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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>63</td>
<td>85</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
for $j \leftarrow 2$ to $n$ do
key $\leftarrow A[j]$
$i \leftarrow j - 1$
while $i > 0$ and $A[i] >$ key do
    $A[i + 1] \leftarrow A[i]$
    $i = i - 1$
$A[i + 1] \leftarrow$ key

What is the running time of InsertionSort?

$\Theta(n + d)$

Each iteration fixes one inverted pair from the input.

Fewest inv pairs input: zero

Most inverted pairs: $(\Omega)$
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 for loop?
(including the time we find \( j > n \) and stop)

\[
A[1...j-1] \text{ is sorted AND is the elements that began there}
\]
About that running time ...

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

Expected number of inverted pairs?

\[
\frac{1}{2} \binom{n}{2}
\]

Potentially inverted

\[
\frac{1}{2} \cdot n(n-1)
\]
HeapSort

**Idea:** Use a max heap.

- Find max, put max at end
- Then second-max, etc.
- Use the yet-to-be-sorted array as max heap

**Heapify:** make array into max heap
- Idea 1: insert each into growing heap

```python
H = empty max heap
for i = 1..n
    H.insert(A[i]) # O(n log n)

for j = n..1
    A[j] = H.extractMax() # O(n log n)
```

*not "in place"*
Heapify: Better way

- Treat array as heap. Where are leaf nodes?
- What should we do with non-leaf nodes?
- In which order?
How much work to heapify?

how many nodes? work each?

\[
\frac{n}{2}
\]

0

\[
\frac{n}{4}
\]

1

\[
\frac{n}{8}
\]

2

\[
\frac{n}{16}
\]

3

\[
\frac{n}{32}
\]

4
How long to heapify?

- The cost to insert varies by height.
- Node at height $h$ costs $O(h)$.

How many nodes at height $h$?

\[ \sim \frac{n}{2^{h+1}} \]

- How many different height values are there?
  \[ O\ldots \log n \]

- Cost for total is:

\[
\sum_{h=0}^{\lfloor \log n \rfloor} \left[ \frac{h}{2^{h+1}} \right] O(h) = O\left(n \sum_{h=0}^{\lfloor \log n \rfloor} \frac{h}{2^h} \right)
\]

Using formula for infinite geometric series:

\[
\sum_{h=0}^{\infty} \frac{h}{2^h} = \frac{1/2}{(1 - 1/2)^2}
\]
And how, HeapSort