- Previous Lecture:
  - Linear Search
  - Bubble Sort, Insertion Sort
- Today's Lecture:
  - "Divide and conquer" strategies
    - Binary search
    - Merge sort
    - Recursion
- Announcements:
  - Discussion this week in classrooms
  - Prelim 3 will be returned at end of lecture. If your paper isn't here, pick it up from CS1112 consultants in ACCEL during consulting hrs (Sunday to Thursdays 5-10pm)
  - Project 6 due May 5th. Part 1 posted; Part 2 to be posted later.

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.

"Savings" is significant!

n	log2(n)
100	7
1000	10
10000	13

Binary search: target x = 70

	1	2	3	4	5	6	7	8	9	10	11	12
v	12	15	33	35	42	45	51	62	73	75	86	98

L: 1

Mid: 6

R: 12

v(Mid) <= x

So throw away the left half…

Binary search: target x = 70

	1	2	3	4	5	6	7	8	9	10	11	12
v	12	15	33	35	42	45	51	62	73	75	86	98

L: 6

Mid: 9

R: 12

`x < v(Mid)`

So throw away the right half…

Binary search: target x = 70

	1	2	3	4	5	6	7	8	9	10	11	12
v	12	15	33	35	42	45	51	62	73	75	86	98

L: 6

Mid: 7

R: 9

`v(Mid) <= x`

So throw away the left half…

Binary search: target x = 70

	1	2	3	4	5	6	7	8	9	10	11	12
v	12	15	33	35	42	45	51	62	73	75	86	98

L: 7

Mid: 8

R: 9

`v(Mid) <= x`

So throw away the left half…

Binary search: target x = 70

	1	2	3	4	5	6	7	8	9	10	11	12
v	12	15	33	35	42	45	51	62	73	75	86	98

L: 8

Mid: 8

R: 9

Done because
`R-L = 1`


```
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


    else


    end
end
```

Binary search is efficient, but how do we sort a vector in the first place so that we can use binary search?

- Many different algorithms out there…
- We saw bubble sort and insertion sort
- Let's look at **merge sort**
- An example of the "divide and conquer" approach
- We'll compare their efficiency later

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…

Subdivide the sorting task

H	E	M	G	B	K	A	Q	F	L	P	D	R	C	J	N

H	E	M	G	B	K	A	Q

F	L	P	D	R	C	J	N


```
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);
      yL = mergeSortL(x(1:m));
      yR = mergeSortR(x(m+1:n));
      y  = merge(yL,yR);
end
```


The central sub-problem is the merging of two sorted arrays into one single sorted array

12	33	35	45

15	42	55	65	75

12	15	33	35	42	45	55	65	75

Merge

x: | 12 | 33 | 35 | 45 | ix: 1

y: | 15 | 42 | 55 | 65 | 75 | iy: 1

z: | | | | | | | | | | iz: 1

ix<=4 and iy<=5: x(ix) <= y(iy) ???

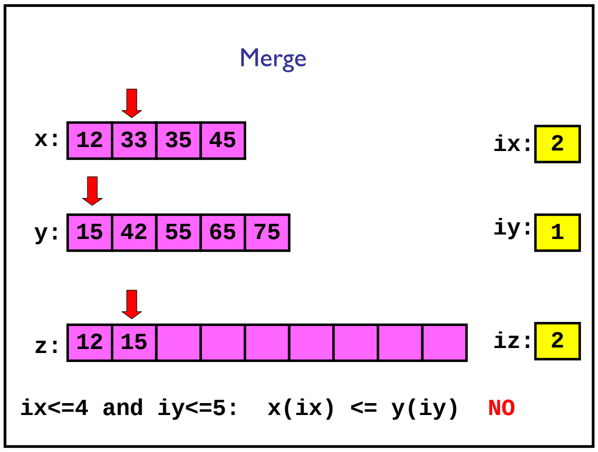
Merge

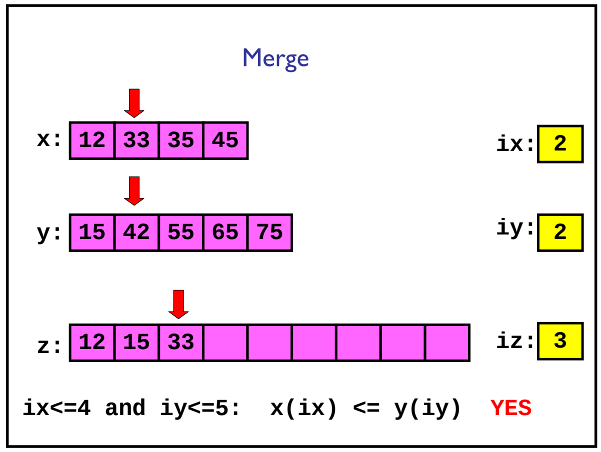
x: | 12 | 33 | 35 | 45 | ix: 1

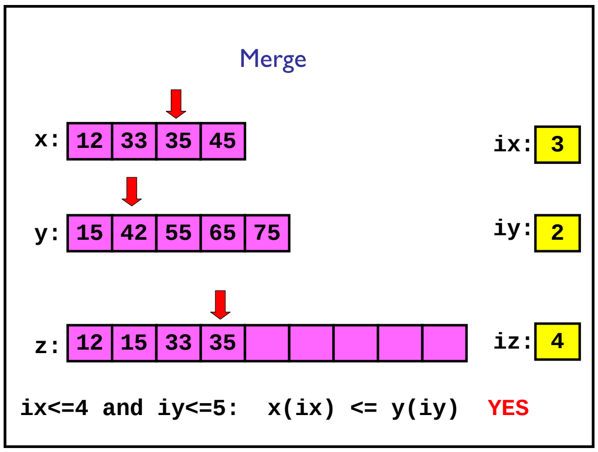
y: | 15 | 42 | 55 | 65 | 75 | iy: 1

z: | 12 | | | | | | | | | iz: 1

ix<=4 and iy<=5: x(ix) <= y(iy) YES








```
function z = merge(x,y)
nx = length(x); ny = length(y);
z = zeros(1, nx+ny);
ix = 1; iy = 1; iz = 1;
while ix<=nx && iy<=ny



end
% Deal with remaining values in x or y
```

```
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);
      yL = mergeSort(x(1:m));
      yR = mergeSort(x(m+1:n));
      y  = merge(yL,yR);
end
```

Lecture 25 64

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

Lecture 25 65