Sorting Arrays

- Sorting data is putting the data into some particular, meaningful order, such as ascending or descending.
- We’ll take a look at the following two sorts:
  - The bubble sort
  - The selection sort

Sorting Arrays: bubble sort

- The bubble sort is also known as the sinking sort.
- It makes multiple passes through the array, and on each pass it compares adjacent pairs of values, putting those individual pairs in order.
- After each pass, one more item that is in order will be at the end of the list. In other words, after the first pass, the last element will be in the correct position. After the second pass, the last two elements will be in their correct positions, and so on.
Bubble sort, first pass, step-by-step

<table>
<thead>
<tr>
<th>Step</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15</td>
<td>51</td>
<td>62</td>
<td>7</td>
<td>16</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>15</td>
<td>51</td>
<td>62</td>
<td>7</td>
<td>16</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td>51</td>
<td>62</td>
<td>7</td>
<td>16</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>51</td>
<td>7</td>
<td>62</td>
<td>16</td>
<td>12</td>
</tr>
<tr>
<td>5</td>
<td>15</td>
<td>51</td>
<td>7</td>
<td>62</td>
<td>16</td>
<td>12</td>
</tr>
</tbody>
</table>

Comparing positions: Step 0

- 1, 2: NO
- 2, 3: YES
- 3, 4: NO
- 4, 5: YES

Bubble sort, all passes

<table>
<thead>
<tr>
<th>Pass</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15</td>
<td>51</td>
<td>62</td>
<td>7</td>
<td>16</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>15</td>
<td>51</td>
<td>62</td>
<td>7</td>
<td>16</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td>7</td>
<td>62</td>
<td>16</td>
<td>12</td>
<td>51</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>12</td>
<td>16</td>
<td>12</td>
<td>51</td>
<td>62</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>12</td>
<td>16</td>
<td>12</td>
<td>51</td>
<td>62</td>
</tr>
</tbody>
</table>

Bubble sort code

```java
1. public class BubbleSort {
2.   public static void main ( String args[] ) {
3.     int[] a = { 15, 62, 51, 7, 16, 12 }; 
4.     showArray ( "original", a ); 
5.     bubSort ( a ); 
6.     showArray ( "final", a ); 
7. }
8. }
9. public static void showArray ( String label, int[] t ) { 
10.    System.out.println ( "
11.     |" + label + " | order: "); 
12.    for ( int item: t ) 
13.        System.out.printf ( "%4d", item ); 
14.    System.out.println (); 
15. }
16. }
```
Bubble sort code

1. public static void bubSort ( int foo[] ) { 
2. for ( int pass = 0; pass < foo.length - 1; pass ++ ) { 
3. for ( int pos = 0; pos < foo.length - 1; pos++ ) { 
4. if ( foo[pos] > foo[pos+1] ) { 
5. int temp = foo[pos]; 
6. foo[pos] = foo[pos+1]; 
7. foo[pos+1] = temp; 
8. }
9. }
10. showArray ("Pass "+(pass+1), foo);
11. }
12. }
13. }

The selection sort

• As the name implies, the selection sort puts items into a meaningful order by selecting the next item to put into its correct place.
• The selection sort starts by selecting the smallest element out of the entire list and swapping that value with the first element. After this first “pass” through the array, the smallest element is in the first position of the array.
• On the second time through the array, the smallest of the remaining elements (2nd element and up) is located and swapped with the second position. After this second “pass” through the array, the first two elements are in their correct positions.
• This continues up through the end of the array for the third, fourth, etc. elements.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>15</td>
<td>62</td>
<td>61</td>
<td>7</td>
<td>60</td>
<td>12</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>62</td>
<td>61</td>
<td>7</td>
<td>60</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>62</td>
<td>61</td>
<td>15</td>
<td>40</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>12</td>
<td>14</td>
<td>15</td>
<td>60</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>12</td>
<td>14</td>
<td>16</td>
<td>60</td>
<td>12</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>12</td>
<td>16</td>
<td>16</td>
<td>60</td>
<td>12</td>
</tr>
</tbody>
</table>

Selection sort, step-by-step
Selection sort code

1. public class SelectionSort {
2.   public static void main ( String args[] ) {
3.     int[] a = { 15, 62, 51, 7, 16, 12 }; 
4.     showArray ( "original", a ); 
5.     selectSort ( a );
6.     showArray ( "final", a );
7. }
8. public static void showArray ( String label, int[] t ) {
9.     System.out.println ( "\n" + label + " order: ");
10.    for ( int item: t )
11.        System.out.printf ( "%4d", item );
12.    System.out.println ( );
13. }
14. public static void selectSort ( int[] data ) {
15.    int smallest;
16.    for ( int pos = 0; pos < data.length - 1; pos ++ ) {
17.        smallest = pos;
18.        for ( int a = pos + 1; a < data.length; a++ )
19.            if ( data[a] < data [smallest] )
20.                smallest = a;
21.        int temp = data[pos];
22.        data[pos] = data[smallest];
23.        data[smallest] = temp;
24.        showArray ( "Pass #" + (pos+1), data );
25.    }
26. }
27. }

Big-O Analysis for sorting/searching

- Big-O notation is a means of comparing various algorithms to one another to find which is more efficient/productive.
- Basically, Big-O measures the amount of work that needs to be expended to complete the given algorithm. Work is typically measured by the most "expensive" action.
- While Big-O can be applied to any algorithm, algorithms of similar natures should be compared to one another (i.e., searching vs. searching, sorting vs. sorting).

<table>
<thead>
<tr>
<th>n elements</th>
<th>Linear Search</th>
<th>Binary Search</th>
<th>Bubble and Selection Sorts</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024 (2^10)</td>
<td>1024</td>
<td>10</td>
<td>1046276</td>
</tr>
<tr>
<td>1048576 (2^20)</td>
<td>1048576</td>
<td>20</td>
<td>109951162776</td>
</tr>
<tr>
<td>1073741824 (2^30)</td>
<td>1073741824</td>
<td>30</td>
<td>1159232310460084976</td>
</tr>
<tr>
<td>1099511627776 (2^40)</td>
<td>1099511627776</td>
<td>40</td>
<td>13211446048317479568</td>
</tr>
</tbody>
</table>