Previous Lecture:
- OOP: Overriding methods
- Introduction to recursion

Today’s Lecture:
- Recursion examples: remove all occurrences of a character in a string, a mesh of triangles
- Sort algorithm: Insertion Sort
- See Insight §8.2 for the Bubble Sort algorithm

Announcements:
- Last call to notify us of final exam conflict
- Project 6 parts A and B due 5/2, Thurs, at 11pm

Recursion
- The Fibonacci sequence is defined recursively: 
  \[ F(1)=1, \ F(2)=1, \ F(3)= F(1) + F(2) = 2 \]
  \[ F(4)= F(2) + F(3) = 3 \]
- It is defined in terms of itself; its definition invokes itself.

- Algorithms, and functions, can be recursive as well.
  I.e., a function can call itself.
- Example: remove all occurrences of a character from a string
  'gc aatc gga c '  \(\rightarrow\)  'gcaatcggac'

Example: removing all occurrences of a character
  - Can solve using recursion
    - Original problem: remove all the blanks in string s
    - Decompose into two parts: 1. remove blank in s(1)
      2. remove blanks in s(2:length(s))

```matlab
function s = removeChar(c, s)
    if length(s)==0  % Base case: nothing to do
        return
    else
        if s(1)~=c
            s = [s(1) removeChar(c, s(2:length(s)));
        else
            s = removeChar(c, s(2:length(s)));
        end
    end
end
```

```
function s = removeChar(c, s)
if length(s)==0    % Base case: nothing to do
    return
else
    if s(1)==c
        s = [s(1) removeChar(c, s(2:length(s)));
    else
        s = removeChar(c, s(2:length(s)));
    end
end
```

Example: removing all occurrences of a character from a string
  'gc aatc gga c '  \(\rightarrow\)  'gcaatcggac'
Divide-and-conquer methods, such as recursion, is useful in geometric situations

Chop a region up into triangles with smaller triangles in "areas of interest"

Recursive mesh generation

Why is mesh generation a divide-&-conquer process?
Let's draw this graphic
The “level-2” partition of the triangle

The “level-3” partition of the triangle

The “level-4” partition of the triangle

The basic operation at each level

if the triangle is small
Don’t subdivide and just color it yellow.
else
Subdivide:
Connect the side midpoints;
color the interior triangle magenta;
apply same process to each outer triangle.
end

function MeshTriangle(x,y,L)
% x,y are 3-vectors that define the vertices of a triangle.
% Draw level-L partitioning. Assume hold is on.
if L==0
% Recursion limit reached; no more subdivision required.
fill(x,y,'y')  % Color this triangle yellow
else
% Need to subdivide: determine the side midpoints; connect
% midpts to get "interior triangle": color it magenta.
% Apply the process to the three “corner” triangles...
end

Key to recursion

- Must identify (at least) one base case, the “trivially simple” case
- No recursion is done in this case
- The recursive case(s) must reflect progress towards the base case
  - E.g., give a shorter vector as the argument to the recursive call – see removeChar
  - E.g., ask for a lower level of subdivision in the recursive call – see MeshTriangle

Lecture slides
Sorting data allows us to search more easily

<table>
<thead>
<tr>
<th>Rank</th>
<th>Athlete</th>
<th>Splits</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NED Ranomi Kromowidjojo</td>
<td>25.76, 0.77, 27.24</td>
<td>53.00</td>
</tr>
<tr>
<td>2</td>
<td>BLR Aliaksandra Herasimenia</td>
<td>25.22, 0.74, 28.16</td>
<td>53.38</td>
</tr>
<tr>
<td>3</td>
<td>CHN Tang Yi</td>
<td>25.95, 0.71, 27.49</td>
<td>53.44</td>
</tr>
<tr>
<td>4</td>
<td>AUS Melanie Schlanger</td>
<td>26.09, 0.73, 27.38</td>
<td>53.47</td>
</tr>
<tr>
<td>5</td>
<td>USA Missy Franklin</td>
<td>26.22, 0.78, 27.42</td>
<td>53.64</td>
</tr>
<tr>
<td>6</td>
<td>GBR Francesca Halsall</td>
<td>25.78, 0.70, 27.88</td>
<td>53.66</td>
</tr>
<tr>
<td>7</td>
<td>DEN Jeanette Ottesen Gray</td>
<td>25.50, 0.70, 28.25</td>
<td>53.75</td>
</tr>
<tr>
<td>8</td>
<td>USA Jessica Hardy</td>
<td>25.69, 0.72, 28.33</td>
<td>54.02</td>
</tr>
</tbody>
</table>

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!

The Insertion Process
- Given a sorted array $x$, insert a number $y$ such that the result is sorted

$$\begin{array}{cccc}
2 & 3 & 6 & 9 \\
\uparrow & \uparrow & \uparrow & \uparrow \\
& & & y \\
2 & 3 & 6 & 8 & 9
\end{array}$$

Sort vector $x$ using the Insertion Sort algorithm

Need to start with a sorted subvector. How do you find one?

- Length 1 subvector is “sorted”
- $\text{Insert } x(2) = \text{Insert}(x(1:2))$
- $\text{Insert } x(3) = \text{Insert}(x(1:3))$
- $\text{Insert } x(4) = \text{Insert}(x(1:4))$
- $\text{Insert } x(5) = \text{Insert}(x(1:5))$
- $\text{Insert } x(6) = \text{Insert}(x(1:6))$

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