Previous Lecture:
- OOP: Overriding methods
- Recursion example: Remove all occurrences of a character in a string

Today’s Lecture:
- Recursion example: A mesh of triangles
- Sort algorithm: Insertion Sort
- Efficiency Analysis
- See Insight §8.2 for the Bubble Sort algorithm

Announcements:
- Project 6 due Tue 11pm. Remember academic integrity!
- Regular office/consulting hours end Wednesday. Study break hours will be posted next week

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

Start with a triangle

A “level-1” partition of the triangle
(obtained by connecting the midpoints of the sides of the original triangle)

The “level-2” partition of the triangle

Now do the same partitioning (connecting midpts) on each corner (white) triangle to obtain the “level-2” partitioning
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
    a = [(x(1)+x(2))/2 (x(2)+x(3))/2 (x(3)+x(1))/2];
    b = [(y(1)+y(2))/2 (y(2)+y(3))/2 (y(3)+y(1))/2];
    fill(a,b,'m')
    \% Apply the process to the three “corner” triangles...
  end
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

Searching for an item in a collection

Is the collection organized?
What is the organizing scheme?

Sorting data allows us to search more easily

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
    a = [(x(1)+x(2))/2 (x(2)+x(3))/2 (x(3)+x(1))/2];
    b = [(y(1)+y(2))/2 (y(2)+y(3))/2 (y(3)+y(1))/2];
    fill(a,b,'m')
    % Apply the process to the three “corner” triangles...
  end

There are many algorithms for sorting

- Insertion Sort (to be discussed today)
- Bubble Sort (read Insight §8.2)
- Merge Sort (to be discussed next lecture)
- 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.

The problem of inserting an element into a sorted array is a common problem in computer science. The process involves finding the correct position of the item tobe inserted and then shifting all the elements that come after the insertion point to make space for the new item. This process is repeated for each element that needs to be inserted. The resulting array is sorted.

The process can be illustrated as follows:

1. Insert 8 into the sorted segment: Just swap 8 & 9.
2. Insert 4 into the sorted segment: Insert 4 & 9.
4. Compare adjacent components: done! No more swaps.

See Insert.m for the insert process.
Sort vector $x$ using the Insertion Sort algorithm

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

$x$

- Length 1 subvector is "sorted"
- $\text{Insert } x(2): [x(1:2), C, S] = \text{Insert}(x(1:2))$
- $\text{Insert } x(3): [x(1:3), C, S] = \text{Insert}(x(1:3))$
- $\text{Insert } x(4): [x(1:4), C, S] = \text{Insert}(x(1:4))$
- $\text{Insert } x(5): [x(1:5), C, S] = \text{Insert}(x(1:5))$
- $\text{Insert } x(6): [x(1:6), C, S] = \text{Insert}(x(1:6))$

`InsertionSort.m`

How much “work” is insertion sort?

- In the worst case, make $k$ comparisons to insert an element in a sorted array of $k$ elements. For an array of length $N$:

$$\text{______________} \text{for big } N$$

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.

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 statement
C. About half the code
D. Most of the code

$\text{function } x = \text{InsertionSortInplace}(x)$

% Sort vector $x$ in ascending order with insertion sort

$n = \text{length}(x);$  
for $i = 1:n-1$

% Sort $x(1:i+1)$ given that $x(1:i)$ is sorted

$j = i;$  
while

% swap $x(j+1)$ and $x(j)$

$j = j-1;$
end