There are many algorithms for sorting

- **Insertion Sort** (to be discussed today)
- **Bubble Sort** (read Insight §8.2)
- **Merge Sort** (to be discussed Thursday)
- **Quick Sort** (a variant used by Matlab's built-in `sort` function)

Each has advantages and disadvantages. Some algorithms are faster (time-efficient) while others are memory-efficient.

Great opportunity for learning how to analyze programs and algorithms!

---

### Sorting data allows us to search more easily

```plaintext
<table>
<thead>
<tr>
<th>Name</th>
<th>Time</th>
<th>State</th>
<th>Country</th>
<th>Ctz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chao, Hui</td>
<td>2:25:25</td>
<td>ETH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vivian, Maria</td>
<td>2:25:30</td>
<td>KB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Siti, Ayu</td>
<td>2:25:13</td>
<td>LAT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zelina, Fionn</td>
<td>2:25:00</td>
<td>ETH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ella, Erica</td>
<td>2:25:10</td>
<td>ITA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lily, Alice</td>
<td>2:25:27</td>
<td>RUS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Laura, Diana</td>
<td>2:25:32</td>
<td>CAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Emily, Olivia</td>
<td>2:25:45</td>
<td>USA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thea, Amelia</td>
<td>2:25:50</td>
<td>CAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Grace, Tracy</td>
<td>2:26:00</td>
<td>CAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Emma, Karen</td>
<td>2:26:10</td>
<td>USA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sarah, Ukraine</td>
<td>2:26:20</td>
<td>CAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Anna, Elena</td>
<td>2:26:30</td>
<td>USA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olivia, Diana</td>
<td>2:26:40</td>
<td>USA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Amanda, Raquel</td>
<td>2:26:50</td>
<td>CAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maria, Beatriz</td>
<td>2:27:00</td>
<td>USA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sofia, Clara</td>
<td>2:27:10</td>
<td>CAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Emily, Sofia</td>
<td>2:27:20</td>
<td>USA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Katrina, Rachel</td>
<td>2:27:30</td>
<td>CAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Campo, Alexandra</td>
<td>2:27:40</td>
<td>USA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jessica, Ivan</td>
<td>2:28:00</td>
<td>CAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Claire, Maria</td>
<td>2:28:10</td>
<td>USA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nia, Sophia</td>
<td>2:28:20</td>
<td>CAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Anna, Sofia</td>
<td>2:28:30</td>
<td>USA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lila, Elena</td>
<td>2:28:40</td>
<td>CAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Felicia, Maria</td>
<td>2:29:00</td>
<td>USA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Magdalena, Ana</td>
<td>2:29:10</td>
<td>CAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Samantha, Raquel</td>
<td>2:29:20</td>
<td>USA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cristina, Ana</td>
<td>2:29:30</td>
<td>CAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Leonor, Maria</td>
<td>2:29:40</td>
<td>USA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Valeria, Sofia</td>
<td>2:30:00</td>
<td>CAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Carolina, Ana</td>
<td>2:30:10</td>
<td>USA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sofia, Emma</td>
<td>2:30:20</td>
<td>CAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Elisa, Sofia</td>
<td>2:30:30</td>
<td>USA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maria, Sofia</td>
<td>2:30:40</td>
<td>CAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Beatriz, Sofia</td>
<td>2:30:50</td>
<td>USA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sofia, Carolina</td>
<td>2:31:00</td>
<td>CAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Anna, Sofia</td>
<td>2:31:10</td>
<td>USA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sofia, Marta</td>
<td>2:31:20</td>
<td>CAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sofia, Carolina</td>
<td>2:31:30</td>
<td>USA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sofia, Sofia</td>
<td>2:31:40</td>
<td>CAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sofia, Anna</td>
<td>2:31:50</td>
<td>USA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sofia, Maria</td>
<td>2:32:00</td>
<td>CAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sofia, Sofia</td>
<td>2:32:10</td>
<td>USA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sofia, Anna</td>
<td>2:32:20</td>
<td>CAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sofia, Sofia</td>
<td>2:32:30</td>
<td>USA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sofia, Sofia</td>
<td>2:32:40</td>
<td>CAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sofia, Sofia</td>
<td>2:32:50</td>
<td>USA</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

---

### The Insertion Process

**Given a sorted array x, insert a number y such that the result is sorted**

```plaintext
2 3 6 9
+---
| 8 |

sorted
```

---

**Insertion**

**one insert process**

```plaintext
2 3 6 9 8
```

```plaintext
2 3 6 8 9
```

Just swap 8 & 9

---

**Insertion**

**one insert process**

```plaintext
2 3 6 9 8
```

```plaintext
2 3 6 8 9
```

Insert 8 into the sorted segment

---

**Insertion**

**one insert process**

```plaintext
2 3 6 9 8
```

```plaintext
2 3 6 8 9
```

Insert 9 into the sorted segment

---

**Insertion**

**one insert process**

```plaintext
2 3 6 9 8
```

```plaintext
2 3 6 8 9
```

Just swap 8 & 9

---

**Insertion**

**one insert process**

```plaintext
2 3 6 9 8
```

```plaintext
2 3 6 8 9
```

Insert 4 into the sorted segment

---

**Insertion**

**one insert process**

```plaintext
2 3 6 9 8
```

```plaintext
2 3 6 8 9
```

Just swap 8 & 9
Insertion Sort vs. Bubble Sort

- Read about Bubble Sort in Insight §8.2
- Both algorithms involve the repeated comparison of adjacent values and swaps
- Find out which algorithm is more efficient on average
Other efficiency considerations

- Worst case, best case, average case
- Use of subfunction incurs an “overhead”
- Memory use and access

- Example: Rather than directing the `insert` process to a subfunction, have it done “in-line.”
- Also, Insertion sort can be done “in-place,” i.e., using “only” the memory space of the original vector.

```matlab
function x = InsertionSortInplace(x)
% Sort vector x in ascending order with insertion sort
n = length(x);
for i= 1:n-1
    % Sort x(1:i+1) given that x(1:i) is sorted
end
```

Sort an array of objects

- Given `x`, a 1-d array of `Interval` references, sort `x` according to the widths of the `Intervals` from narrowest to widest
- Use the insertion sort algorithm
- How much of our code needs to be changed?

  - A. No change
  - B. One or two statements
  - C. About half the code
  - D. Most of the code

Searching for an item in an unorganized collection?

- May need to look through the whole collection to find the target item
- E.g., find value `x` in vector `v`

```
% Linear Search
% f is index of first occurrence
% of value x in vector v.
% f is -1 if x not found.
k= 1;
while  k<=length(v) && v(k)~=x
    k= k + 1;
end
if  k>length(v)
    f= -1; % signal for x not found
else
    f= k;
end
```

Suppose another vector is twice as long as `v`. The expected “effort” required to do a linear search is ...

```
% Linear Search
% f is index of first occurrence
% of value x in vector v.
% f is -1 if x not found.
k= 1;
while  k<=length(v) && v(k)~=x
    k= k + 1;
end
if  k>length(v)
    f= -1; % signal for x not found
else
    f= k;
end
```
% Linear Search
% f is index of first occurrence of value x in vector v.
% f is -1 if x not found.
k= 1;
while  k<=length(v) && v(k)~=x
    k= k + 1;
end
if  k>length(v)
    f= -1; % signal for x not found
else
    f= k;
end

An ordered (sorted) list

The Manhattan phone book has 1,000,000+ entries.

How is it possible to locate a name by examining just a tiny, tiny fraction of those entries?

Key idea of “phone book search”: repeated halving

To find the page containing Pat Reed’s number...

    while  (Phone book is longer than 1 page)
        Open to the middle page.
        if  “Reed” comes before the first entry,
            Rip and throw away the 2nd half.
        else
            Rip and throw away the 1st half.
        end
    end

What happens to the phone book length?

Original: 3000 pages
After 1 rip: 1500 pages
After 2 rips: 750 pages
After 3 rips: 375 pages
After 4 rips: 188 pages
After 5 rips: 94 pages
:  
After 12 rips: 1 page

Binary Search

Repeatedly halving the size of the “search space” is the main idea behind the method of binary search.

An item in a sorted array of length n can be located with just \( \log_2 n \) comparisons.

\( \% \) Linear Search
\( \% \) f is index of first occurrence of value x in vector v.
\( \% \) f is -1 if x not found.
k= 1;
while  k<=length(v) && v(k)~=x
    k= k + 1;
end
if  k>length(v)
    f= -1; % signal for x not found
else
    f= k;
end

Searching in a sorted list should require less work.
**Binary Search**

Repeatedly halving the size of the “search space” is the main idea behind the method of binary search.

An item in a sorted array of length n can be located with just \( \log_2 n \) comparisons.

“Savings” is significant!

<table>
<thead>
<tr>
<th>n</th>
<th>( \log_2(n) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>7</td>
</tr>
<tr>
<td>1000</td>
<td>10</td>
</tr>
<tr>
<td>10000</td>
<td>13</td>
</tr>
</tbody>
</table>

**Binary search: target x = 70**

1  2  3  4  5  6  7  8  9  10  11  12

\[ \begin{array}{cccccccccccc}
12 & 15 & 33 & 35 & 42 & 45 & 51 & 62 & 73 & 75 & 86 & 98 \\
\end{array} \]

\[ v \]

L: 1  \[ v(Mid) \leq x \]

Mid: 6  So throw away the left half...

R: 12

L: 6  \[ v(Mid) \leq x \]

Mid: 7  So throw away the left half...

R: 9

L: 7  \[ v(Mid) \leq x \]

Mid: 8  So throw away the left half...

R: 9

L: 8  Done because R-L = 1

Mid: 8

R: 9
function L = binarySearch(x, v)
% Find position after which to insert x. v(1)<..<v(end).
% L is the index such that v(L) <= x < v(L+1);
% L=0 if x<v(1).  If x>v(end), L=length(v) but x~v(L).
% Maintain a search window [L..R] such that v(L)<=x<v(R).
% Since x may not be in v, initially set ... 
L=0;  R=length(v)+1;

% Keep halving [L..R] until R-L is 1, 
% always keeping v(L) <= x < v(R)
while  R ~= L+1
  m= floor((L+R)/2);  % middle of search window
  if  v(m) <= x
    L= m;
  else
    R= m;
  end
end
end

This version is different
from that in Insight!

function L = binarySearch(x, v)
% Find position after which to insert x. v(1)<..<v(end).
% L is the index such that v(L) <= x < v(L+1);
% L=0 if x<v(1).  If x>v(end), L=length(v) but x~v(L).
% Maintain a search window [L..R] such that v(L)<=x<v(R).
% Since x may not be in v, initially set ... 
L=0;  R=length(v)+1;

% Keep halving [L..R] until R-L is 1, 
% always keeping v(L) <= x < v(R)
while  R ~= L+1
  m= floor((L+R)/2);  % middle of search window
  if  v(m) <= x
    L= m;
  else
    R= m;
  end
end

Play with showBinarySearch.m