Introduction to Algorithms

Lecture 10

Prof. Constantinos Daskalakis

CLRS 8.1-8.4
Menu

• Show that $\Theta(n \lg n)$ is the best possible running time for a sorting algorithm.
• Design an algorithm that sorts in $\Theta(n)$ time.
• Hint: maybe the models are different?
Comparison sort

All the sorting algorithms we have seen so far are *comparison sorts*: only use comparisons to determine the relative order of elements.

- *E.g.*, merge sort, heapsort.

The best running time that we’ve seen for comparison sorting is $O(n \lg n)$.

*Is* $O(n \lg n)$ *the best we can do?*

*Decision trees* can help us answer this question.
A recipe for sorting $n$ numbers $\langle a_1, a_2, \ldots, a_n \rangle$

- Nodes are suggested comparisons: $i:j$ means compare $a_i$ to $a_j$, for $i, j \in \{1, 2, \ldots, n\}$.

- Branching direction depends on outcome of comparisons.

- Leaves are labeled with permutations corresponding to the outcome of the sorting.
Each internal node is labeled \( i:j \) for \( i, j \in \{1, 2, \ldots, n\} \).
- The left subtree shows subsequent comparisons if \( a_i \leq a_j \).
- The right subtree shows subsequent comparisons if \( a_i \geq a_j \).
Decision-tree example

Sort $\langle a_1, a_2, a_3 \rangle$
$= \langle 9, 4, 6 \rangle$:

Each internal node is labeled $i:j$ for $i, j \in \{1, 2, \ldots, n\}$.
- The left subtree shows subsequent comparisons if $a_i \leq a_j$.
- The right subtree shows subsequent comparisons if $a_i \geq a_j$. 
Decision-tree example

Sort $\langle a_1, a_2, a_3 \rangle$
$= \langle 9, 4, 6 \rangle$:

Each internal node is labeled $i:j$ for $i, j \in \{1, 2, \ldots, n\}$.
- The left subtree shows subsequent comparisons if $a_i \leq a_j$.
- The right subtree shows subsequent comparisons if $a_i \geq a_j$. 
Decision-tree example

Sort $\langle a_1, a_2, a_3 \rangle$
= $\langle 9, 4, 6 \rangle$:

Each leaf contains a permutation $\langle \pi(1), \pi(2), \ldots, \pi(n) \rangle$ to indicate that the ordering $a_{\pi(1)} \leq a_{\pi(2)} \leq \cdots \leq a_{\pi(n)}$ has been established.
Decision-tree model

A decision tree can model the execution of any comparison sort:

- One tree for each input size $n$.
- A path from the root to the leaves of the tree represents a trace of comparisons that the algorithm may perform.
- The running time of the algorithm $= \text{the length of the path taken.}$
- Worst-case running time $= \text{height of tree.}$
**Lower bound for decision-tree sorting**

**Theorem.** Any decision tree that can sort $n$ elements must have height $\Omega(n \log n)$.

**Proof.** *(Hint: how many leaves are there?)*

- The tree must contain $\geq n!$ leaves, since there are $n!$ possible permutations.
- A height-$h$ binary tree has $\leq 2^h$ leaves.
- Thus $2^h \geq n!$.

$$h \geq \log(n!) \geq \log((n/e)^n) = n \log n - n \log e = \Omega(n \log n).$$
Sorting in linear time

**Counting sort:** No comparisons between elements.

- **Input:** $A[1 \ldots n]$, where $A[j] \in \{1, 2, \ldots, k\}$.
- **Output:** $B[1 \ldots n]$, a sorted permutation of $A$.
- **Auxiliary storage:** $C[1 \ldots k]$. 
Counting sort

for $i \leftarrow 1$ to $k$
    do $C[i] \leftarrow 0$

for $j \leftarrow 1$ to $n$
    do $C[A[j]] \leftarrow C[A[j]] + 1$

for $i \leftarrow 2$ to $k$
    do $C[i] \leftarrow C[i] + C[i-1]$ \hfill store in $C$ the frequencies of the different keys in $A$

    i.e. $C[i] = |\{\text{key} = i\}|$

for $j \leftarrow n$ downto $1$
    do $B[C[A[j]]] \leftarrow A[j]$ \hfill now $C$ contains the cumulative

    $C[A[j]] \leftarrow C[A[j]] - 1$ \hfill frequencies of different keys in

    using cumulative \hfill A, i.e. $C[i] = |\{\text{key} \leq i\}|$

    frequencies build \hfill sorted permutation

    sorted permutation
Counting-sort example

A: 4 1 3 4 3

B: 

C:  

one index for each possible key stored in A
Loop 1: initialization

\[ A: \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 \\ 4 & 1 & 3 & 4 & 3 \end{array} \]

\[ C: \begin{array}{cccc} 1 & 2 & 3 & 4 \\ 0 & 0 & 0 & 0 \end{array} \]

\[
\text{for } i \leftarrow 1 \text{ to } k \\
\text{do } C[i] \leftarrow 0
\]
Loop 2: count frequencies

\[
\begin{align*}
\text{for } j & \leftarrow 1 \text{ to } n \\
\text{do } & \ C[A[j]] \leftarrow C[A[j]] + 1 \quad \triangleright C[i] = |\{\text{key} = i\}| 
\end{align*}
\]
Loop 2: count frequencies

\[
\begin{array}{c|ccccc}
1 & 2 & 3 & 4 & 5 \\
\hline
A: & 4 & 1 & 3 & 4 & 3 \\
\end{array}
\quad C: \begin{array}{c|ccccc}
1 & 2 & 3 & 4 & 5 \\
\hline
1 & 0 & 0 & 41 \\
\end{array}
\]

\[
\text{for } j \leftarrow 1 \text{ to } n \\
\text{do } C[A[j]] \leftarrow C[A[j]] + 1 \quad \triangleright C[i] = |\{\text{key} = i\}|
\]
Loop 2: count frequencies

\[
\begin{align*}
A: & \quad 4 \quad 1 \quad 3 \quad 4 \quad 3 \\
B: & \quad \quad \quad \quad \quad \\
C: & \quad 1 \quad 0 \quad 1 \quad 1 \quad 1 \\
\end{align*}
\]

for \( j \leftarrow 1 \) to \( n \)

\[
\text{do } C[A[j]] \leftarrow C[A[j]] + 1 \quad \triangleright \quad C[i] = |\{\text{key} = i\}|
\]
Loop 2: count frequencies

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

\[
\text{for } j \leftarrow 1 \text{ to } n \\
\text{do } C[A[j]] \leftarrow C[A[j]] + 1 \quad \triangleright \quad C[i] = |\{\text{key} = i\}|
Loop 2: count frequencies

\[
\begin{align*}
A: & \quad 4 \quad 1 \quad 3 \quad 4 \quad 3 \\
B: & \quad \quad \quad \quad \quad \quad \quad \quad \\
C: & \quad 1 \quad 0 \quad 2 \quad 2
\end{align*}
\]

for \( j \leftarrow 1 \) to \( n \)
do \( C[A[j]] \leftarrow C[A[j]] + 1 \) \( \triangleright C[i] = |\{\text{key} = i\}| \)
Loop 2: count frequencies

\[
\begin{array}{llllll}
A: & 1 & 2 & 3 & 4 & 5 \\
& 4 & 1 & 3 & 4 & 3 \\
\end{array}
\quad
\begin{array}{llll}
C: & 1 & 0 & 2 & 2 \\
& & & & \\
\end{array}
\]

\[
\text{for } j \leftarrow 1 \text{ to } n \\
\text{do } C[A[j]] \leftarrow C[A[j]] + 1 \quad \triangleright C[i] = |\{\text{key} = i\}|
\]
[A parenthesis: a quick finish

Walk through frequency array and place the appropriate number of each key in output array...
A parenthesis: a quick finish

A: 4 1 3 4 3
B: 1
C: 

1 2 3 4
A parenthesis: a quick finish

<table>
<thead>
<tr>
<th>A:</th>
<th>4</th>
<th>1</th>
<th>3</th>
<th>4</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>B:</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C:</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

1 2 3 4 5
A parenthesis: a quick finish

$A$: 4 1 3 4 3

$B$: 1 3 3

$C$: 1 0 2 2
A parenthesis: a quick finish

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A:</strong></td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td><strong>B:</strong></td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td><strong>C:</strong></td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

B is sorted!
but it is not “stably sorted”…]
Loop 2: count frequencies

for $j \leftarrow 1$ to $n$
  do $C[A[j]] \leftarrow C[A[j]] + 1$  \> $C[i] = |\{\text{key} = i\}|$
Loop 3: cumulative frequencies

<table>
<thead>
<tr>
<th></th>
<th>A:</th>
<th>B:</th>
<th>C:</th>
<th>C':</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td></td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td></td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td></td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

for $i \leftarrow 2$ to $k$

\[
\text{do } C[i] \leftarrow C[i] + C[i-1] \quad \triangleright \quad C[i] = |\{\text{key} \leq i\}|
\]
Loop 3: cumulative frequencies

for $i \leftarrow 2$ to $k$

do $C[i] \leftarrow C[i] + C[i-1]$

\[ C[i] = |\{\text{key} \leq i\}| \]
Loop 3: cumulative frequencies

for $i \leftarrow 2$ to $k$

    do $C[i] \leftarrow C[i] + C[i-1]$  \quad \triangleright C[i] = |\{\text{key } \leq i\}|
Loop 4: permute elements of A

for \( j \leftarrow n \) downto 1
    do \( B[C[A[j]]] \leftarrow A[j] \)
        \( C[A[j]] \leftarrow C[A[j]] - 1 \)
Loop 4: permute elements of A

for \( j \leftarrow n \) downto 1
  do \( B[C[A[j]]] \leftarrow A[j] \)
      \( C[A[j]] \leftarrow C[A[j]] - 1 \)

Loop 4: permute elements of \( A \)

\[
\begin{align*}
A: & \quad 4 & 1 & 3 & 4 & 3 \\
B: & & 3 \\
C: & 1 & 1 & 3 & 5 \\
\end{align*}
\]

for \( j \leftarrow n \) downto 1 

\[
\begin{align*}
do & \quad B[C[A[j]]] \leftarrow A[j] \\
& \quad C[A[j]] \leftarrow C[A[j]] - 1
\end{align*}
\]

Used-up one 3; update counter.
Loop 4: permute elements of A

for $j \leftarrow n$ downto 1
  do $B[C[A[j]]] \leftarrow A[j]$\
      $C[A[j]] \leftarrow C[A[j]] - 1$
Loop 4: permute elements of $A$

for $j \leftarrow n$ downto 1
    do $B[C[A[j]]] \leftarrow A[j]$
        $C[A[j]] \leftarrow C[A[j]] - 1$
Loop 4: permute elements of A

There are exactly 5 elements \( \leq A[4] \), so where should I place \( A[4] \)?

\[
\text{for } j \leftarrow n \ \text{downto} \ 1 \\
\text{do } B[C[A[j]]] \leftarrow A[j] \\
C[A[j]] \leftarrow C[A[j]] - 1
\]
Loop 4: permute elements of A

for $j \leftarrow n$ downto 1
  do $B[C[A[j]]] \leftarrow A[j]$
  $C[A[j]] \leftarrow C[A[j]] - 1$
Loop 4: permute elements of A

\[
\text{for } j \leftarrow n \text{ downto } 1 \\
\text{do } B[C[A[j]]] \leftarrow A[j] \\
    C[A[j]] \leftarrow C[A[j]] - 1
\]
Loop 4: permute elements of A

for $j \leftarrow n$ downto 1
    do $B[C[A[j]]] \leftarrow A[j]$
    $C[A[j]] \leftarrow C[A[j]] - 1$
Loop 4: permute elements of A

for $j \leftarrow n$ downto 1
  do $B[C[A[j]]] \leftarrow A[j]
       C[A[j]] \leftarrow C[A[j]] - 1
for $j \leftarrow n$ downto 1  
  do $B[C[A[j]]] \leftarrow A[j]$  
      $C[A[j]] \leftarrow C[A[j]] - 1$
Loop 4: permute elements of $A$

for $j \leftarrow n$ downto 1
  do $B[C[A[j]]] \leftarrow A[j]$
      $C[A[j]] \leftarrow C[A[j]] - 1$
for $j \leftarrow n \text{ downto } 1$

\begin{align*}
&\text{do } B[C[A[j]]] \leftarrow A[j] \\
&C[A[j]] \leftarrow C[A[j]] - 1
\end{align*}
Loop 4: permute elements of A

for $j \leftarrow n \text{ downto } 1$
  do $B[C[A[j]]] \leftarrow A[j]$
      $C[A[j]] \leftarrow C[A[j]] - 1$
Loop 4: permute elements of $A$

for $j \leftarrow n$ downto 1

do $B[C[A[j]]] \leftarrow A[j]$

$C[A[j]] \leftarrow C[A[j]] - 1$
Loop 4: permute elements of A

for $j \leftarrow n \text{ downto } 1$

do $B[C[A[j]]] \leftarrow A[j]$

$C[A[j]] \leftarrow C[A[j]] - 1$
Analysis

\[ \Theta(k) \]
- for \( i \leftarrow 1 \) to \( k \)
  - do \( C[i] \leftarrow 0 \)

\[ \Theta(n) \]
- for \( j \leftarrow 1 \) to \( n \)
  - do \( C[A[j]] \leftarrow C[A[j]] + 1 \)

\[ \Theta(k) \]
- for \( i \leftarrow 2 \) to \( k \)
  - do \( C[i] \leftarrow C[i] + C[i-1] \)

\[ \Theta(n) \]
- for \( j \leftarrow n \) downto \( 1 \)
  - do \( B[C[A[j]]] \leftarrow A[j] \)
  - \( C[A[j]] \leftarrow C[A[j]] - 1 \)

\[ \Theta(n + k) \]
Running time

If $k = O(n)$, then counting sort takes $\Theta(n)$ time.
- But, sorting takes $\Omega(n \lg n)$ time!
- Where’s the fallacy?

**Answer:**
- *Comparison sorting* takes $\Omega(n \lg n)$ time.
- Counting sort is not a *comparison sort*.
- In fact, not a single comparison between elements occurs!
Stable sorting

Counting sort is a *stable* sort: it preserves the input order among equal elements.

**A:**

\[
\begin{array}{cccccc}
4 & 1 & 3 & 4 & 3 \\
\end{array}
\]

**B:**

\[
\begin{array}{cccccc}
1 & 3 & 3 & 4 & 4 \\
\end{array}
\]
Radix sort

- **Origin**: Herman Hollerith’s card-sorting machine for the 1890 U.S. Census. (See Appendix.)
- Digit-by-digit sort.
- Hollerith’s original (bad) idea: sort on most-significant digit first.
- Good idea: Sort on *least-significant digit first* with auxiliary *stable* sort.
Operation of radix sort

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>329</td>
<td>720</td>
<td>720</td>
<td>329</td>
</tr>
<tr>
<td>457</td>
<td>355</td>
<td>329</td>
<td>355</td>
</tr>
<tr>
<td>657</td>
<td>436</td>
<td>436</td>
<td>436</td>
</tr>
<tr>
<td>839</td>
<td>457</td>
<td>839</td>
<td>457</td>
</tr>
<tr>
<td>436</td>
<td>657</td>
<td>355</td>
<td>657</td>
</tr>
<tr>
<td>720</td>
<td>329</td>
<td>457</td>
<td>720</td>
</tr>
<tr>
<td>355</td>
<td>839</td>
<td>657</td>
<td>839</td>
</tr>
</tbody>
</table>
Correctness of radix sort

*Induction on digit position*

- Assume that the numbers are sorted by their low-order \( t - 1 \) digits.
- Sort on digit \( t \)
Correctness of radix sort

Induction on digit position

• Assume that the numbers are sorted by their low-order \( t - 1 \) digits.

• Sort on digit \( t \)
  ▪ Two numbers that differ in digit \( t \) are correctly sorted.
Correctness of radix sort

*Induction on digit position*

- Assume that the numbers are sorted by their low-order \( t-1 \) digits.

- Sort on digit \( t \)
  - Two numbers that differ in digit \( t \) are correctly sorted.
  - Two numbers equal in digit \( t \) are put in the same order as the input \( \Rightarrow \) correct order.
Runtime Analysis of radix sort

- Assume counting sort is the auxiliary stable sort.
- Sort $n$ computer words of $b$ bits each.
- Each word can be viewed as having $b/r$ base-$2^r$ digits.
- If each $b$-bit word is broken into $r$-bit pieces, each pass of counting sort takes $\Theta(n + 2^r)$ time.
- Setting $r = \log n$ gives $\Theta(n)$ time per pass, or $\Theta(n \frac{b}{\log n})$ total.

Example: 32-bit word

```
8 8 8 8
```