Today

- Quiz 6
- Selection sort (cont)
- Insertion sort

Assignments

- HW6 out (due after Spring Break)
Selection Sort

The basic idea

- Select (i.e., “find”) the largest item
- Swap it with the last item in the list
- Repeat with items 0 to \( n - 2 \)
- Stop when only one item left

Pass 1:

\[
\begin{array}{cccc}
29 & 10 & 14 & 13 \\
\end{array}
\Rightarrow
\begin{array}{cccc}
13 & 10 & 14 & 29 \\
\end{array}
\]

Pass 2:

\[
\begin{array}{cccc}
13 & 10 & 14 & 29 \\
\end{array}
\Rightarrow
\begin{array}{cccc}
13 & 10 & 14 & 29 \\
\end{array}
\]

Pass 3:

\[
\begin{array}{cccc}
13 & 10 & 14 & 29 \\
\end{array}
\Rightarrow
\begin{array}{cccc}
10 & 13 & 14 & 29 \\
\end{array}
\]

The algorithm (pseudo code):

\[
\text{SelectionSort}: \text{T array[]} , \text{int n} \\
1. \text{for i = n - 1 to 1:} \quad \# \text{n-1 passes} \\
2. \quad \text{index := 0} \\
3. \text{for j = 1 to i:} \quad \# \text{i iterations} \\
4. \quad \text{if array[j] > array[index] then} \quad \# \text{1 comparison} \\
5. \quad \quad \text{index := j} \quad \# \text{1 move} \\
6. \quad \text{swap(array[index], array[i])} \quad \# \text{3 moves}
\]

Exercise: Give a trace of the algorithm on the input \{29, 10, 14, 13\}
Choices for evaluating selection sort:

- Total number of operations ... a lot of (unnecessary) work
- Number of comparisons = pairs of items compared
- Number of moves = moving list items (including “swaps”)

Often useful to consider comparisons and moves separately for sorting

- we’ll see examples as we go

Q: how are swaps implemented?

```cpp
swap(T& x, T& y)
1. T tmp := x
2. x := y
3. y := tmp
```

Note this requires 3 moves!
Q: What is the best and worst case?

- reverse order causes more index assignments ... however
- there isn’t an order of magnitude difference for selection sort

Q: What is the cost? ... using comparisons

- The outer loop executes \( n - 1 \) times (i.e., “passes”)
- Each pass \( p \) requires \( n - p \) comparisons (e.g., 1st is \( n - 1 \) comparisons, etc.)
- Note that:

\[
\sum_{i=1}^{n} i = 1 + 2 + \cdots + n = \frac{n(n+1)}{2}
\]

- Putting it all together:

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

- And so selection sort is \( O(n^2) \)
Note that we get a different result when counting moves instead ...

\[ \sum_{p=1}^{n-1} 3 = 3(n - 1) = 3n - 3 \]

- Which is \(O(n)\)
- Each pass requires only one swap (which is three moves)
- However, the overall time is still \(O(n^2)\)

<table>
<thead>
<tr>
<th>Sorting Alg</th>
<th>Best Case</th>
<th>Avg Case</th>
<th>Worst Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection Sort</td>
<td>(O(n^2))</td>
<td>(O(n^2))</td>
<td>(O(n^2))</td>
</tr>
<tr>
<td>Bubble Sort</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Insertion Sort</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Merge Sort</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quick Sort</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tree Sort</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Heap Sort</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Selection Sort for HW6

- Instead of “moving” (swapping) node values, “splice” nodes into correct spot.
- We’ll also do selection sort “backwards” ...
  - find smallest each pass, insert in front of unsorted region.
  - why do it this way for a linked list? ... less list navigation.

Example ...

Input:

```
  head  e, 5   c, 3   a, 1   d, 2

Pass 1: Splice a before c
  head  a, 1   e, 5   c, 3   d, 2

Pass 2: Splice c before e
  head  a, 1   c, 3   e, 5   d, 2

Pass 3: Splice d before e
  head  a, 1   c, 3   d, 2   e, 5
```
(Linear) Insertion Sort

The basic idea (note: beginning of list is sorted, rest is unsorted)

- Select first item (in unsorted region)
- Insert item into correct position in sorted region (shifting larger items over)
- Stop when unsorted region is empty

Pass 1:

\[
\begin{array}{cccc}
29 & 10 & 14 & 13 \\
\end{array} \Rightarrow \begin{array}{cccc}
10 & 29 & 14 & 13 \\
\end{array}
\]

Pass 2:

\[
\begin{array}{cccc}
10 & 29 & 14 & 13 \\
\end{array} \Rightarrow \begin{array}{cccc}
10 & 14 & 29 & 13 \\
\end{array}
\]

Pass 3:

\[
\begin{array}{cccc}
10 & 14 & 29 & 13 \\
\end{array} \Rightarrow \begin{array}{cccc}
10 & 13 & 14 & 29 \\
\end{array}
\]
Insertion Sort for HW6

- Instead of “moving” (shifting) nodes, we’ll **splice** them into the correct spot

Example ...

```
Input:

head -> e, 5 -> c, 3 -> a, 1 -> d, 2

Pass 1: Splice c before e
head -> c, 3 -> e, 5 -> a, 1 -> d, 2

Pass 2: Splice a before c
head -> a, 1 -> c, 3 -> e, 5 -> d, 2

Pass 3: Splice d before e
head -> a, 1 -> c, 3 -> d, 2 -> e, 5
```