Merge Sort

- Problem:
  Sort an array quickly

- $O(n^2)$ sorts are easy to write, but slow

- Merge sort is $O(n \log n)$!
Merge Sort is Recursive

128 values

Running Merge Sort on the entire array...

Stack
mergesort(0,128)
Merge Sort is Recursive

Split the array in half.

(No need to move anything around, yet.)

Stack
mergesort(0,128)
Merge Sort is Recursive

<table>
<thead>
<tr>
<th>64 values</th>
<th>64 values</th>
</tr>
</thead>
</table>

Recurse into one half.

Recursion will probably go very deep (many steps).

When it returns, recurse into the other.

Stack
mergesort(0,128)
mergesort(0,64)
Merge Sort is Recursive

Recursion in Merge Sort:

- 64 values
- Recurse into one half.

Stack:
- mergesort(0,128)
- mergesort(0,64)
- mergesort(0,32)
Merge Sort is Recursive

Recurse into one half.

Stack
mergesort(0,128)
mergesort(0,64)
mergesort(0,32)
mergesort(0,16)
Merge Sort is Recursive

<table>
<thead>
<tr>
<th>8</th>
<th>8</th>
<th>16</th>
<th>32 values</th>
<th>64 values</th>
</tr>
</thead>
</table>

Recurse into one half.

- We usually end up going pretty deep.
- When the block sizes are small, sort with something simple (often: Insertion Sort)

Stack
- mergesort(0,128)
- mergesort(0,64)
- mergesort(0,32)
- mergesort(0,16)
- mergesort(0,8)
Merge Sort is Recursive

When we return from recursion, recurse into the right-hand half...

Stack
mergesort(0,128)
mergesort(0,64)
mergesort(0,32)
mergesort(0,16)
mergesort(8,8)
When two blocks are sorted, we use a **merge** operation to join them.

**Usually**: Only merge same-size blocks! Other merges will hurt performance!
The Merge Step
More and more blocks build up – since we only merge duplicate sized blocks.
Merge Sort

Sometimes, we will merge many times in a row.

(As the last step, we always merge everything, smallest to largest, even if things aren't nice powers of 2.)
Merge Sort

Sometimes, we will merge many times in a row.

(As the last step, we always merge everything, smallest to largest, even if things aren't nice powers of 2.)
Merge Sort

32  32

Sometimes, we will merge many times in a row.

(As the last step, we always merge everything, smallest to largest, even if things aren't nice powers of 2.)
Merge Sort

Sometimes, we will merge many times in a row.

(As the last step, we always merge everything, smallest to largest, even if things aren't nice powers of 2.)
The Merge Algorithm

Until all values are copied:
compare $\text{first(left)}, \text{first(right)}$
copy smaller value into destination

• Requires a destination buffer
  – Can't copy values in-place
  – Same size as **merge output**
  – For the last merge, has to be the same size as the input!
Merge Sort

13 17 -10 0 13 100 24 13 11 12 17 10

merge()
Merge Sort

merge()
Merge Sort

merge()

-10 0
Merge Sort

merge()
Merge Sort

merge()

-10 0 13 17

13 100 24 13 11 12 17 10
Merge Sort

-10  0  13  17  13  100  24  13  11  12  17  10

Copy
Back

-10  0  13  17
Merge Sort

-10  0  13  17  13  100  13  24  11  12  17  10

merge()
Merge Sort

-10 0 13 17 100 13 24 11 12 17 10

merge()
Merge Sort

```
|   -10 |    0 |   13 |   17 |   100 |   24 |   11 |   12 |   17 |   10 |
```

merge()
Merge Sort

-10 0 13 17 **100** 13 17 10

merge()

13 13 **24**
Merge Sort

merge()
Merge Sort

-10 0 13 17 13 13 24 100 11 12 17 10

Copy
Back

13 13 24 100
## Merge Sort

<table>
<thead>
<tr>
<th>-10</th>
<th>0</th>
<th>13</th>
<th>17</th>
<th>13</th>
<th>13</th>
<th>24</th>
<th>100</th>
<th>11</th>
<th>12</th>
<th>17</th>
<th>10</th>
</tr>
</thead>
</table>

**merge()**
Merge Sort

merge()

-10
Merge Sort

merge()

13 17 13 13 24 100 11 12 17 10

-10 0
Merge Sort

```
-10  0  13
```

```
17  13  13  24  100  11  12  17  10
```
Merge Sort

merge()
Why a 2\textsuperscript{nd} Buffer?

- To merge in place, we have to shift values around
  - Each chosen value would require $\Theta(n)$ shifts.
  - Merging two lists would be $\Theta(n^2)$!

- Instead, we use a 2\textsuperscript{nd} buffer, and copy back
  - $\Theta(n)$ time per pass
  - But also $\Theta(n)$ space!
Why Merge Sort?

- $O(n^2)$ sorts are easy to write, but **slow**
  - Insertion Sort, Bubble Sort

- Merge sort is $O(n \ lg \ n)$!
\[ O(n \ lg \ n) \]

<table>
<thead>
<tr>
<th>( n )</th>
<th>( n^2 )</th>
<th>( n \ lg \ n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>32</td>
</tr>
<tr>
<td>100</td>
<td>( 10^4 )</td>
<td>650</td>
</tr>
<tr>
<td>1000</td>
<td>( 10^6 )</td>
<td>( 10^4 )</td>
</tr>
<tr>
<td>10000000</td>
<td>( 10^{12} )</td>
<td>( 2 \times 10^7 )</td>
</tr>
</tbody>
</table>