All sorting algorithms we have seen so far are based on comparing elements.

- E.g., Insertion sort, Selection sort, Mergesort, Heapsort, and Quicksort.

Insertion sort, Selection sort, and Quicksort have worst-case running times \( \Theta(n^2) \), while the others have worst-case running time \( \Theta(n \log n) \).

**Question**
Can we do better?

**Goal**
We will prove that any comparison-based sorting algorithm has a worst-case running time \( \Omega(n \log n) \).
Sort \(< a_1, a_2, \ldots, a_n >\)

Each internal node is labeled \(a_i : a_j\) for \(\{1, 2, \ldots, n\}\)

- The left subtree shows subsequent comparisons if \(a_i \leq a_j\)
- The right subtree shows subsequent comparisons if \(a_i > a_j\)

Each leaf corresponds to an input ordering
Each internal node is labeled $a_i : a_j$ for \{1, 2, \ldots, n\}

- The left subtree shows subsequent comparisons if $a_i \leq a_j$
- The right subtree shows subsequent comparisons if $a_i > a_j$

Each leaf corresponds to an input ordering
Each internal node is labeled $a_i : a_j$ for $\{1, 2, \ldots, n\}$
- The left subtree shows subsequent comparisons if $a_i \leq a_j$
- The right subtree shows subsequent comparisons if $a_i > a_j$

Each leaf corresponds to an input ordering.
Decision-tree Example

Sort \(< a_1, a_2, a_3 >\)
\(= < 6, 8, 5 >:\)

Each internal node is labeled \(a_i : a_j\) for \(\{1, 2, \ldots, n\}\)

- The left subtree shows subsequent comparisons if \(a_i \leq a_j\)
- The right subtree shows subsequent comparisons if \(a_i > a_j\)

Each leaf corresponds to an input ordering
Each internal node is labeled $a_i : a_j$ for $\{1, 2, \ldots, n\}$

- The left subtree shows subsequent comparisons if $a_i \leq a_j$
- The right subtree shows subsequent comparisons if $a_i > a_j$

Each leaf corresponds to an input ordering.
A decision tree can model the execution of any comparison-based sorting algorithm

- One tree for each input size $n$
- Worst-case running time = height of tree
Theorem

Any comparison-based sorting algorithm requires $\Omega(n \log n)$ comparisons in the worst case.

Proof.

- A decision tree to sort $n$ elements must have at least $n!$ leaves, since each of the $n!$ orderings is a possible answer.
- A binary tree of height $h$ has at most $2^h$ leaves.
- Thus, $n! \leq 2^h$
  \[ h \geq \log n! = \Omega(n \log n) \quad \text{(Stirling’s approximation)} \]

Corollary

Heapsort and merge sort are asymptotically optimal comparison-based sorting algorithms.
We just proved that worst case number of comparisons used is $\Omega(n \log n)$.

Suppose that each of the $n!$ input permutations is equally likely. What can be said about the average case running time?

*Note: Average is taken by adding up individual running time of algorithm on each possible input and dividing total by $n!$.*

**Theorem**

When all input permutations are equally likely, any comparison-based sorting algorithm requires $\Omega(n \log n)$ comparisons on average.
**Theorem**

*When all input permutations are equally likely, any comparison-based sorting algorithm requires $\Omega(n \log n)$ comparisons on average.*

**Proof.**

- The *External Path Length (EPL)* of a tree is the sum over all leaves of the tree, of the length of the paths from the root to the leaves.
- Average number of comparisons used by a sorting algorithm is EPL of its associated comparison tree divided by $n!$.
- The EPL of a binary tree with $m$ leaves is at least $m \log_2 m + O(m)$.
- The comparison tree has $m = n!$ leaves
  - $\Rightarrow$ its external path length is $n! \log_2 n! + O(n!)$
  - $\Rightarrow$ average number of comparisons used is $\log_2 n! + O(1)$.
- We already saw $\log_2 n! = \Omega(n \log n)$. 
Can we do better?

Are there sorting algorithms which are not comparison-based? Can they beat the $\Omega(n \log n)$ lower bound?

- Counting sort
  - Assumes items are in set $\{1, 2, \ldots, k\}$.
  - Is a *stable* sort (defined soon).

- Radix sort
  - Assumes items are stored in fixed size words using finite alphabet
Counting Sort

Counting-sort(\(A, B, k\))

**Input:** \(A[1 \ldots n]\), where \(A[j] \in \{1, 2, \ldots, k\}\)

**Output:** \(B[1 \ldots n]\), sorted

let \(C[1 \ldots k]\) be a new array;

for \(i \leftarrow 1 \text{ to } k\) do
  \(C[i] \leftarrow 0;\)
end

for \(j \leftarrow 1 \text{ to } n\) do
  \(C[A[j]] \leftarrow C[A[j]] + 1; \quad // \quad C[i] = |\{\text{key} = i\}|\)
end

for \(i \leftarrow 2 \text{ to } k\) do
  \(C[i] \leftarrow C[i] + C[i - 1]; \quad // \quad C[i] = |\{\text{key} \leq i\}|\)
end

for \(j \leftarrow n \text{ to } 1\) do
  \(B[C[A[j]]] \leftarrow A[j];\)
  \(C[A[j]] \leftarrow C[A[j]] - 1;\)
end
### Example: Counting Sort

#### Array $A$:

<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>2</td>
<td>1</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

#### Array $B$:

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$B$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Array $C$:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example: Counting Sort

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

$A = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 4 & 2 & 1 & 4 & 2 \end{bmatrix}$

$B = \begin{bmatrix} & & & & \\ & & & & \end{bmatrix}$

$C = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & 0 & 0 & 0 \end{bmatrix}$
Example: Counting Sort

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

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

\[
\text{for } j \leftarrow 1 \text{ to } n \text{ do }
\]
\[
\text{C}[A[j]] \leftarrow C[A[j]] + 1; \quad // \quad C[i] = |\{\text{key} = i\}|
\]
\[
\text{end}
\]
Example: Counting Sort

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

<table>
<thead>
<tr>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

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

\[
\text{for } j \leftarrow 1 \text{ to } n \text{ do } \\
\quad C[A[j]] \leftarrow C[A[j]] + 1; \quad // \quad C[i] = |\{\text{key} = i\}|
\]

end
Example: Counting Sort

\[
\begin{array}{cccccc}
A & 1 & 2 & 3 & 4 & 5 \\
 & 4 & 2 & 1 & 4 & 2 \\
\end{array}
\quad
\begin{array}{cccc}
C & 1 & 2 & 3 & 4 \\
 & 1 & 1 & 0 & 1 \\
\end{array}
\]

\[
B
\]

\[
\text{for } j \leftarrow 1 \text{ to } n \text{ do}
\]
\[
\quad C[A[j]] \leftarrow C[A[j]] + 1; \quad \text{// } C[i] = |\{\text{key} = i\}|
\]
end
Example: Counting Sort

```
for j ← 1 to n do
    C[A[j]] ← C[A[j]] + 1; // C[i] = |{key = i}|
end
```
Example: Counting Sort

for $j \leftarrow 1$ to $n$ do
   $C[A[j]] \leftarrow C[A[j]] + 1$; // $C[i] = |\{\text{key} = i\}|$
end
Example: Counting Sort

\[
A = \begin{array}{c|c|c|c|c|c}
1 & 2 & 3 & 4 & 5 \\
\hline
4 & 2 & 1 & 4 & 2 \\
\end{array}
\]

\[
C = \begin{array}{c|c|c|c|c}
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 0 & 2 \\
\end{array}
\]

\[
B = \begin{array}{c|c|c|c|c}
\hline
\end{array}
\]

\[
C' = \begin{array}{c|c|c|c|c}
1 & 3 & 0 & 2 \\
\hline
\end{array}
\]

\[
\text{for } i \gets 2 \text{ to } k \text{ do}
\]
\[
\quad C[i] \gets C[i] + C[i - 1]; // C[i] = |\{\text{key} \leq i\}|
\]
\[
\text{end}
\]
Example: Counting Sort

for $i ← 2$ to $k$ do
    $C[i] ← C[i] + C[i − 1]$;  //  $C[i] = |\{\text{key} ≤ i\}|$
end
**Example: Counting Sort**

<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>2</td>
<td>1</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>0</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>3</th>
<th>3</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>C’</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```for i ← 2 to k do
   C[i] ← C[i] + C[i − 1];  // C[i] = |\{key ≤ i\}|
end```

Sorting: Lower Bounds and Linear Time

Last Revision: August 25, 2016
Example: Counting Sort

$$\begin{array}{lllll}
A & 1 & 2 & 3 & 4 & 5 \\
   & 4 & 2 & 1 & 4 & 2
\end{array}$$

$$\begin{array}{llllll}
B & 1 & 2 & 3 & 5 \\
   & 2
\end{array}$$

$$\begin{array}{lllll}
C & 1 & 2 & 3 & 4 \\
   & 1 & 3 & 3 & 5
\end{array}$$

$$\begin{array}{llll}
C' & 1 & 2 & 3 & 5 \\
   & & & &
\end{array}$$

```
for j ← n to 1 do
B[C[A[j]]] ← A[j];
C[A[j]] ← C[A[j]] − 1;
end
```
Example: Counting Sort

\[
\begin{array}{ccccc}
1 & 2 & 3 & 4 & 5 \\
4 & 2 & 1 & 4 & 2 \\
\end{array}
\]

\[
\begin{array}{ccccc}
1 & 2 & 3 & 4 & 5 \\
1 & 3 & 3 & 5 & 4 \\
\end{array}
\]

\[
\begin{array}{ccccc}
B & & & & \\
 & 2 & & 4 & \\
\end{array}
\]

\[
\begin{array}{ccccc}
C & & & & \\
1 & 3 & 3 & 5 & \\
\end{array}
\]

\[
\begin{array}{ccccc}
C' & & & & \\
1 & 2 & 3 & 4 & \\
\end{array}
\]

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

\[
\begin{aligned}
B[C[A[j]]] &\leftarrow A[j]; \\
C[A[j]] &\leftarrow C[A[j]] - 1;
\end{aligned}
\]

end
Example: Counting Sort

\[
\begin{array}{lllll}
A & 1 & 2 & 3 & 4 & 5 \\
  & 4 & 2 & 1 & 4 & 2 \\
B & 1 & 2 & 4 & 4 & 2 \\
\end{array}
\]

\[
\begin{array}{lllll}
C & 1 & 2 & 3 & 4 & 5 \\
 & 1 & 3 & 3 & 5 & 5 \\
C' & 0 & 2 & 3 & 3 & 4 \\
\end{array}
\]

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

\[
\begin{align*}
B[C[A[j]]] & \leftarrow A[j]; \\
C[A[j]] & \leftarrow C[A[j]] - 1;
\end{align*}
\]
end
for $j \leftarrow n$ to 1 do
  $C[A[j]] \leftarrow C[A[j]] - 1$;
end
Example: Counting Sort

\[
\begin{align*}
A & = \begin{bmatrix} 4 & 2 & 1 & 4 & 2 \end{bmatrix} \\
B & = \begin{bmatrix} 1 & 2 & 2 & 4 & 4 \end{bmatrix} \\
C & = \begin{bmatrix} 1 & 3 & 3 & 5 \end{bmatrix} \\
C' & = \begin{bmatrix} 0 & 1 & 3 & 3 \end{bmatrix}
\end{align*}
\]

\[
\text{for } j \leftarrow n \text{ to } 1 \text{ do} \\
\quad B[C[A[j]]] \leftarrow A[j]; \\
\quad C[A[j]] \leftarrow C[A[j]] - 1;
\end{align*}
\]

end
Input: $A[1\ldots n]$, where $A[j] \in \{1, 2, \ldots, k\}$
Output: $B[1\ldots n]$, sorted

let $C[1\ldots k]$ be a new array;

for $i \leftarrow 1$ to $k$ do
    $C[i] \leftarrow 0$; // $O(k)$
end

for $j \leftarrow 1$ to $n$ do
    $C[A[j]] \leftarrow C[A[j]] + 1$; // $O(n)$
end

for $i \leftarrow 2$ to $k$ do
    $C[i] \leftarrow C[i] + C[i - 1]$; // $O(k)$
end

for $j \leftarrow n$ to 1 do
    $C[A[j]] \leftarrow C[A[j]] - 1$; // $O(n)$
end

Total: $O(n + k)$
If $k = O(n)$, then counting sort takes $O(n)$ time.

- But didn’t we prove that sorting must take $\Omega(n \log n)$ time?
- No, actually we proved that any comparison-based sorting algorithm takes $\Omega(n \log n)$ time.
- Note that counting sort is not a comparison-based sorting algorithm.
- In fact, it makes no comparisons at all!
Counting sort is a stable sort

- it preserves the input order among equal elements.

Exercise

What other sorts have this property?
Sort on *least significant* digit first using stable sort
Induction on digit position

- Assume that the numbers are sorted by their low-order \( i - 1 \) digits
- Sort on digit \( i \)
Induction on digit position

- Assume that the numbers are sorted by their low-order $i - 1$ digits
- Sort on digit $i$
  - Two numbers that differ on digit $i$ are correctly sorted by their low-order $i$ digits
Radix Sort: Correctness

*Induction on digit position*

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

* Sort on digit \( i \)
  * Two numbers that differ on digit \( i \) are correctly sorted by their low-order \( i \) digits
  * Two numbers equal on digit \( i \) are put in the same order as the input \( \Rightarrow \) correctly sorted by their low-order \( i \) digits
Lemma

Given $n$ $d$-digit numbers in which each digit can take on up to $k$ possible values, radix sort correctly sorts these numbers in $O(d(n + k))$ time if the stable sort it uses takes $O(n + k)$ time.

Application:
Sorting numbers in the range from 0 to $n^b - 1$, where $b$ is a constant

- $b \log n$ bits for each number
- each number can be viewed as having $O(b)$ digits of $\log n$ bits each
- running time is $O(d(n + k)) = O(b(n + 2^{\log n})) = O(bn)$
- since $b$ is a constant, the running time is $O(n)$. 