Topic 24
sorting and searching arrays

"There's nothing in your head
the sorting hat can't see. So try
me on and I will tell you where
you ought to be."

- The Sorting Hat,
*Harry Potter and
the Sorcerer's Stone*

Searching

- Given an array of ints find the index of the
  first occurrence of a target int

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>89</td>
<td>0</td>
<td>27</td>
<td>-5</td>
<td>42</td>
<td>11</td>
</tr>
</tbody>
</table>

- Given the above array and a target of 27 the
  method returns 2
- What if not present?
- What if more than one occurrence?

---

**clicker**

- Given an array with 1,000,000 distinct
  elements in random order, how many
  elements do you expect to look at when
  searching if:

<table>
<thead>
<tr>
<th>item present</th>
<th>item not present</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. 0</td>
<td>1,000,000</td>
</tr>
<tr>
<td>B. 500,000</td>
<td>1,000,000</td>
</tr>
<tr>
<td>C. 1,000,000</td>
<td>1,000,000</td>
</tr>
<tr>
<td>D. 1,000</td>
<td>500,000</td>
</tr>
<tr>
<td>E. 20</td>
<td>1,000,000</td>
</tr>
</tbody>
</table>

---

**linear or sequential search**
Sorting

INEFFECTIVE Sorts

- Define half-heartedbubbleSort(list):
  - if length(list) < 2:
    - return list
  - pivot = int(length(list) / 2)
  - A = half-heartedbubbleSort(list[0:pivot])
  - B = half-heartedbubbleSort(list[pivot:])
  - return merge(A, B)

- Define quickSort(list):
  - if length(list) < 2:
    - return list
  - pivot = select(list)
  - left = quickSort(list[0:pivot])
  - right = quickSort(list[pivot:酊)
  - return merge(left, right)

- Define mergeSort(list):
  - if length(list) < 2:
    - return list
  - pivot = int(length(list) / 2)
  - left = mergeSort(list[0:pivot])
  - right = mergeSort(list[pivot:])
  - return merge(left, right)

A fundamental application for computers
Done to make finding data (searching) faster
Many different algorithms for sorting
One of the difficulties with sorting is working with a fixed size storage container (array)
  - if resize, that is expensive (slow)
  - Trying to apply a human technique of sorting can be difficult
  - try sorting a pile of papers and clearly write out the algorithm you follow

Selection Sort

- To sort a list into ascending order:
  - Find the smallest item in an array, the minimum
  - Put that value in the first element of the array
    - Where to put the value that was in the first location?
  - And now...?

http://tinyurl.com/d7kxxx
animation of selection sort algorithm
Implementation of Selection Sort

- Include println commands to trace the sort

Clicker

- Determine how long it takes to sort an array with 100,000 elements in random order using selection sort. When the number of elements is increased to 200,000 how long will it take to sort the array?
  A. About the same
  B. 1.5 times as long
  C. 2 times as long
  D. 4 times as long
  E. 16 times as long

Insertion Sort

- Another of the Simple sort
- The first item is sorted
- Compare the second item to the first
  - if smaller swap
- Third item, compare to item next to it
  - need to swap
  - after swap compare again
- And so forth...

Insertion Sort in Practice

44 68 191 119 119 37 83 82 191 45 158 130 76 153 39 25

http://tinyurl.com/d8spm2l
animation of insertion sort algorithm
**Binary Search**

- My client couldn't have killed anyone with this arrow, and I can prove it!
- I'd like to examine your proof, Zeno. You may approach the bench.
- But never reach it!

**Searching in a Sorted List**

- If items are sorted then we can divide and conquer.
- Dividing your work in half with each step: generally a good thing.
- The Binary Search on List in Ascending order:
  - Start at middle of list.
  - Is that the item?
  - If not, is it less than or greater than the item?
  - Less than, move to second half of list.
  - Greater than, move to first half of list.
  - Repeat until found or sub list size = 0.

**Implement Binary Search**

<table>
<thead>
<tr>
<th>List</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>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>11</td>
<td>13</td>
<td>17</td>
<td>19</td>
<td>23</td>
<td>29</td>
<td>31</td>
<td>37</td>
<td>41</td>
<td>43</td>
<td>47</td>
<td>53</td>
</tr>
</tbody>
</table>

- List: low item, middle item, high item.
- Is middle item what we are looking for? If not is it more or less than the target item? (Assume lower).
- List: low item, middle item, high item.
- and so forth…
Trace When Key == 3
Trace When Key == 30

Variables of Interest?

clicker

- Given an array with 1,000,000 elements in sorted order, how many elements do you expect to look at when searching (with binary search) for a value if:

<table>
<thead>
<tr>
<th></th>
<th>item present</th>
<th>item not present</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.</td>
<td>0</td>
<td>500,000</td>
</tr>
<tr>
<td>B.</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>C.</td>
<td>1</td>
<td>1,000,000</td>
</tr>
<tr>
<td>D.</td>
<td>1,000</td>
<td>500,000</td>
</tr>
<tr>
<td>E.</td>
<td>1,000</td>
<td>1,000</td>
</tr>
</tbody>
</table>