CMPS112
Lecture 34: Selection and Insertion Sort

Prof. John Wenskovitch
11/30/2015
Last Time

• Hash Tables
  – How do we handle two data items hashing into the same location?
    • Separate Chaining
    • Linear Probing
Why Does Sorting Matter?

• “About a quarter of all computer cycles are spent sorting.” – Donald Knuth, 1973
  – Bank account transactions
  – Search engine results
  – Scientific computations – astrophysics, molecular dynamics, weather prediction, linguistics

• Suitable “prototype problem” – easily modeled and has good mathematical properties.

• The first step towards organizing and evaluating data is often to sort it.
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.

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

<table>
<thead>
<tr>
<th>i</th>
<th>min</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>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>SORTEXAMPLE</td>
<td>SORTEXAMPLE</td>
<td>SORTEXAMPLE</td>
<td>SORTEXAMPLE</td>
<td>SORTEXAMPLE</td>
<td>SORTEXAMPLE</td>
<td>SORTEXAMPLE</td>
<td>SORTEXAMPLE</td>
<td>SORTEXAMPLE</td>
<td>SORTEXAMPLE</td>
<td>SORTEXAMPLE</td>
</tr>
<tr>
<td>0</td>
<td>6</td>
<td>SORTEXAMPLE</td>
<td>SORTEXAMPLE</td>
<td>SORTEXAMPLE</td>
<td>SORTEXAMPLE</td>
<td>SORTEXAMPLE</td>
<td>SORTEXAMPLE</td>
<td>SORTEXAMPLE</td>
<td>SORTEXAMPLE</td>
<td>SORTEXAMPLE</td>
<td>SORTEXAMPLE</td>
<td>SORTEXAMPLE</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>AORTEXAMPLE</td>
<td>AORTEXAMPLE</td>
<td>AORTEXAMPLE</td>
<td>AORTEXAMPLE</td>
<td>AORTEXAMPLE</td>
<td>AORTEXAMPLE</td>
<td>AORTEXAMPLE</td>
<td>AORTEXAMPLE</td>
<td>AORTEXAMPLE</td>
<td>AORTEXAMPLE</td>
<td>AORTEXAMPLE</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>AERTOXAMPLE</td>
<td>AERTOXAMPLE</td>
<td>AERTOXAMPLE</td>
<td>AERTOXAMPLE</td>
<td>AERTOXAMPLE</td>
<td>AERTOXAMPLE</td>
<td>AERTOXAMPLE</td>
<td>AERTOXAMPLE</td>
<td>AERTOXAMPLE</td>
<td>AERTOXAMPLE</td>
<td>AERTOXAMPLE</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>AEETOXSAMPLE</td>
<td>AEETOXSAMPLE</td>
<td>AEETOXSAMPLE</td>
<td>AEETOXSAMPLE</td>
<td>AEETOXSAMPLE</td>
<td>AEETOXSAMPLE</td>
<td>AEETOXSAMPLE</td>
<td>AEETOXSAMPLE</td>
<td>AEETOXSAMPLE</td>
<td>AEETOXSAMPLE</td>
<td>AEETOXSAMPLE</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>AELOXSMPTRE</td>
<td>AELOXSMPTRE</td>
<td>AELOXSMPTRE</td>
<td>AELOXSMPTRE</td>
<td>AELOXSMPTRE</td>
<td>AELOXSMPTRE</td>
<td>AELOXSMPTRE</td>
<td>AELOXSMPTRE</td>
<td>AELOXSMPTRE</td>
<td>AELOXSMPTRE</td>
<td>AELOXSMPTRE</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>AEELMXSOPTR</td>
<td>AEELMXSOPTR</td>
<td>AEELMXSOPTR</td>
<td>AEELMXSOPTR</td>
<td>AEELMXSOPTR</td>
<td>AEELMXSOPTR</td>
<td>AEELMXSOPTR</td>
<td>AEELMXSOPTR</td>
<td>AEELMXSOPTR</td>
<td>AEELMXSOPTR</td>
<td>AEELMXSOPTR</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>AEELMOSXPTRE</td>
<td>AEELMOSXPTRE</td>
<td>AEELMOSXPTRE</td>
<td>AEELMOSXPTRE</td>
<td>AEELMOSXPTRE</td>
<td>AEELMOSXPTRE</td>
<td>AEELMOSXPTRE</td>
<td>AEELMOSXPTRE</td>
<td>AEELMOSXPTRE</td>
<td>AEELMOSXPTRE</td>
<td>AEELMOSXPTRE</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td>AEELMOPXSSTR</td>
<td>AEELMOPXSSTR</td>
<td>AEELMOPXSSTR</td>
<td>AEELMOPXSSTR</td>
<td>AEELMOPXSSTR</td>
<td>AEELMOPXSSTR</td>
<td>AEELMOPXSSTR</td>
<td>AEELMOPXSSTR</td>
<td>AEELMOPXSSTR</td>
<td>AEELMOPXSSTR</td>
<td>AEELMOPXSSTR</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>AEELMOPRXSTX</td>
<td>AEELMOPRXSTX</td>
<td>AEELMOPRXSTX</td>
<td>AEELMOPRXSTX</td>
<td>AEELMOPRXSTX</td>
<td>AEELMOPRXSTX</td>
<td>AEELMOPRXSTX</td>
<td>AEELMOPRXSTX</td>
<td>AEELMOPRXSTX</td>
<td>AEELMOPRXSTX</td>
<td>AEELMOPRXSTX</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>AEELMOPRSTXX</td>
<td>AEELMOPRSTXX</td>
<td>AEELMOPRSTXX</td>
<td>AEELMOPRSTXX</td>
<td>AEELMOPRSTXX</td>
<td>AEELMOPRSTXX</td>
<td>AEELMOPRSTXX</td>
<td>AEELMOPRSTXX</td>
<td>AEELMOPRSTXX</td>
<td>AEELMOPRSTXX</td>
<td>AEELMOPRSTXX</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>AEEELMOPRSTXX</td>
<td>AEEELMOPRSTXX</td>
<td>AEEELMOPRSTXX</td>
<td>AEEELMOPRSTXX</td>
<td>AEEELMOPRSTXX</td>
<td>AEEELMOPRSTXX</td>
<td>AEEELMOPRSTXX</td>
<td>AEEELMOPRSTXX</td>
<td>AEEELMOPRSTXX</td>
<td>AEEELMOPRSTXX</td>
<td>AEEELMOPRSTXX</td>
</tr>
</tbody>
</table>

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

- entries in black are examined to find the minimum
- entries in red are a[min]
- entries in gray are in final position
Selection Sort Evaluation

• 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 $a[\text{min}]$.
  – $1 + 1 + 1 + \cdots + 1 = n$ exchanges
Selection Sort Evaluation

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

Entries in black are examined to find the minimum.

Entries in red are $a[\text{min}]$.

Entries in gray are in final position.

Selection and Insertion Sort
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.

\[
\text{for (int } i = 1; i < N; i++) \{ \\
\quad \text{for (int } j = i; j > 0 \text{ && } (a[j] < a[j-1]); j--) \{ \\
\quad\quad \text{exch}(a, j, j-1); \\
\quad \} //\text{for} \\
\} //\text{for} 
\]
Insertion Sort Visual

Trace of insertion sort (array contents just after each insertion)
Insertion Sort Evaluation

• 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.
  – $[1 + 2 + \cdots + (n - 2) + (n - 1)]/2 = \sim \frac{n^2}{4}$ compares

• `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)
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.
Comparing Sorting Algorithms

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