Topic 6: Sorting Lower Bound and Radix Sort
(CLRS 8.0–8.3)

CPS 230, Fall 2001

1 Comparison model sorting lower bound

- We have seen two sorting $\Theta(n \log n)$ algorithms: Merge sort and quicksort (using median selection).
- These algorithms only use comparisons to gain information about the input.
- We will prove that such algorithms have to do $\Omega(n \log n)$ comparisons.
- To prove bound, we need *formal model*

<table>
<thead>
<tr>
<th>Decision tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Binary tree where each internal node is labeled $a_i \leq a_j$ ($a_i$ is the $i$th input element).</td>
</tr>
<tr>
<td>- Execution corresponds to root-leaf path.</td>
</tr>
<tr>
<td>* at each internal node, the corresponding comparison $a_i \leq a_j$ is performed.</td>
</tr>
<tr>
<td>* If the comparison is true, the left branch is taken; otherwise the right branch is taken.</td>
</tr>
<tr>
<td>- Leaf contains result of computation.</td>
</tr>
</tbody>
</table>

- Example: Decision tree for sorting 3 elements.

- a leaf contains the permutation giving the final sorted order.
- For example, the leaf (1, 3, 2) means that $a_1 \leq a_3 \leq a_2$.

- Note: Decision tree model corresponds to algorithms where
  - Only comparisons can be used to gain knowledge about input
  - Data movement, control, etc, are ignored

- The worst-case number of comparisons performed corresponds to the height of the decision tree (i.e., the longest root-to-leaf path).
• Therefore, a lower bound on height $\implies$ lower bound on sorting

**Theorem:** Any decision tree sorting $n$ elements has height $\Omega(n \log n)$.

Proof:
- Assume elements are the (distinct) numbers 1 through $n$
- There must be $n!$ leaves (one for each of the $n!$ permutations of $n$ elements)
- Tree of height $h$ has at most $2^h$ leaves

\[
2^h \geq n! \implies h \geq \log n!
\]

\[
= \log(n(n-1)(n-2)\cdots(2))
\]

\[
= \log n + \log(n-1) + \log(n-2) + \cdots + \log 2
\]

\[
= \sum_{i=2}^{n} \log i
\]

\[
= \sum_{i=2}^{n/2-1} \log i + \sum_{i=n/2}^{n} \log i
\]

\[
\geq 0 + \sum_{i=n/2}^{n} \log \frac{n}{2}
\]

\[
= \frac{n}{2} \cdot \log \frac{n}{2}
\]

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

2 Beating sorting lower bound (bucket sort)

• While proving the $\Omega(n \log n)$ comparison lower bound we assumed that the input were integers 1 through $n$

• We can easily sort integers 1 through $n$ in $O(n)$ time.

  – just move element $i$ to position $i$ in output array

  \[
  \begin{array}{cccccccc}
  4 & 7 & 6 & 2 & 5 & 3 & 10 & 9 & 1 & 8 \\
  \end{array}
  \]

\[
\downarrow
\]

\[
\begin{array}{cccccccc}
  1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
  \end{array}
  \]

• What about the more general problem of sorting $n$ elements in range 1..$k$?

  – Move element $i$ to linked list of element $i$
  – Produce sorted output
• Algorithm uses $O(n + k)$ time and space

• Note:
  - We did not use comparison at all!
  - We beat the $\Omega(n \log n)$ bound by using values of elements to index into array—*Indirect addressing*

• Note:
  - Algorithm is *stable* (i.e., the order of equal elements is maintained)
  - Algorithm is not *in-place* (since asymptotically more than $\sim n$ space is used) or even *linear-space* (since more than $O(n)$ space is used). All other sorting algorithms we have seen have been linear-space, and quicksort was in-place.

• Note:
  - Book calls a simplified version of this algorithm *counting sort*. That algorithm just counts how many keys have each possible value. It therefore can’t handle sorting records that have both keys and auxiliary information.
  - The book uses the term *bucket sort* for the algorithm where each bucket corresponds to a contiguous range of values, not a single value. Each bucket is then sorted using an algorithm like insertion sort.
  - For simplicity, we call all these algorithms, including the one we cover above, *bucket sort*.

3 Radix sort

• Problem with bucket sort is that the amount $k$ of auxiliary space can be very large
  - Example: 32 bit integers $\implies k = 2^{32} \approx 10^9 \implies$ space used is $10^9 \cdot 4 \text{ bytes} \approx 4 \text{Gbytes}$!

• Large $k$ results in a running time not proportional to $n$ (as well as other problems like disk swapping)

3.1 MSD radix sort

• MSD radix sort regards numbers as being made up of digits
  - Bucket sort by most significant digit (MSD)
  - Recursively sort buckets with more than one element (according to next digit)

• Correctness is straightforward (by induction)

• Example: Sorting numbers $< 1000$ ($k = 1000$) using 10 buckets
• Problem with MSD radix sort
  – We need to keep track of a lot of recursion (buckets)
  – Many buckets $\Rightarrow$ space use

• Advantages of MSD radix sort
  – We only need to look at distinguishing prefix (what we need to look at)

3.2 LSD radix sort

• LSD radix sort:
  – Sort by least significant digit (LSD)
  – Sort by second least significant digit (using a stable sorting algorithm)
    :
  – Sort by most significant digit (using a stable sorting algorithm)

• Correctness again by induction

• Example:

<table>
<thead>
<tr>
<th>329</th>
<th>0 : 720</th>
<th>720</th>
<th>0 : 720</th>
<th>720</th>
<th>0 : 329</th>
</tr>
</thead>
<tbody>
<tr>
<td>457</td>
<td>1 :</td>
<td>1</td>
<td>1 :</td>
<td>1</td>
<td>1 :</td>
</tr>
<tr>
<td>657</td>
<td>2 :</td>
<td>2</td>
<td>2 : 720,329</td>
<td>720</td>
<td>2 :</td>
</tr>
</tbody>
</table>

• Problems with LSD radix sort:
  – We look at all the numbers in all phases
  – Not generally linear-space (since $n$ might be much smaller than the number of buckets)
3.3 Linear-space radix sort

- To get a linear-space radix sort algorithm (but not in-place, which means $o(n)$ extra space), we simply choose the number of buckets to be $O(n)$.
  
  - In example, we had $n = 7$ and the number of buckets was 10. We divided the key range into 10 intervals.

- Let’s choose the number of buckets to be $n$ so that we use linear space. If numbers are $\leq R$, the number of phases $i$ is such that $n^i \geq R \implies i = \lceil \log R \over \log n \rceil$
  
  - In example, we had $R = 839$, $10^3 > 839$ \implies 3 phases
  
  $\implies O(n)$ space and $O(n \cdot \log R \over \log n)$ time

- Note: When is linear-space radix sort using time $n \lceil \log R \over \log n \rceil$ better than a sort using time equal to $1 \cdot n \log n$ (for 32 bit integers)?
  
  - $n \cdot \lceil \log R \over \log n \rceil < n \log n \implies n \geq 64$

- Note: Recent algorithm by Andersson and Nilsson (1997) combines advantages of MSD and LSD radix sort
  
  - Linear-space
  
  - Only looks at distinguishing prefixes