Sorting Lower Bound
Radix Sort

CSE 373
Data Structures
Lecture 15

Reading

• Reading
  › Sections 7.8-7.11

How fast can we sort?

• Heapsort, Mergesort, and Quicksort all run in O(N log N) best case running time
• Can we do any better?
• No, if the basic action is a comparison.

Sorting Model

• Recall our basic assumption: we can only compare two elements at a time
  › we can only reduce the possible solution space by half each time we make a comparison
• Suppose you are given N elements
  › Assume no duplicates
• How many possible orderings can you get?
  › Example: a, b, c (N = 3)

Permutations

• How many possible orderings can you get?
  › Example: a, b, c (N = 3)
  › (a b c), (a c b), (b a c), (b c a), (c a b), (c b a)
  › 6 orderings = 3·2·1 = 3! (i.e., "3 factorial")
  › All the possible permutations of a set of 3 elements
• For N elements
  › N choices for the first position, (N-1) choices for the second position, …, (2) choices, 1 choice
  › N(N-1)(N-2)…(2)(1)=N! possible orderings

Decision Tree

The leaves contain all the possible orderings of a, b, c
Decision Trees

- A Decision Tree is a Binary Tree such that:
  - Each node = a set of orderings
  - Each edge = 1 comparison
  - Each leaf = 1 unique ordering
  - How many leaves for N distinct elements?
    - \( N! \), i.e., a leaf for each possible ordering
  - Only 1 leaf has the ordering that is the desired correctly sorted arrangement

Decision Trees and Sorting

- Every sorting algorithm corresponds to a decision tree
  - Finds correct leaf by choosing edges to follow
  - i.e., by making comparisons
  - Each decision reduces the possible solution space by one half
  - Run time is \( \geq \) maximum no. of comparisons
    - maximum number of comparisons is the length of the longest path in the decision tree, i.e. the height of the tree

Decision Tree Example

- Possible orders
- Actual order

How many leaves on a tree?

- Suppose you have a binary tree of height \( d \).
  - How many leaves can the tree have?
    - \( d = 1 \) \( \rightarrow \) at most 2 leaves,
    - \( d = 2 \) \( \rightarrow \) at most 4 leaves, etc.

Lower bound on Height

- A binary tree of height \( d \) has at most \( 2^d \) leaves
  - depth \( d = 1 \) \( \rightarrow \) 2 leaves, \( d = 2 \) \( \rightarrow \) 4 leaves, etc.
  - Can prove by induction
- Number of leaves, \( L \leq 2^d \)
- Height \( d \geq \log_2 L \)
- The decision tree has \( N! \) leaves
- So the decision tree has height \( d \geq \log_2(N!) \)

\( \log(N!) = \Omega(N \log N) \)

\[
\log(N!) = \log(N \cdot (N-1) \cdot \ldots \cdot 1) \\
\geq \log N + \log(N-1) + \log(N-2) + \ldots + \log 2 + \log 1 \\
\geq \log N + \log(N-1) + \log(N-2) + \ldots + \log \frac{N}{2} \\
\geq \frac{N}{2} \log \frac{N}{2} \\
\geq \frac{N}{2} \left( \log N - \log 2 \right) = \frac{N}{2} \log N - \frac{N}{2} \\
= \Omega(N \log N)
\]
\( \Omega(N \log N) \)

- Run time of any comparison-based sorting algorithm is \( \Omega(N \log N) \)
- Can we do better if we don’t use comparisons?

Radix Sort: Sorting integers

- Historically goes back to the 1890 census.
- Radix sort = multi-pass bucket sort of integers in the range 0 to \( B^r - 1 \)
- Bucket sort from least significant to most significant “digit” (base B)
- Requires \( P(B + N) \) operations where \( P \) is the number of passes (the number of base B digits in the largest possible input number).
- If \( P \) and \( B \) are constants then \( O(N) \) time to sort!

Radix Sort Example

<table>
<thead>
<tr>
<th>Input data</th>
<th>Bucket sort by 1’s digit</th>
<th>After 1st pass</th>
</tr>
</thead>
<tbody>
<tr>
<td>478</td>
<td>721</td>
<td>3</td>
</tr>
<tr>
<td>532</td>
<td>721</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>123</td>
<td>67</td>
</tr>
<tr>
<td>721</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>38</td>
<td>3</td>
<td>38</td>
</tr>
<tr>
<td>123</td>
<td>3</td>
<td>38</td>
</tr>
<tr>
<td>67</td>
<td>9</td>
<td>38</td>
</tr>
</tbody>
</table>

This example uses base 10 and base 10 digits for simplicity of demonstration. Larger bucket counts should be used in an actual implementation.

Radix Sort Example

<table>
<thead>
<tr>
<th>After 1st pass</th>
<th>After 2nd pass</th>
</tr>
</thead>
<tbody>
<tr>
<td>721</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>721</td>
</tr>
<tr>
<td>123</td>
<td>67</td>
</tr>
<tr>
<td>537</td>
<td>123</td>
</tr>
<tr>
<td>38</td>
<td>478</td>
</tr>
<tr>
<td>67</td>
<td>478</td>
</tr>
</tbody>
</table>

Radix Sort Example

<table>
<thead>
<tr>
<th>After 2nd pass</th>
<th>After 3rd pass</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>721</td>
<td>38</td>
</tr>
<tr>
<td>123</td>
<td>123</td>
</tr>
<tr>
<td>537</td>
<td>537</td>
</tr>
<tr>
<td>38</td>
<td>478</td>
</tr>
<tr>
<td>67</td>
<td>478</td>
</tr>
</tbody>
</table>

Invariant: after k passes the low order k digits are sorted.

Implementation Options

- List
  - List of data, bucket array of lists.
  - Concatenate lists for each pass.
- Array / List
  - Array of data, bucket array of lists.
- Array / Array
  - Array of data, array for all buckets.
  - Requires counting.
**Array / Array**

<table>
<thead>
<tr>
<th>Data Array</th>
<th>Count Array</th>
<th>Address Array</th>
<th>Target Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>2</td>
<td>0</td>
<td>0 720</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>2</td>
<td>2 123</td>
</tr>
<tr>
<td>14</td>
<td>3</td>
<td>1</td>
<td>3 667</td>
</tr>
<tr>
<td>17</td>
<td>4</td>
<td>0</td>
<td>4 878</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>0</td>
<td>5 687</td>
</tr>
<tr>
<td>24</td>
<td>3</td>
<td>1</td>
<td>3 667</td>
</tr>
<tr>
<td>30</td>
<td>2</td>
<td>2</td>
<td>2 123</td>
</tr>
<tr>
<td>35</td>
<td>1</td>
<td>3</td>
<td>3 667</td>
</tr>
<tr>
<td>37</td>
<td>1</td>
<td>3</td>
<td>3 667</td>
</tr>
<tr>
<td>40</td>
<td>1</td>
<td>3</td>
<td>3 667</td>
</tr>
<tr>
<td>47</td>
<td>1</td>
<td>3</td>
<td>3 667</td>
</tr>
<tr>
<td>50</td>
<td>1</td>
<td>3</td>
<td>3 667</td>
</tr>
<tr>
<td>57</td>
<td>1</td>
<td>3</td>
<td>3 667</td>
</tr>
</tbody>
</table>

add(0) = 0
add() = add([-1] + count[-1]), i > 0

**Choosing Parameters for Radix Sort**

- N number of integers – given
- m bit numbers - given
- B number of buckets
  - B = 2^k calculations can be done by shifting.
  - N/B not too small, otherwise too many empty buckets.
  - P = m/B should be small.
- Example – 1 million 64 bit numbers. Choose B = 2^16 = 65,536. 1 Million / B = 15 numbers per bucket. P = 64/16 = 4 passes.

**Properties of Radix Sort**

- Not in-place
  - needs lots of auxiliary storage.
- Stable
  - equal keys always end up in same bucket in the same order.
- Fast
  - B = 2^k buckets on m bit numbers
  - $O(n \cdot (n + 2^k))$ time

**Internal versus External Sorting**

- So far assumed that accessing A[i] is fast – Array A is stored in internal memory (RAM)
  - Algorithms so far are good for internal sorting
- What if A is so large that it doesn’t fit in internal memory?
  - Data on disk or tape
  - Delay in accessing A[i] – e.g. need to spin disk and move head

- Need sorting algorithms that minimize disk/tape access time
  - External sorting – Basic Idea:
    - Load chunk of data into RAM, sort, store this “run” on disk/tape
    - Use the Merge routine from Mergesort to merge runs
    - Repeat until you have only one run (one sorted chunk)
  - Text gives some examples
Summary of Sorting

- Sorting choices:
  - $O(N^2)$ – Bubblesort, Insertion Sort
  - $O(N \log N)$ average case running time:
    - Heapsort: In-place, not stable.
    - Mergesort: $O(N)$ extra space, stable.