Lecture 21: Fundamentals of Sorting I
What is sorting?

- Input: sequence of $n$ comparable values
- Reorder the input to be non-descending.
- Items we wish to sort are called “keys”
- Not here: retain associated information
Why discuss sorting?

- Standard library has sorting
- Why not use that and move on?

In this class, sorting is:

- a good intro for techniques
- a good intro to comparative algorithms
**SelectionSort**

**Idea:** Swap min into first spot, second-min to second, etc.  
(This is hand-wavy on purpose)

<table>
<thead>
<tr>
<th></th>
<th>85</th>
<th>24</th>
<th>63</th>
<th>45</th>
<th>17</th>
<th>31</th>
<th>96</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>17</td>
<td>24</td>
<td>63</td>
<td>45</td>
<td>85</td>
<td>31</td>
<td>96</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>17</td>
<td>24</td>
<td>63</td>
<td>45</td>
<td>85</td>
<td>31</td>
<td>96</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>17</td>
<td>24</td>
<td>31</td>
<td>45</td>
<td>85</td>
<td>63</td>
<td>96</td>
<td>50</td>
</tr>
</tbody>
</table>
SelectionSort

\[
\text{for } i \leftarrow 1 \text{ to } n - 1 \text{ do } \\
\quad \text{min } \leftarrow i \\
\text{for } j \leftarrow i + 1 \text{ to } n \text{ do } \\
\quad \text{if } A[j] < A[\text{min}] \text{ then } \\
\quad \quad \text{min } \leftarrow j \\
\text{Swap } A[i] \text{ and } A[\text{min}] \\
\]
What’s nice about SelectionSort?

- Easy to program
- Easy to explain
- Does it waste memory?  
  - only $O(1)$ "in place"
- Does it only work for numbers?
- What other info do we need?
- Are there inputs that are sorted faster?
- Is there a lot of data movement?  
  $2(n-1)$?  a lot??
### BubbleSort

**Idea:** Think globally act locally

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>85</td>
<td>24</td>
<td>63</td>
<td>45</td>
<td>17</td>
<td>31</td>
<td>96</td>
<td>50</td>
</tr>
<tr>
<td>24</td>
<td><strong>63</strong></td>
<td>45</td>
<td>17</td>
<td>31</td>
<td>85</td>
<td>50</td>
<td><strong>96</strong></td>
</tr>
<tr>
<td>24</td>
<td><strong>45</strong></td>
<td>17</td>
<td>31</td>
<td>63</td>
<td>50</td>
<td>85</td>
<td><strong>96</strong></td>
</tr>
<tr>
<td>24</td>
<td>17</td>
<td>31</td>
<td>45</td>
<td>50</td>
<td><strong>63</strong></td>
<td>85</td>
<td><strong>96</strong></td>
</tr>
</tbody>
</table>
BubbleSort

for $i \leftarrow 1$ to $n - 1$ do
  for $j \leftarrow 1$ to $n - i$ do
      Swap $A[j]$ and $A[j + 1]$

If no swap inside inner loop, sorted already.
# InsertionSort

**Idea:**

<table>
<thead>
<tr>
<th>85</th>
<th>24</th>
<th>63</th>
<th>45</th>
<th>17</th>
<th>31</th>
<th>96</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>85</td>
<td>63</td>
<td>45</td>
<td>17</td>
<td>31</td>
<td>96</td>
<td>50</td>
</tr>
<tr>
<td>24</td>
<td>63</td>
<td>85</td>
<td>45</td>
<td>17</td>
<td>31</td>
<td>96</td>
<td>50</td>
</tr>
<tr>
<td>24</td>
<td>45</td>
<td>63</td>
<td>85</td>
<td>17</td>
<td>31</td>
<td>96</td>
<td>50</td>
</tr>
</tbody>
</table>
\textbf{InsertionSort}

\begin{verbatim}
for j ← 2 to n do
  key ← A[j]
  i ← j − 1
  while i > 0 and A[i] > key do
    A[i + 1] ← A[i]
    i = i − 1
  A[i + 1] ← key
\end{verbatim}

What is the running time of InsertionSort?

$O(n + \alpha)$

\# inverted pairs in input
InsertionSort

\[
\textbf{for } j \leftarrow 2 \text{ to } n \textbf{ do}
\]
\[
\text{key } \leftarrow A[j]
\]
\[
i \leftarrow j - 1
\]
\[
\textbf{while } i > 0 \text{ and } A[i] > \text{key} \textbf{ do}
\]
\[
A[i + 1] \leftarrow A[i]
\]
\[
i = i - 1
\]
\[
A[i + 1] \leftarrow \text{key}
\]

Why is InsertionSort correct?

What is true every time we check the \textbf{for} loop?
(including the time we find \( j > n \) and stop)
About that running time ...

- Why are we so concerned with worst case?
- Why not examine average case?