Linear Time Sorting
Lower Bound for Sorting
Lower Bound

Theorem

Comparison based sorting of $n$ elements requires $\Omega(n \log n)$ time.
Lower Bound

Recall decision trees.

Decision tree for sorting
- Leaf represents permutation of $A$.
  - $n!$ leafs
  - Thus, height is at least $\log n!$
What is $\log n$? 

$$
\log n! = \log (1 \cdot 2 \cdot \ldots \cdot n)
= \log 1 + \log 2 + \ldots + \log(n/2) + \ldots + \log n \quad (\leq n \log n)
\geq \frac{n}{2} \log(n/2)
= \frac{n}{2} \log n - \frac{n}{2}
\in \Omega(n \log n)
$$
Comparison based sorting requires $\Omega(n \log n)$ time.

What if we do something different?

Integer Sorting

Given $n$ integers in $\{0, \ldots, k - 1\}$

- Each integer fits in a word.
- All operations for words are allowed, e.g., $+$, $-$, $\ldots$
- Allows linear time sorting if $k$ is limited.
Counting Sort
Counting Sort

Idea

- Count for each $v \in \{0, \ldots, k - 1\}$ how often it is in $A$.
- Use this to compute index of $A[i]$ in sorted array.

\[
A = \begin{bmatrix}
5 & 2_a & 4 & 6_a & 3 & 1 & 6_b & 2_b \\
\end{bmatrix}
\]

\[
C = \begin{bmatrix}
0 & 1 & 2 & 1 & 1 & 1 & 2 \\
0 & 1 & 2 & 3 & 4 & 5 & 6 \\
\end{bmatrix}
\]

How do we compute the index?
Counting Sort

Last position of some value $v$ in $B$:

$\text{Ind}_B(v) = C[0] + C[1] + \ldots + C[v] - 1$

($-1$ because array is 0-based)

\[
\begin{array}{c|cc|c}
B & < v & v & > v \\
\hline
C[0] + C[1] + \ldots + C[v] & \\
\end{array}
\]
Observation

- $\text{Ind}_B(v) = \text{Ind}_B(v - 1) + C[v]$
- Therefore, update $C$ such that $C'[i] = C[0] + C[1] + \ldots + C[i] - 1$.

<table>
<thead>
<tr>
<th>$C$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C'$</td>
<td>-1</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>
Counting Sort

Algorithm

\[
\begin{array}{cccccccc}
A & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
5 & 2_a & 4 & 6_a & 3 & 1 & 6_b & 2_b \\
\end{array}
\]

\[
\begin{array}{cccccccc}
C & & & & & & & \\
 & & & & & & & \\
B & & & & & & & \\
\end{array}
\]
Counting Sort

Algorithm

- Count (in array $C$) how often each $v$ is in $A$.

\[
\begin{array}{cccccccc}
\text{A} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
5 & 2_a & 4 & 6_a & 3 & 1 & 6_b & 2_b \\
\end{array}
\]

\[
\begin{array}{cccccccc}
\text{C} & 0 & 1 & 2 & 1 & 1 & 1 & 2 \\
\end{array}
\]
Counting Sort

Algorithm

- Count (in array $C$) how often each $v$ is in $A$.
- Update $C$ such that $\text{Ind}_B(v) = C[v]$. 

\[
\begin{array}{cccccccc}
A & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
5 & 2_a & 4 & 6_a & 3 & 1 & 6_b & 2_b \\
\end{array}
\]

\[
\begin{array}{cccccccc}
C & 0 & 1 & 2 & 3 & 4 & 5 & 7 \\
-1 & 0 & 2 & 3 & 4 & 5 & 7 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
B & & & & & & & \\
& & & & & & & \\
& & & & & & & \\
& & & & & & & \\
\end{array}
\]
Counting Sort

Algorithm

- Count (in array $C$) how often each $v$ is in $A$.
- Update $C$ such that $\text{Ind}_B(v) = C[v]$.
Counting Sort

Algorithm
- Count (in array $C$) how often each $v$ is in $A$.
- Update $C$ such that $\text{Ind}_B(v) = C[v]$.

<table>
<thead>
<tr>
<th>$A$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>$2_a$</td>
<td>4</td>
<td>$6_a$</td>
<td>3</td>
<td>1</td>
<td>$6_b$</td>
<td>$2_b$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$C$</th>
<th>−1</th>
<th>0</th>
<th>1</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>$B$</th>
<th>2$_b$</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>$6_b$</th>
</tr>
</thead>
</table>
Counting Sort

Algorithm

- Count (in array $C$) how often each $v$ is in $A$.
- Update $C$ such that $\text{Ind}_B(v) = C[v]$.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>5</td>
<td>$2_a$</td>
<td>4</td>
<td>$6_a$</td>
<td>3</td>
<td>1</td>
<td>$6_b$</td>
<td>$2_b$</td>
</tr>
<tr>
<td>$C$</td>
<td>$-1$</td>
<td>$-1$</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>$B$</td>
<td>1</td>
<td>$2_b$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$6_b$</td>
<td></td>
</tr>
</tbody>
</table>
Counting Sort

Algorithm

- Count (in array $C$) how often each $v$ is in $A$.
- Update $C$ such that $\text{Ind}_B(v) = C[v]$.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>5</td>
<td>$2_a$</td>
<td>4</td>
<td>$6_a$</td>
<td>3</td>
<td>1</td>
<td>$6_b$</td>
<td>$2_b$</td>
</tr>
<tr>
<td>$C$</td>
<td>$-1$</td>
<td>$-1$</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>$B$</td>
<td>1</td>
<td>$2_b$</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>$6_b$</td>
<td></td>
</tr>
</tbody>
</table>
Counting Sort

Algorithm

- Count (in array $C$) how often each $v$ is in $A$.
- Update $C$ such that $\text{Ind}_B(v) = C[v]$.
Counting Sort

Algorithm

- Count (in array C) how often each v is in A.
- Update C such that Ind_B(v) = C[v].
Counting Sort

Algorithm

- Count (in array $C$) how often each $v$ is in $A$.
- Update $C$ such that $\text{Ind}_B(v) = C[v]$.

\[
\begin{array}{cccccccc}
A & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
5 & 2_a & 4 & 6_a & 3 & 1 & 6_b & 2_b \\
\end{array}
\]

\[
\begin{array}{ccccccc}
C & -1 & -1 & 0 & 2 & 3 & 5 & 5 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
B & 1 & 2_a & 2_b & 3 & 4 & 6_a & 6_b \\
\end{array}
\]
Counting Sort

Algorithm

- Count (in array $C$) how often each $v$ is in $A$.
- Update $C$ such that $\text{Ind}_B(v) = C[v]$.
Counting Sort

Properties

- Runtime: $\mathcal{O}(n + k)$, i.e., $\mathcal{O}(n)$ if $k \in \mathcal{O}(n)$.
- Memory: $\mathcal{O}(n + k)$ (for $C$ and $B$)
- Stable (if implemented correctly)
Counting Sort

**Input:** An array $A$ such that $|A| = n$ and $A[i] \in \{0, 1, \ldots, k - 1\}$.

**Output:** An array $B$ containing the elements of $A$ in sorted order.

1. Create two arrays $B$ and $C$ with $|B| = n$, $|C| = k$ and initial value 0.
2. For $i := 0$ To $n - 1$
3. \[ C[A[i]] := C[A[i]] + 1 \]
4. $C[0] := C[0] - 1$
5. For $i := 1$ To $k - 1$
6. \[ C[i] := C[i] + C[i - 1] \]
7. For $i := n - 1$ DownTo 0
9. \[ C[A[i]] := C[A[i]] - 1 \]
Radix Sort
Radix Sort

Idea

- Imagine integers as sequence of $d$ digits with base $b$.
- Sort integers by digit.
- Start with the least significant digit and use a stable sort (e.g. counting sort).

\[
\begin{array}{cccc}
141 & 430 & 202 & 121 \\
224 & 141 & 304 & 141 \\
341 & 341 & 313 & 202 \\
304 & 121 & 121 & 224 \\
121 & 202 & 224 & 304 \\
430 & 313 & 430 & 313 \\
202 & 224 & 141 & 341 \\
313 & 304 & 341 & 430 \\
\end{array}
\]
Radix Sort – Runtime

Input

- $n$ integers in range $[0, k - 1]$ and a base $b$

Runtime

- Sorting by a single digit: $\mathcal{O}(n + b)$
- Sorting by all $d$ digits: $\mathcal{O}((n + b) \cdot d) = \mathcal{O}((n + b) \log_b k)$
- $\mathcal{O}(nc)$ time if $b = n$ and $k \leq n^c$
  linear if $c$ is constant

Other properties

- Stable
- $\mathcal{O}(n + b)$ additional memory is used.

We assume counting sort for sorting by digit.