Bucket Sort and Radix Sort

Steve Tanimoto
Winter 2016

Sorting: The Big Picture

Surprising amount of neat stuff to say about sorting:

- Simple algorithms: \(O(n^2)\)
- Fancier algorithms: \(O(n \log n)\)
- Comparison lower bound: \(\Omega(n \log n)\)
- Specialized algorithms: \(O(n)\)
- Handling huge data sets

<table>
<thead>
<tr>
<th>Insertion sort</th>
<th>Heap sort</th>
<th>Bucket sort</th>
<th>External sorting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection sort</td>
<td>Merge sort</td>
<td>Radix sort</td>
<td></td>
</tr>
<tr>
<td>Shell sort</td>
<td>Quick sort</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Radix sort

- Origins go back to the 1890 U.S. census
- Radix = “the base of a number system”
  - Examples will use 10 because we are used to that
  - In implementations use larger numbers
    - For example, for ASCII strings, might use 128
- Idea:
  - Bucket sort on one digit at a time
  - Number of buckets = radix
  - Starting with least significant digit
  - Keeping sort stable
  - Do one pass per digit
  - Invariant: After \(k\) passes (digits), the last \(k\) digits are sorted

Example

Radix = 10

First pass:
- Stable bucket sort by ones digit
- Order now: 721 3 143 537 478 38

Input:
- 478 537 9 721 3 38 143 67 478 9
- First pass: bucket sort by ones digit
- Order now: 3 9 721 143 537 478 38

Second pass:
- Stable bucket sort by tens digit
- Order now: 3 9 143 537 38 67 478

Example

Radix = 10

Third pass:
- Stable bucket sort by 100s digit
- Order now: 3 9 721 537 143 478 537 721

桶排序和基排序

Steve Tanimoto
Winter 2016

排序：大局

排序中有很多令人惊奇的内容。

- 简单算法：\(O(n^2)\)
- 花哨算法：\(O(n \log n)\)
- 比较下界：\(\Omega(n \log n)\)
- 专用算法：\(O(n)\)
- 处理大量数据

<table>
<thead>
<tr>
<th>插入排序</th>
<th>堆排序</th>
<th>桶排序</th>
<th>外部排序</th>
</tr>
</thead>
<tbody>
<tr>
<td>选择排序</td>
<td>合并排序</td>
<td></td>
<td></td>
</tr>
<tr>
<td>壳排序</td>
<td>快速排序</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

基排序

- 来源可以追溯到1890年的美国人口普查
- Radix = “数字系统的基础”
  - 示例将使用10，因为我们习惯如此
  - 在实现中使用较大的数字
    - 例如，对于ASCII字符串，可能使用128
- 概念：
  - 按位分桶排序
  - 位数 = Radix
  - 从最低位开始
  - 保持排序稳定
  - 做一次排序
  - Invariant：经过\(k\)次排序（位），最后\(k\)位是排序好的

例子

基数 = 10

第一轮：
- 稳定桶排序的一位
- 现在的顺序：721 3 143 537 478 38

输入：
- 478 537 9 721 3 38 143 67 478 9
- 第一次排序：按一位排序
- 现在的顺序：3 9 721 143 537 478

第二轮：
- 稳定桶排序的十位
- 现在的顺序：3 9 143 537 38 67 478

例子

基数 = 10

第三次排序：
- 稳定桶排序的100位
- 现在的顺序：3 9 721 537 143 478 537 721
Analysis

Input size: $n$
Number of buckets = Radix: $B$
Number of passes = "Digits": $P$
Work per pass is 1 bucket sort: $O(B+n)$
Total work is $O(P(B+n))$

Compared to comparison sorts, sometimes a win, but often not
- Example: Strings of English letters up to length 15
  - Run-time proportional to: $15^5(52 + n)$
  - This is less than $n \log n$ only if $n > 33,000$
  - Of course, cross-over point depends on constant factors of the implementations
    - And radix sort can have poor locality properties

Sorting: The Big Picture

Surprising amount of neat stuff to say about sorting:

Simple algorithms: $O(n^2)$
Fancier algorithms: $O(n \log n)$
Comparison lower bound: $\Omega(n \log n)$
Specialized algorithms: $O(n)$
Handling huge data sets

Insertion sort
Selection sort
Shell sort
Heap sort
Merge sort
Quick sort
Bucket sort
Radix sort
External sorting

Last Slide on Sorting

- Simple $O(n^2)$ sorts can be fastest for small $n$
- Selection sort, Insertion sort (latter linear for mostly-sorted)
- Good for "below a cut-off" to help divide-and-conquer sorts
- $O(n \log n)$ sorts
- Heap sort, in-place but not stable nor parallelizable
- Merge sort, not in place but stable and works as external sort
- Quick sort, in place but not stable and $O(n^2)$ in worst-case
- Often fastest, but depends on costs of comparisons/copies
- $\Omega(n \log n)$ is worst-case and average lower-bound for sorting by comparisons
- Non-comparison sorts
  - Bucket sort good for small number of possible key values
  - Radix sort uses fewer buckets and more phases
- Best way to sort? It depends!

Done with sorting! (phew..)

- Moving on…
- There are many many algorithm techniques in the world
  - We’ve learned a few
- What are a few other “classic” algorithm techniques you should at least have heard of?
  - And what are the main ideas behind how they work?