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

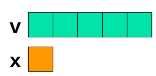
Searching for an item in a collection

Is the collection organized?
What is the organizing scheme?

Indiana Jones and the Raiders of the Lost Ark

Not organized?

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



- Linear search

```
% 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 …

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 <u>tiny</u>, <u>tiny</u> fraction of those entries?

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
```

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.

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

April 17, 2008 Lecture 24 13

```
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

        L= m;
    else

        R= m;
    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

April 17, 2008 Lecture 24 18

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...
- Let's look at **merge sort**
- An example of the "divide and conquer" approach

April 17, 2008 Lecture 24 19

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…

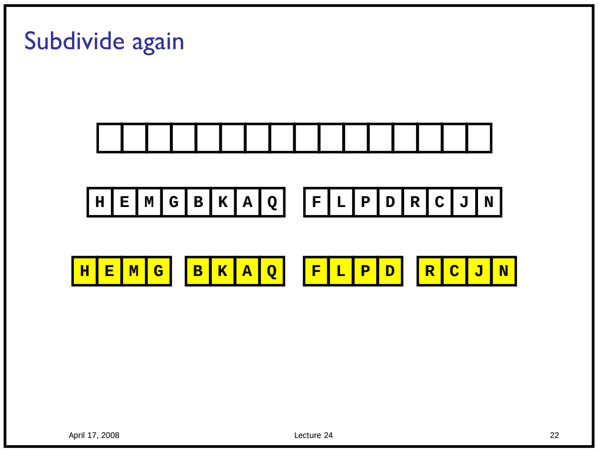
Subdivide the sorting task

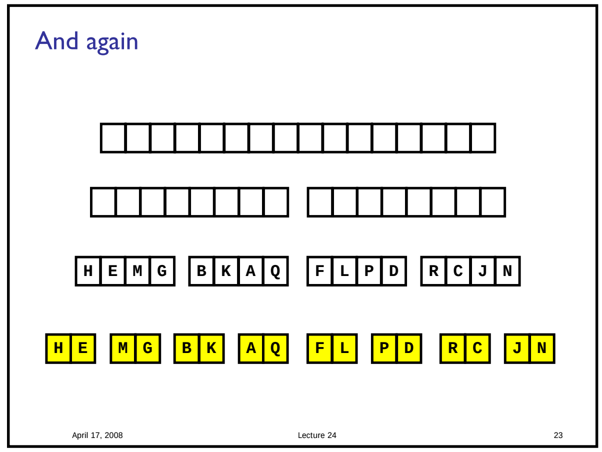
H	E	M	G	B	K	A	Q	F	L	P	D	R	C	J	N

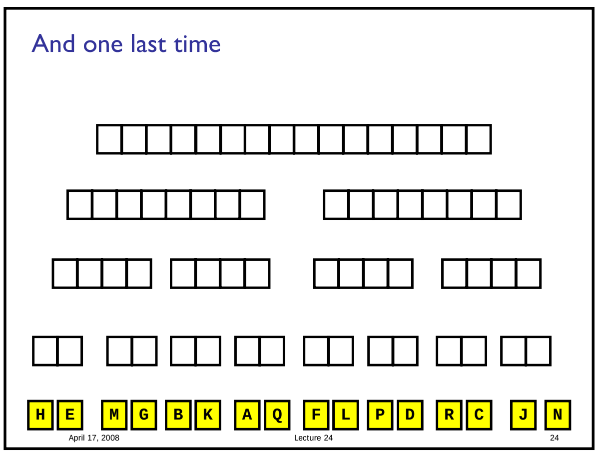
H	E	M	G	B	K	A	Q

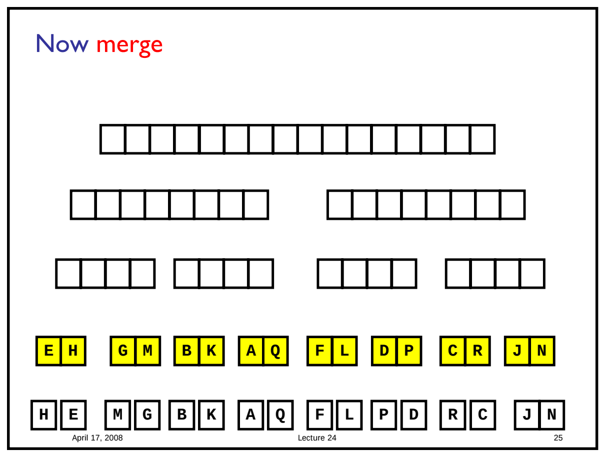
F	L	P	D	R	C	J	N

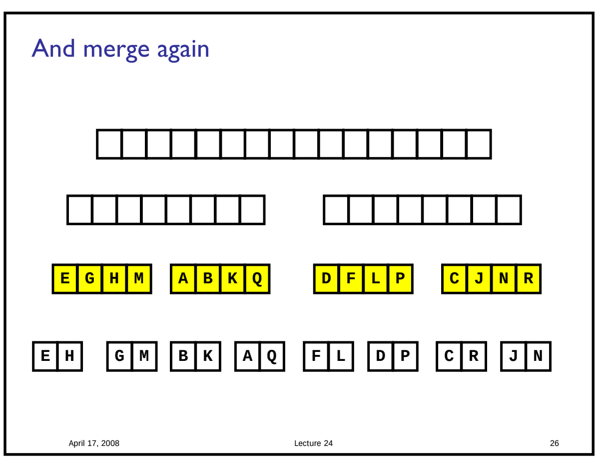
April 17, 2008 Lecture 24 21

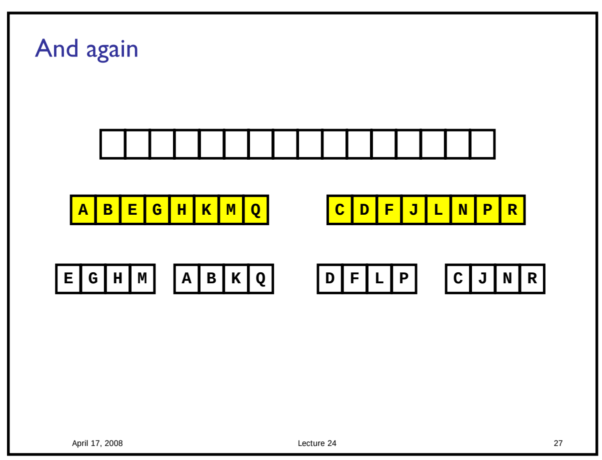

Subdivide again
H E M G B K A Q
F L P D R C J N
H E M G
B K A Q
F L P D
R C J N
April 17, 2008
Lecture 24
22


And again
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
April 17, 2008
Lecture 24
23


And one last time
H
E
M
G
B
K
A
Q
F
L
P
D
R
C
J
N
April 17, 2008
Lecture 24
24


Now merge
E H
G M
B K
A Q
F L
D P
C R
J N
H
E
M
G
B
K
A
Q
F
L
P
D
R
C
J
N
April 17, 2008
Lecture 24
25


And merge again
E G H M
A B K Q
D F L P
C J N R
E H
G M
B K
A Q
F L
D P
C R
J N
April 17, 2008
Lecture 24
26


And again
A B E G H K M Q
C D F J L N P R
E G H M
A B K Q
D F L P
C J N R
April 17, 2008
Lecture 24
27

And one last time

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

A	B	E	G	H	K	M	Q

C	D	F	J	L	N	P	R

Done!

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


```
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
```


An important sub-problem is the merging to 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