
- any number of digits at a time
Example radix sort base 10, one digit at a time

| 867 | 498 | 206 | 449 | 28 | 944 | 964 | 663 | 744 | 962 |

partition by least significant digit (1’s place)

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>962</td>
<td>663</td>
<td>944</td>
<td>964</td>
<td>744</td>
<td>206</td>
<td>867</td>
<td>498</td>
<td>28</td>
<td>449</td>
</tr>
</tbody>
</table>

partition by next digit (10’s place)

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>206</td>
<td>28</td>
<td>944</td>
<td>744</td>
<td>449</td>
<td>962</td>
<td>663</td>
<td>964</td>
<td>867</td>
<td>498</td>
</tr>
</tbody>
</table>

partition by next digit (100’s place)

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>028</td>
<td>206</td>
<td>449</td>
<td>498</td>
<td>663</td>
<td>744</td>
<td>867</td>
<td>944</td>
<td>962</td>
<td>964</td>
</tr>
</tbody>
</table>

crucially, on step d, the order of elements that share the digit d is maintained from the previous step (works if you partition from left to right)
Easier to see in an animation


Watch carefully for the three phases

1. **Count** number of items in each bucket
2. Do a running sum over the counts to **find bucket starting points**
3. **Partition** the items into the buckets, incrementing the bucket starting point

Then notice when the algorithm moves onto the next digit.
1. **Count** number of items in each bucket

| 35 | 88 | 101 | 117 | 122 | 158 | 193 | 203 | 252 | 286 | 326 | 377 | 643 | 716 | 669 | 743 | 643 | 88 | 117 | 878 | 326 | 158 | 193 | 122 | 549 | 377 | 544 | 974 | 626 | 252 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 |

| 0  | 1  | 2  | 2  | 0  | 1  | 2  | 2  | 2  | 0  | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  |
2. Do a running sum over the counts to find bucket ending (exclusive) points
3. **Partition** the items into the buckets, decrementing the bucket ending point
Peer Instruction

What does the counting array look like *after* counting and running sum steps are done for the 1’s digit?

<table>
<thead>
<tr>
<th></th>
<th>72</th>
<th>10</th>
<th>24</th>
<th>7</th>
<th>95</th>
<th>1</th>
<th>33</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

https://b.socrative.com/login/student/
room CS2230X ids 1000-2999
room CS2230Y ids 3000+
Running time of base 10 radix sort (1 digit at a time), D digits

• how long for counting and partitioning?
  • each requires 1 pass over the collection
  • $O(N)$

• how long for running sum?
  • one pass over the counts array
  • $O(10) = O(1)$

• how many iterations?
  • $D$: one iteration per digit

$O(D * N)$
Running time of base 10 radix sort (E digits at a time), D digits

• how long for counting and partitioning?
  • each requires 1 pass over the collection
    • ___

• how long for running sum?
  • one pass over the counts array
    • ___

• how many iterations?
  • One iteration per group of E digits
    • ___

_________________

https://b.socrative.com/login/student/
room CS2230X ids 1000-2999
room CS2230Y ids 3000+
Memory usage of radix sort (E digits at a time)

- don’t count memory usage of input array

- $O(10^E)$ for the array of counts (i.e., 1 for every possible combination of E digits)

- $O(N)$ for the array representing the buckets, since we manage to fit the items into it perfectly.

- total = $O(N+10^E)$
Radix vs comparison sorts

So what’s better?

<table>
<thead>
<tr>
<th>Radix sort</th>
<th>Efficient comparison sorts like mergesort and quicksort</th>
</tr>
</thead>
<tbody>
<tr>
<td>cannot be used if all you have is compareTo(a, b) method</td>
<td>can be used on any kind of items with a total order</td>
</tr>
<tr>
<td>if the data fall in a large range of values then either memory usage or</td>
<td>doesn’t need a finite range of values</td>
</tr>
<tr>
<td>time will increase (depending on your choice of base and E)</td>
<td></td>
</tr>
</tbody>
</table>

either one can be fastest, it depends on the situation e.g., Tencent the fastest sorting machine used both during different stages
Today’s learning objectives

• Calculate the number of comparisons needed to find a sort
• Execute sorting algorithms that are not comparison-based
• Calculate running time of a radix sort