- Previous Lecture:
  - Acoustic data: frequency computation
  - Touchtone phone
- Today's Lecture:
  - "Divide and conquer" strategies
    - Binary search
    - Merge sort
    - Recursion

```
% 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
```

A. squared

B. doubled

C. the same

D. halved

Suppose another vector is twice as long as v. The expected "effort" required to do a linear search is …

Key idea: 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
```


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.


```
function L = binSearch(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

    else

    end
end
```

What happens if the values in the sorted vector are not unique? Say, the target value is in the vector and that value appears in the vector multiple times…

A. The first occurrence is identified

B. The last occurrence is identified

C. Any one of the occurrences may be identified

D. Binary search doesn't work

Merge sort: Motivation

If I have two helpers, I'd…

- Give each helper half the array to sort
- Then I get back the sorted subarrays and merge them.

What if those two helpers each had two sub-helpers?

And the sub-helpers each had two sub-sub-helpers? And…

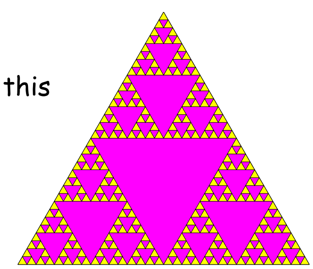
```
function y = mergeSort(x)
% x is a vector.  y is a vector
% consisting of the values in x
% sorted from smallest to largest.

n = length(x);
if n==1
      y = x;
else
      m  = floor(n/2);
      y1 = mergeSort(x(1:m));
      y2 = mergeSort(x(m+1:n));
      y  = merge(y1,y2);
end
```



```
function z = merge(x,y)
n = length(x); m = length(y);
z = zeros(1,n+m);
ix = 1; iy = 1;
for iz=1:(n+m)
    if  ix > n
        z(iz)=
    elseif  iy>m
        z(iz)=
    elseif  x(ix) <= y(iy)
        z(iz)=
    else
        z(iz)=
    end
end
```

Why is mesh generation a divide & conquer process?

Let's draw this graphic



The basic operation

if *the triangle is big enough*

Connect the midpoints.

Color the interior triangle mauve.

else

Color the whole triangle yellow.

end


```
function drawTriangle(x,y,level)
% Draw recursively colored triangles.
% x,y are 3-vectors that define the vertices of a triangle.

if level==5
  % Recursion limit (depth) reached
    fill(x,y,'y')  % Color whole triangle yellow
else
  % Draw the triangle...
    plot([x x(1)],[y y(1)],'k')
  % Determine the midpoints...
    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];
  % Draw and color the interior triangle mauve
    pause
    fill(a,b,'m')
    pause
  % Apply the process to the three "corner" triangles...
    drawTriangle([x(1) a(1) a(3)],[y(1) b(1) b(3)],level+1)
    drawTriangle([x(2) a(2) a(1)],[y(2) b(2) b(1)],level+1)
    drawTriangle([x(3) a(3) a(2)],[y(3) b(3) b(2)],level+1)
end
```