CMPSC250
Lecture 6: Selection and Insertion Sort

Prof. John Wenskovitch
02/03/2016
Selection Sort

- Find the smallest item in the array; put it first.
- Find the next smallest item; put it second.
- Repeat until you’ve reached the last item in the input array.

```c
for (int i = 0; i < N; i++) {
    int min = i;
    for (int j = i+1; j < N; j++) {
        if (less(a[j], a[min]))
            min = j;
    } //if
} //for
exch(a, i, min);
} //for
```
Selection Sort Visual

Trace of selection sort (array contents just after each exchange)
Selection Sort Evaluation

• **less()** compares
  – When $i = 1$, we compare it against the other $(n - 1)$ entries.
  – When $i = 2$, we compare it against the remaining $(n - 2)$ entries.
  – $(n - 1) + (n - 2) + \cdots + 2 + 1 = \sim \frac{n^2}{2}$ compares

• **exch()** exchanges
  – For each $i$ value, we do one exchange, swapping $a[i]$ with $a[min]$.
  – $1 + 1 + 1 + \cdots + 1 = n$ exchanges
Selection Sort Evaluation

Trace of selection sort (array contents just after each exchange)
Selection Sort Evaluation

• Run time is insensitive to input.
  – Finding the smallest item on iteration \( i \) does not give any information about the location of the smallest item in iteration \((i + 1)\).
  – Therefore, worst case = average case.

• Data movement is minimal.
  – Number of exchanges is linear w.r.t. array size.
  – No other sorting algorithm that we will consider has this property.
Insertion Sort

• Look at the current $a[i]$.
• Place it appropriately between items $a[0]$ to $a[i - 1]$, moving it left until it shouldn’t be moved further.
• Repeat until you’ve reached the last item in the input array.

```java
for (int i = 1; i < N; i++) {
    for (int j = i; j > 0 && less(a[j], a[j-1]); j--) {
        exch(a, j, j-1);
    }
}
```
### Insertion Sort Visual

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>S O R T E X A M P L E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>O S R T E X A M P L E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>O R S T E X A M P L E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>O R S T E X A M P L E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>E O R S T X A M P L E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>E O R S T X A M P L E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>A E O R S T X A M P L E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>A E M O R S T X P L E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>A E M O P R S T X L E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>A E L M O P R S T X E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>A E E L M O P R S T X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Trace of insertion sort (array contents just after each insertion)**

- **entries in gray**: do not move
- **entry in red**: is `a[j]`
- **entries in black**: moved one position right for insertion
**Insertion Sort Evaluation**

- **less()** compares
  - When \( i = 1 \), we compare it against a maximum of 1 previous entry.
  - When \( i = 2 \), we compare it against a maximum of 2 previous entries.
  - On average, assume we’re moving the new value halfway to the left.

\[
\frac{1 + 2 + \cdots + (n - 2) + (n - 1)}{2} = \frac{n^2}{4}
\]

- **exch()** exchanges
  - Since **exch()** is called in a loop limited by **less()** calls, the count is identical, \( \sim \frac{n^2}{4} \) exchanges
Insertion Sort Evaluation

• Now we have a worst case and a best case to consider:
  
  – **Worst case:** We need to move every letter the whole way to the left.
    
    • \(1 + 2 + \cdots + (n - 2) + (n - 1) = \sim \frac{n^2}{2}\) compares and exchanges.
  
  – **Best case:** We don’t need to move any letters – the array is already sorted, or all of the keys are identical.
    
    • \(1 + 1 + 1 + \cdots + 1 = n - 1\) compares and 0 exchanges.
Insertion Sort Evaluation

Trace of insertion sort (array contents just after each insertion)

- Entries in gray do not move.
- Entry in red is \(a[j]\).
- Entries in black moved one position right for insertion.
Insertion Sort Evaluation

• Run time and data movement are both sensitive to input.
  – The initial positions of the items has a significant impact on the run time of the algorithm, as well as how far each data item needs to move.
  – Insertion Sort works quite efficiently on data that is already almost sorted and just needs a few tweaks:
    • A small array appended to a large sorted array.
    • An array that was sorted and had a few values update.
Partially Sorted Array

• Inversion – A pair of entries that are out of order in an array.
  – “If the number of inversions is less than a constant multiple of the array size, we call the array partially sorted.”
  – In practice, let’s say halfway between average case and best case is partially sorted.
Comparing Sorting Algorithms

• So which is faster, Insertion Sort or Selection Sort?
  – It depends...

• Compare algorithms by:
  1. Implementing and debugging them
  2. Analyzing their properties
  3. Hypothesizing about their performance
  4. Running experiments to validate the hypotheses
Comparing Sorting Algorithms

1. Implementing and Debugging
   – Done (algorithms)

2. Analyzing properties
   – Done (average/worst case analysis)

3. Hypotheses
   – //TODO

4. Experiments
   – //TODO
Comparing Sorting Algorithms

• **Hypothesis:** Since both Selection Sort and Insertion Sort run in $O(n^2)$ time, their performance is effectively the same, with variations of a constant factor, for randomly ordered arrays of distinct values.

• **Experiments:** Well, let’s find out...
Any Questions?