1 Comparison model sorting lower bound

- We have seen two \( \Theta(n \log n) \) sorting algorithms: Merge-sort and quick-sort (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 of decision tree:
  
  - Binary tree where each internal node is labeled \( a_i \leq a_j \) (\( a_i \) is the \( i \)'th input element).
  - Execution corresponds to root-leaf path.
  - At each internal node comparisons \( a_i \leq a_j \) is performed and branching made.
  - Leaf contains result of computation.

- Example: Decision tree for sorting 3 elements.

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

- Worst case number of comparisons performed corresponds to maximal height of tree \( \Rightarrow \) lower bound on height \( \Rightarrow \) 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

\[
2^h \geq n! \Rightarrow h \geq \log(n!)
\]
\[
= \log(n(n-1)(n-2)\cdots(2))
\]
\[
= \log n + \log(n-1) + \log(n-2) + \cdots + \log 2
\]
\[
= \sum_{i=2}^{n} \log i
\]
\[
= \sum_{i=2}^{n/2-1} \log i + \sum_{i=n/2}^{n} \log i
\]
\[
\geq 0 + \sum_{i=n/2}^{n} \log \frac{n}{2}
\]
\[
= \frac{n}{2} \cdot \log \frac{n}{2}
\]
\[
= \Omega(n \log n)
\]

2 \hspace{1em} **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

```
4 7 6 2 5 3 10 9 1 8
```

```
1 2 3 4 5 6 7 8 9 10
```

- What about the more general problem of sorting $n$ elements in range $1\ldots k$?
  - Move element $i$ to linked list of element $i$
  - Produce sorted output

```
\[\text{n elements}\]
\[\text{i}\]
\[\text{k cells}\]
\[\text{n sorted elements}\]
```
• 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* (Order of equal elements maintained)
  – Algorithm is not *in-place* (more than $O(n)$ space use)—All other sorting algorithms we have seen have been in-place

• Note:
  – Book calls the algorithm (or simplified version of it) *counting sort* and use *bucket sort* for something else
  – I call it *bucket sort* (we put elements in buckets)

3 Radix Sort

• Problem with bucket sort is that $k$ can be very large
  – Example: 32 bit integers $\Rightarrow k = 2^{32} \approx 10^9$ $\Rightarrow$ space used is $10^9 \cdot 4$ bytes $\approx 4$Gbytes!

• Large $k$ result in running time not proportional to $n$ (and 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 (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 ⇒ 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>
</tr>
</thead>
<tbody>
<tr>
<td>457</td>
<td>1: 2</td>
</tr>
<tr>
<td>657</td>
<td>2: 355</td>
</tr>
<tr>
<td>839</td>
<td>3: 436</td>
</tr>
<tr>
<td>436</td>
<td>4: 839</td>
</tr>
<tr>
<td>720</td>
<td>5: 355</td>
</tr>
<tr>
<td>355</td>
<td>6: 720</td>
</tr>
<tr>
<td>8</td>
<td>7: 329</td>
</tr>
<tr>
<td>9</td>
<td>8: 457</td>
</tr>
<tr>
<td>839</td>
<td>9: 657</td>
</tr>
</tbody>
</table>
```

Problems with LSD Radix-sort:
- We look at all the numbers in all phases
- Not generally in-place \((n < 10)\)

3.3 In-place Radix-sort

To get in-place algorithm we simply choose number of buckets equal \(n\) in radix sort
- In example, we had \(n = 7\) and 10 buckets

When doing so we divide the numbers in ranges of \(n\)
- In example, we divided in ranges of 10

If numbers are \(\leq R\) the number of phases \(i\) is \(n^i = R \Rightarrow i = \frac{\log R}{\log n}\)
- In example, we had \(R = 839, 10^3 > 839 \Rightarrow 3\) phases

\(O(n)\) space and \(O(n \cdot \frac{\log R}{\log n})\) time
• Note: When is in-place Radix-sort better than $1 \cdot n \log n$ sort (for 32 bit integers)?

\[
\begin{align*}
- \quad n \cdot \frac{32}{\log n} &< n \log n \Rightarrow \log^2 n > 32 \Rightarrow n > 2^{\sqrt{32}} \\
- \quad 2^{\sqrt{32}} < 2^6 = 64
\end{align*}
\]

• Note: Recent algorithm by Anderson et al. (1997) combines advantages of MSD and LSD radix sort

- In-place
- Only look at distinguishing prefix