Bubble Sort

With thanks to Bill Leahy, GaTech for original slide deck
Sorting

- Sorting takes an unordered collection and makes it an ordered one.
Sorting

• Lots of algorithms for sorting
  – Selection sort (video lecture and textbook)
  – Insertion sort (textbook)
  – Bubble sort
  – Merge sort
  – Radix sort
  – ...
"Bubbling Up" the Largest Element

- Traverse a collection of elements (array)
  - Move from the front to the end
  - "Bubble" the largest value to the end using pair-wise comparisons and swapping

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>77</td>
<td>42</td>
<td>35</td>
<td>12</td>
<td>101</td>
<td>5</td>
</tr>
</tbody>
</table>
"Bubbling Up" the Largest Element

- Traverse a collection of elements
  - Move from the front to the end
  - "Bubble" the largest value to the end using pair-wise comparisons and swapping
"Bubbling Up" the Largest Element

• Traverse a collection of elements
  – Move from the front to the end
  – “Bubble” the largest value to the end using pair-wise comparisons and swapping
"Bubbling Up" the Largest Element

- Traverse a collection of elements
  - Move from the front to the end
  - “Bubble” the largest value to the end using pair-wise comparisons and swapping

```
0 1 2 3 4 5
42 35 12 77 101 5
```
"Bubbling Up" the Largest Element

- Traverse a collection of elements
  - Move from the front to the end
  - "Bubble" the largest value to the end using pair-wise comparisons and swapping

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>35</td>
<td>12</td>
<td><strong>77</strong></td>
<td><strong>101</strong></td>
<td>5</td>
</tr>
</tbody>
</table>

No need to swap
"Bubbling Up" the Largest Element

- Traverse a collection of elements
  - Move from the front to the end
  - "Bubble" the largest value to the end using pair-wise comparisons and swapping
"Bubbling Up" the Largest Element

• Traverse a collection of elements
  – Move from the front to the end
  – "Bubble" the largest value to the end using pair-wise comparisons and swapping

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>35</td>
<td>12</td>
<td>77</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Largest value correctly placed
The “Bubble Up” Algorithm

Given some array, a

for(int i = 0; i < a.length-1; i++) {
    if(a[i] > a[i+1]) {
        int temp = a[i];
        a[i] = a[i+1];
        a[i+1] = temp;
    }
}
Items of Interest

• Notice that only the largest value is correctly placed
• All other values are still out of order
• So we need to repeat this process

Largest value correctly placed
Repeat “Bubble Up” How Many Times?

• If we have N elements...

• And if each time we bubble an element, we place it in its correct location...

• Then we repeat the “bubble up” process $N - 1$ times.

• This guarantees we’ll correctly place all N elements.
“Bubbling” All the Elements

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>42</td>
<td>35</td>
<td>12</td>
<td>77</td>
<td>5</td>
<td>101</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>35</td>
<td>12</td>
<td>42</td>
<td>5</td>
<td>77</td>
<td>101</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td>35</td>
<td>5</td>
<td>42</td>
<td>77</td>
<td>101</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>N-1</td>
<td>12</td>
<td>5</td>
<td>35</td>
<td>42</td>
<td>77</td>
<td>101</td>
</tr>
</tbody>
</table>

N - 1
Reducing the Number of Comparisons

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>77</td>
<td>42</td>
<td>35</td>
<td>12</td>
<td>101</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>42</td>
<td>35</td>
<td>12</td>
<td>77</td>
<td>5</td>
<td>101</td>
</tr>
<tr>
<td>0</td>
<td>35</td>
<td>12</td>
<td>42</td>
<td>5</td>
<td>77</td>
<td>101</td>
</tr>
<tr>
<td>0</td>
<td>12</td>
<td>35</td>
<td>5</td>
<td>42</td>
<td>77</td>
<td>101</td>
</tr>
<tr>
<td>0</td>
<td>12</td>
<td>5</td>
<td>35</td>
<td>42</td>
<td>77</td>
<td>101</td>
</tr>
</tbody>
</table>
Reducing the Number of Comparisons

• On the N\textsuperscript{th} “bubble up”, we only need to do MAX-N comparisons.

• For example:
  – This is the 4\textsuperscript{th} “bubble up”
  – MAX is 6 (total number of elements in array)
  – Thus we have 2 comparisons to do
Putting It All Together
public static void Bubblesort(int[] a) {
    int to_do = a.length - 1;

    while (to_do != 0) {
        for (int i = 0; i < to_do; i++) {
            if (a[i] > a[i + 1]) {
                int temp = a[i];
                a[i] = a[i + 1];
                a[i + 1] = temp;
            }
        }
        to_do = to_do - 1
    }
} //end method
Already Sorted Collections?

• What if the collection was already sorted?
• What if only a few elements were out of place and after a couple of “bubble ups,” the collection was sorted?

• We want to be able to detect this and “stop early”!

0  1  2  3  4  5
5 12 35 42 77 101
Using a Boolean “Flag”

• We can use a boolean variable to determine if any swapping occurred during the “bubble up.”

• If no swapping occurred, then we know that the collection is already sorted!

• This boolean “flag” needs to be reset after each “bubble up.”
public static void Bubblesort(int[] a){
    int to_do = a.length-1;
    boolean did_swap = true;

    while (to_do != 0 && did_swap) {
        did_swap = false;
        for(int i = 0; i < to_do; i++) {
            if(a[i] > a[i+1]) {
                did_swap = true;
                int temp = a[i];
                a[i] = a[i+1];
                a[i+1] = temp;
            }
        }
        to_do = to_do - 1
    } //end method
An Animated Example

did_swap

to_do

i

false

7

0

98 23 45 14 6 67 33 42

0 1 2 3 4 5 6 7
An Animated Example

did_swap

to_do

i

false

Swap

98 23 45 14 6 67 33 42

0 1 2 3 4 5 6 7
An Animated Example

did_swap: true

to_do:
  i: 1

Swap

23 98 45 14 6 67 33 42
0 1 2 3 4 5 6 7
An Animated Example

did_swap: true

to_do: 7

i: 1

```
23 98 45 14 6 67 33 42
```

0 1 2 3 4 5 6 7
An Animated Example

did_swap

to_do

i

true

7

1

Swap

23  98  45  14  6  67  33  42

0  1  2  3  4  5  6  7
An Animated Example

did_swap

true

to_do

7

i

1

Swap

23 45 98 14 6 67 33 42

0 1 2 3 4 5 6 7
An Animated Example

did_swap: true

to_do: 7

i: 2

Swap

23 45 98 14 6 67 33 42

0 1 2 3 4 5 6 7
An Animated Example

did_swap

to_do

i

true

2

Swap

23 45 14 98 6 67 33 42

0 1 2 3 4 5 6 7
An Animated Example

did_swap: true

to_do:
- i: 3

List:
- 23, 45, 14, 98, 6, 67, 33, 42
- 0, 1, 2, 3, 4, 5, 6, 7
An Animated Example

did_swap: true

to_do: 7

i: 3

Swap

23 45 14 98 6 67 33 42

0 1 2 3 4 5 6 7
An Animated Example

did_swap

to_do

i

true

7

3

Swap

23 45 14 6 98 67 33 42

0 1 2 3 4 5 6 7
An Animated Example

did_swap: true

to_do: 7

to_do: 4

i: 4

Array: [23, 45, 14, 6, 98, 67, 33, 42]

Index: [0, 1, 2, 3, 4, 5, 6, 7]
An Animated Example

did_swap: true

to_do: 7

i: 4

Swap

23 45 14 6 98 67 33 42

0 1 2 3 4 5 6 7
An Animated Example

did_swap

true

to_do

7

i

4

Swap

23 45 14 6 67 98 33 42

0 1 2 3 4 5 6 7
An Animated Example

did_swap

to_do

i

true

7

5

23 45 14 6 67 98 33 42

0 1 2 3 4 5 6 7
An Animated Example

did_swap

to_do

i

true

7

5

Swap

23 45 14 6 67 98 33 42

0 1 2 3 4 5 6 7
An Animated Example

did_swap: true

<table>
<thead>
<tr>
<th>i</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
</tr>
</tbody>
</table>

to_do:

<table>
<thead>
<tr>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>33</td>
</tr>
<tr>
<td>98</td>
</tr>
<tr>
<td>42</td>
</tr>
</tbody>
</table>

Swap
An Animated Example

did_swap

to_do

i

true

7

6

23 45 14 6 67 33 98 42

0 1 2 3 4 5 6 7
An Animated Example

did_swap: true

to_do: 7

i: 6

Swap

23 45 14 6 67 33 98 42

0 1 2 3 4 5 6 7
An Animated Example

did_swap: true

to_do:
   0 1 2 3 4 5 6 7

i:
   6

Swap:

23 45 14 6 67 33 42 98

0 1 2 3 4 5 6 7
After First Pass of Outer Loop

```
<table>
<thead>
<tr>
<th>did_swap</th>
<th>true</th>
</tr>
</thead>
<tbody>
<tr>
<td>to_do</td>
<td>7</td>
</tr>
<tr>
<td>i</td>
<td>7</td>
</tr>
</tbody>
</table>
```

Finished first “Bubble Up”

```
23 45 14 6 67 33 42 98
0 1 2 3 4 5 6 7
```
The Second “Bubble Up”

did_swap: false

to_do: 6

i: 0

23 45 14 6 67 33 42 98

0 1 2 3 4 5 6 7
The Second “Bubble Up”

did_swap | false
---|---
to_do | 6
i | 0

No Swap

23 45 14 6 67 33 42 98

0 1 2 3 4 5 6 7
The Second “Bubble Up”

did_swap: false

to_do: 6

i: 1

23 45 14 6 67 33 42 98

0 1 2 3 4 5 6 7
The Second “Bubble Up”

did_swap = false

to_do = 6

i = 1

Swap

23 45 14 6 67 33 42 98

0 1 2 3 4 5 6 7
The Second “Bubble Up”

did_swap: true

to_do: 6

i: 1

Swap

23 14 45 6 67 33 42 98

0 1 2 3 4 5 6 7
The Second “Bubble Up”

did_swap

to_do

i

true

6

2

23 14 45 6 67 33 42 98

0 1 2 3 4 5 6 7
The Second “Bubble Up”

- `did_swap`: `true`
- `to_do`: 6
- `i`: 2

Swap

23 14 45 6 67 33 42 98
0 1 2 3 4 5 6 7
The Second “Bubble Up”

did_swap: true

i: 2

to_do:
  6

Swap

23 14 6 45 67 33 42 98
0 1 2 3 4 5 6 7
The Second “Bubble Up”

did_swap: true

to_do: 6

i: 3

0 1 2 3 4 5 6 7

23 14 6 45 67 33 42 98
The Second “Bubble Up”

did_swap = true

to_do = 6

i = 3

No Swap

23 14 6 45 67 33 42 98

0 1 2 3 4 5 6 7
The Second “Bubble Up”

did_swap

to_do

i

true

6

4

23 14 6 45 67 33 42 98

0 1 2 3 4 5 6 7

did_swap = true

to_do = 6

i = 4
The Second “Bubble Up”

did_swap: true

to_do:
  i: 6

i: 4

Swap

23 14 6 45 67 33 42 98

0 1 2 3 4 5 6 7
The Second “Bubble Up”

did_swap: true

to_do: 6

i: 4

Swap

23 14 6 45 33 67 42 98

0 1 2 3 4 5 6 7
The Second “Bubble Up”

did_swap: true  
to_do: 6  
i: 5

Array:

23 14 6 45 33 67 42 98
The Second “Bubble Up”

did_swap: true

to_do: 6

i: 5

Swap:

23 14 6 45 33 67 42 98

0 1 2 3 4 5 6 7
The Second “Bubble Up”

did_swap: true

to_do: 6

i: 5

Swap

23 14 6 45 33 42 67 98

0 1 2 3 4 5 6 7
After Second Pass of Outer Loop

did_swap

true

to_do

6

i

6

Finished second “Bubble Up”

23 14 6 45 33 42 67 98

0 1 2 3 4 5 6 7
The Third “Bubble Up”

did_swap: false

i: 0
to_do: 5

23 14 6 45 33 42 67 98
The Third “Bubble Up”
The Third “Bubble Up”

did_swap

to_do

i

true

5

0

Swap

14 23 6 45 33 42 67 98

0 1 2 3 4 5 6 7
The Third “Bubble Up”

did_swap: true

to_do: 5

i: 1

0 1 2 3 4 5 6 7

14 23 6 45 33 42 67 98
The Third “Bubble Up”

did_swap
  true

  to_do
    5

  i
    1

Swap

14 23 6 45 33 42 67 98
0 1 2 3 4 5 6 7
The Third “Bubble Up”

did_swap
  true

to_do
  5
  1

i
  1

Swap

14  6  23  45  33  42  67  98
0    1    2    3    4    5    6    7
The Third “Bubble Up”

did_swap: true

to_do: 5

i: 2

14 6 23 45 33 42 67 98

0 1 2 3 4 5 6 7
The Third “Bubble Up”

```
<table>
<thead>
<tr>
<th>did_swap</th>
<th>true</th>
</tr>
</thead>
<tbody>
<tr>
<td>to_do</td>
<td>5</td>
</tr>
<tr>
<td>i</td>
<td>2</td>
</tr>
</tbody>
</table>
```

No Swap

```
0 1 2 3 4 5 6 7
14 6 23 45 33 42 67 98
```
The Third “Bubble Up”

did_swap

to_do

i

true

5

3

14 6 23 45 33 42 67 98

0 1 2 3 4 5 6 7
The Third “Bubble Up”

did_swap

true

5

3

Swap

0 1 2 3 4 5 6 7

14 6 23 45 33 42 67 98
The Third “Bubble Up”

did_swap: true

to_do: 5

i: 3

Swap

14 6 23 33 45 42 67 98
0 1 2 3 4 5 6 7
The Third “Bubble Up”

did_swap = true

to_do = 5

\[
\begin{array}{c}
\text{i} \\
4
\end{array}
\]

\[
\begin{array}{cccccccc}
14 & 6 & 23 & 33 & 45 & 42 & 67 & 98 \\
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7
\end{array}
\]
The Third “Bubble Up”

did_swap: true

to_do: 5

\[ i = 4 \]

Swap

\[ 14 \quad 6 \quad 23 \quad 33 \quad 45 \quad 42 \quad 67 \quad 98 \]

\[ 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \]
The Third “Bubble Up”

did_swap: true

to_do: 5

i: 4

Swap

14 6 23 33 42 45 67 98

0 1 2 3 4 5 6 7
After Third Pass of Outer Loop

Finished third “Bubble Up”
The Fourth “Bubble Up”

did_swap

false

to_do

4

i

0

14 6 23 33 42 45 67 98

0 1 2 3 4 5 6 7
The Fourth “Bubble Up”

did_swap: false

to_do: 4

i: 0

Swap

14 6 23 33 42 45 67 98

0 1 2 3 4 5 6 7
The Fourth “Bubble Up”

did_swap: true

to_do: 4

i: 0

Swap

6 14 23 33 42 45 67 98

0 1 2 3 4 5 6 7
The Fourth “Bubble Up”

did_swap: true

to_do: 4

i: 1

6 14 23 33 42 45 67 98

0 1 2 3 4 5 6 7
The Fourth “Bubble Up”

did_swap: true

No Swap

<table>
<thead>
<tr>
<th>i</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>to_do</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>6</th>
<th>14</th>
<th>23</th>
<th>33</th>
<th>42</th>
<th>45</th>
<th>67</th>
<th>98</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>
The Fourth “Bubble Up”

did_swap: true

to_do: 4

i: 2

6 14 23 33 42 45 67 98

0 1 2 3 4 5 6 7
The Fourth “Bubble Up”

did_swap: true

to_do: 4

i: 2

No Swap

6 14 23 33 42 45 67 98
The Fourth “Bubble Up”

did_swap = true

to_do = 4

i = 3

6  14  23  33  42  45  67  98

0  1  2  3  4  5  6  7
The Fourth “Bubble Up”

did_swap: true

to_do: 4

i: 3

No Swap

6 14 23 33 42 45 67 98
After Fourth Pass of Outer Loop

Finished fourth “Bubble Up”
The Fifth “Bubble Up”

did_swap: false

to_do: 3

i: 0

<table>
<thead>
<tr>
<th>6</th>
<th>14</th>
<th>23</th>
<th>33</th>
<th>42</th>
<th>45</th>
<th>67</th>
<th>98</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>
The Fifth “Bubble Up”

did_swap: false

to_do: 3

i: 0

No Swap

6  14  23  33  42  45  67  98

0  1  2  3  4  5  6  7
The Fifth “Bubble Up”

did_swap
  to_do
    i

false
  3
    1

0 1 2 3 4 5 6 7

6 14 23 33 42 45 67 98
The Fifth “Bubble Up”

did_swap

false

to_do

3

i

1

No Swap

6  14  23  33  42  45  67  98

0  1  2  3  4  5  6  7
The Fifth “Bubble Up”

did_swap: false

to_do: 3

i: 2

0 1 2 3 4 5 6 7

6 14 23 33 42 45 67 98
The Fifth “Bubble Up”

did_swap: false

to_do: 3

i: 2

No Swap

6 14 23 33 42 45 67 98

0 1 2 3 4 5 6 7
After Fifth Pass of Outer Loop

```plaintext
did_swap = false
to_do = 3
i = 3
```

Finished fifth “Bubble Up”
Finished “Early”

We didn’t do any swapping, so all of the other elements must be correctly placed.

We can “skip” the last two passes of the outer loop.
Summary

• “Bubble Up” algorithm will move largest value to its correct location (to the right/end of array)

• Repeat “Bubble Up” until all elements are correctly placed:
  – Maximum of N-1 times
  – Can finish early if no swapping occurs

• We reduce the number of elements we compare each time one is correctly placed
Truth in CS Act

• NOBODY EVER USES BUBBLE SORT

• NOBODY

• NOT EVER

• BECAUSE IT IS EXTREMELY INEFFICIENT
Questions?