

- Previous Lecture:
  - Recursion – partitioning a triangle
  - Insertion Sort
  - (Read about Bubble Sort in *Insight*)
- Today's Lecture:
  - “Divide and conquer” strategies
    - Binary search
    - Merge sort

## Announcements

- P6 due **Wednesday** at 11pm
- **Final exam:**
  - Saturday, 5/16, 9am, **Barton Hall Indoor Track WEST**
- Please fill out course evaluation on-line, see “Exercise 15”
- Regular office/consulting hours end Wednesday night. Look for new hours for next week
- Pick up papers during consulting hours at Carpenter
- **Read announcements on course website!**

Lecture 27

3

## Searching for an item in an unorganized collection?

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

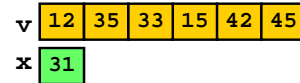


- Linear search

Lecture 27

17

```
% 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
```



Lecture 27

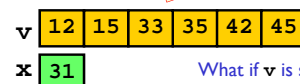
19

```
% 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 ...

```
% 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
```



What if  $v$  is sorted?

Lecture 27

22

## 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?

Lecture 27

25

## 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

```

Lecture 27

27

## 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

Lecture 27

28

## 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.

Lecture 27

29

```

% 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

```

$n$  comparisons against the target  
 are needed in worst case,  
 $n = \text{length}(v)$ .

Lecture 27

30

## 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!

$n$	$\log_2(n)$
100	7
1000	10
10000	13

Lecture 27

31

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

↑
↑
↑

$L: 1$ 
 $v(Mid) \leq x$

$Mid: 6$

$R: 12$

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

$x < v(\text{Mid})$

So throw away the right half...

Lecture 27 33

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

$L: \boxed{6}$        $v(\text{Mid}) \leq x$

$\text{Mid}: \boxed{7}$


$R: \boxed{9}$

So throw away the left half...

Lecture 27 34

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
---

$v(\text{Mid}) \leq x$

So throw away the left half...

35

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
----	----	----	----	----	----	----	----	----	----	----	----

↑ ↑

**Done because**  
 $R - L = 1$

L: 8

Mid: 8

R: 9

Lecture 27 36

```
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 v(m) <= x
        L = m;
    else
        R = m;
    end
end
```

20	30	40	46	50	52	68	70		
0	1	2	3	4	5	6	7	8	9

Play with `showBinarySearch.m`

Binary search is efficient, but we need to sort the vector in the first place so that we can use binary search

- Many different algorithms out there...
- We saw insertion sort (and read about bubble sort)
- Let's look at **merge sort**
- An example of the “divide and conquer” approach using recursion

Lecture 27

43

Which task is “easier,” **sort a length 1000 array** or **merge\* two length 500 sorted arrays into one?**

A. Sort

B. Merge

\*Merge two sorted arrays so that the resultant array is sorted

Lecture 27

46

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

Lecture 27

57

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
----	----	----	----	----	----	----	----	----

Lecture 27

59

### Merge

x: 

12	33	35	45
----	----	----	----

ix: 

1
---

y: 

15	42	55	65	75
----	----	----	----	----

iy: 

1
---

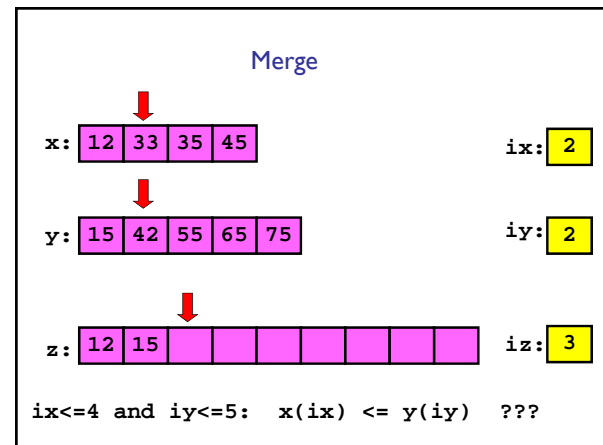
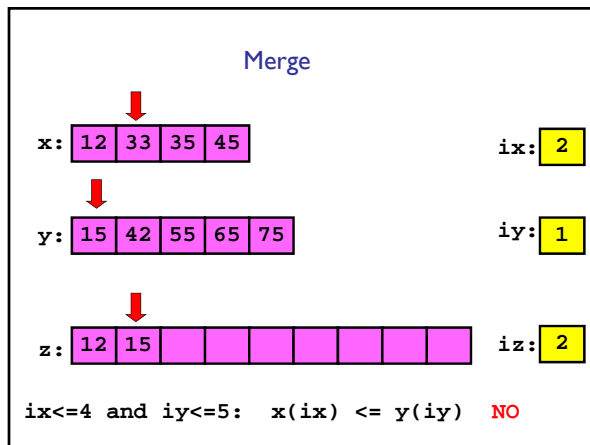
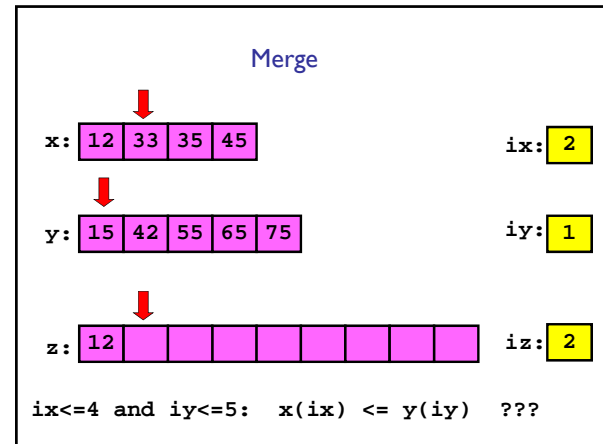
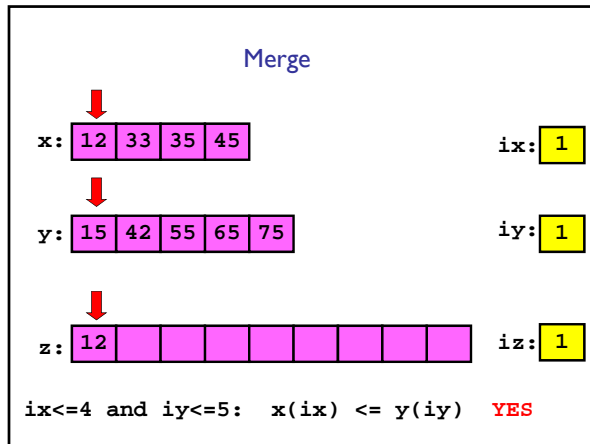
z: 

--	--	--	--	--	--	--	--	--

iz: 

1
---

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



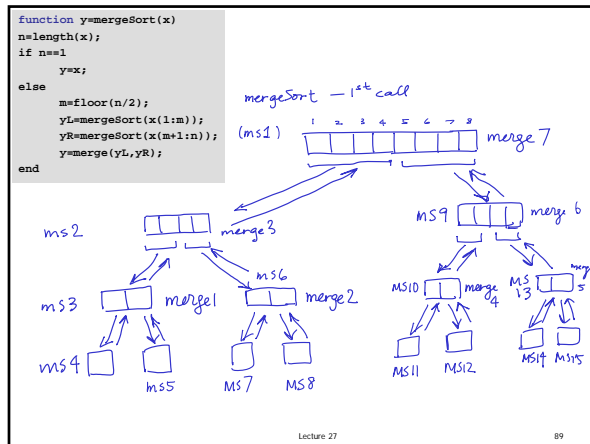
```
function z = merge(x,y)
nx = length(x); ny = length(y);
z = zeros(1, nx+ny);
ix = 1; iy = 1; iz = 1;
while ix<=nx && iy<=ny
    if x(ix) <= y(iy)
        z(iz)= x(ix); ix=ix+1; iz=iz+1;
    else
        z(iz)= y(iy); iy=iy+1; iz=iz+1;
    end
end
while ix<=nx % copy remaining x-values
    z(iz)= x(ix); ix=ix+1; iz=iz+1;
end
while iy<=ny % copy remaining y-values
    z(iz)= y(iy); iy=iy+1; iz=iz+1;
end
```

```
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 27

85



How do merge sort and insertion sort compare?

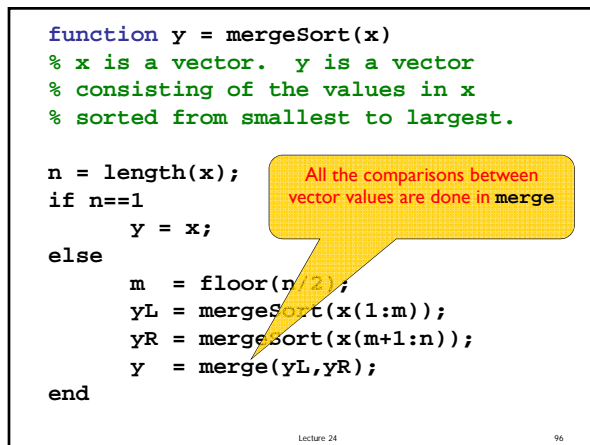
- Insertion sort: (worst case) makes  $k$  comparisons to insert an element in a sorted array of  $k$  elements. For an array of length  $N$ :

\_\_\_\_\_ for big  $N$

- Merge sort: \_\_\_\_\_
- Insertion sort is done *in-place*; merge sort (recursion) requires much more memory

Lecture 24

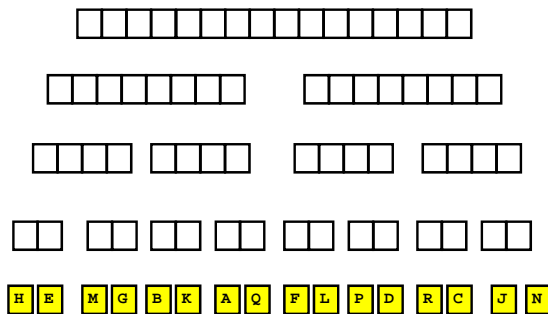
94



```
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
    if x(ix) <= y(iy)
        z(iz) = x(ix); ix=ix+1; iz=iz+1;
    else
        z(iz) = y(iy); iy=iy+1; iz=iz+1;
    end
end
while ix<=nx % copy remaining x-values
    z(iz) = x(ix); ix=ix+1; iz=iz+1;
end
while iy<=ny % copy remaining y-values
    z(iz) = y(iy); iy=iy+1; iz=iz+1;
end
```

Lecture 24 97

Merge sort:  $\log_2(N)$  “levels”;  $N$  comparisons each level



How to choose??

- Depends on application
- Merge sort is especially good for sorting **large data set** (but watch out for memory usage)
- Insertion sort is “order  $N^2$ ” at **worst case**, but what about an **average case**? If the application requires that you *maintain* a sorted array, insertion sort may be a good choice

Lecture 24

101

### What we learned...

- Develop/implement **algorithms** for problems
- Develop programming skills
  - Design, implement, document, test, and debug
- Programming “tool bag”
  - Functions for reducing redundancy
  - Control flow (if-else; loops)
  - Recursion
  - Data structures
  - Graphics
  - File handling

Lecture 27

103

### What we learned... (cont'd)

- Applications and concepts
  - Image processing
  - Object-oriented programming
  - Sorting and searching—you should know the algorithms covered
  - Divide-and-conquer strategies
  - Approximation and error
  - Simulation
  - Computational effort and efficiency

Lecture 27

104

### Computing gives us *insight* into a problem

- Computing is not about getting one answer!
- We build models and write programs so that we can “play” with the models and programs, learning—gaining insights—as we vary the parameters and assumptions
- Good models require domain-specific knowledge (and experience)
- Good programs ...
  - are modular and cleanly organized
  - are well-documented
  - use appropriate data structures and algorithms
  - are reasonably efficient in time and memory

Lecture 27

105

### Final Exam

- Saturday 5/16, 9-11:30am, Barton Hall indoor tracks WEST
- Covers entire course; some emphasis on material after Prelim 2
- Closed-book exam, no calculators
- Bring student ID card
- Check for announcements on webpage:
  - Study break office/consulting hours
  - Review session time and location
  - Review questions
  - List of potentially useful functions

Lecture 27

108