CSE 143
Lecture 6 (b)

Binary Search

reading: 13.1

slides created by Marty Stepp
http://www.cs.washington.edu/143/
Sequential search

- **sequential search**: Locates a target value in an array/list by examining each element from start to finish.
  - How many elements will it need to examine?
  - Example: Searching the array below for the value 42:

<table>
<thead>
<tr>
<th>index</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>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>-4</td>
<td>2</td>
<td>7</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>22</td>
<td>25</td>
<td>30</td>
<td>36</td>
<td>42</td>
<td>50</td>
<td>56</td>
<td>68</td>
<td>85</td>
<td>92</td>
<td>103</td>
</tr>
</tbody>
</table>

- Notice that the array is sorted. Could we take advantage of this?
### Binary search (13.1)

- **binary search**: Locates a target value in a *sorted* array/list by successively eliminating half of the array from consideration.

  - How many elements will it need to examine?
  - Example: Searching the array below for the value 42:

<table>
<thead>
<tr>
<th>index</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>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>-4</td>
<td>2</td>
<td>7</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>22</td>
<td>25</td>
<td>30</td>
<td>36</td>
<td>42</td>
<td>50</td>
<td>56</td>
<td>68</td>
<td>85</td>
<td>92</td>
<td>103</td>
</tr>
</tbody>
</table>

- **min**: The minimum value in the array.
- **mid**: The index where the search is performed.
- **max**: The maximum value in the array.
## The Arrays class

- **Class Arrays in java.util** has many useful array methods:

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>binarySearch(array, value)</code></td>
<td>returns the index of the given value in a <em>sorted</em> array (or &lt; 0 if not found)</td>
</tr>
<tr>
<td><code>binarySearch(array, minIndex, maxIndex, value)</code></td>
<td>returns index of given value in a <em>sorted</em> array between indexes min/max - 1 (&lt; 0 if not found)</td>
</tr>
<tr>
<td><code>copyOf(array, length)</code></td>
<td>returns a new resized copy of an array</td>
</tr>
<tr>
<td><code>equals(array1, array2)</code></td>
<td>returns true if the two arrays contain same elements in the same order</td>
</tr>
<tr>
<td><code>fill(array, value)</code></td>
<td>sets every element to the given value</td>
</tr>
<tr>
<td><code>sort(array)</code></td>
<td>arranges the elements into sorted order</td>
</tr>
<tr>
<td><code>toString(array)</code></td>
<td>returns a string representing the array, such as &quot;[10, 30, -25, 17]&quot;</td>
</tr>
</tbody>
</table>

- **Syntax:** `Arrays.methodName(parameters)`
Arrays.binarySearch

// searches an entire sorted array for a given value
// returns its index if found; a negative number if not found
// Precondition: array is sorted
Arrays.binarySearch(array, value)

// searches given portion of a sorted array for a given value
// examines minIndex (inclusive) through maxIndex (exclusive)
// returns its index if found; a negative number if not found
// Precondition: array is sorted
Arrays.binarySearch(array, minIndex, maxIndex, value)

• The binarySearch method in the Arrays class searches an array very efficiently if the array is sorted.
  – You can search the entire array, or just a range of indexes (useful for "unfilled" arrays such as the one in ArrayIntList)
  – If the array is not sorted, you may need to sort it first
Using `binarySearch`

```java
// index    0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15
int[] a = {-4, 2, 7, 9, 15, 19, 25, 28, 30, 36, 42, 50, 56, 68, 85, 92};

int index  = Arrays.binarySearch(a, 0, 16, 42); // index1 is 10
int index2 = Arrays.binarySearch(a, 0, 16, 21); // index2 is -7
```

- `binarySearch` returns the index where the value is found
- If the value is *not* found, `binarySearch` returns:
  
  ```java
  -(insertionPoint + 1)
  
  where `insertionPoint` is the index where the element *would* have been, if it had been in the array in sorted order.
  
  To insert the value into the array, negate `insertionPoint + 1`
  ```

  ```java
  int indexToInsert21 = -(index2 + 1); // 6
  ```