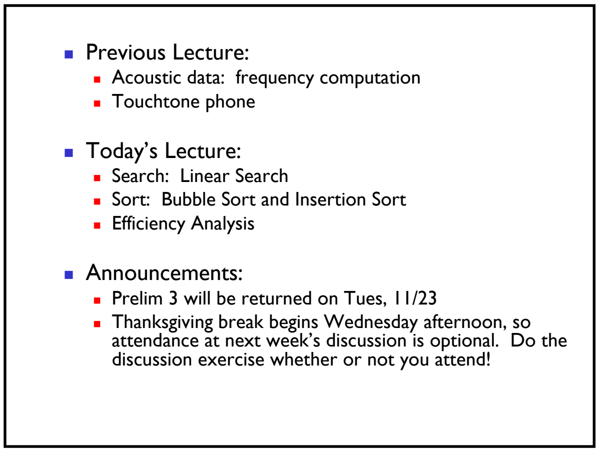
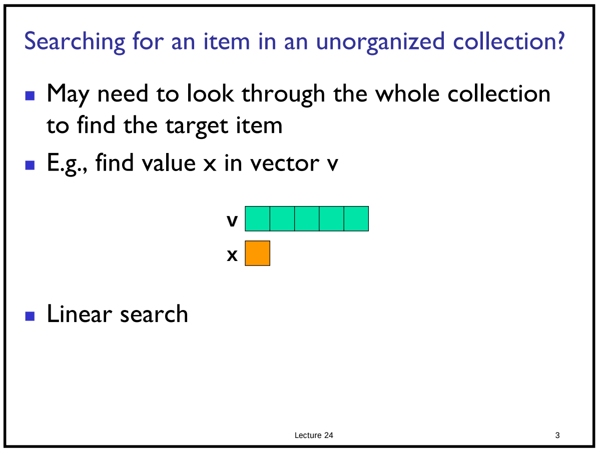


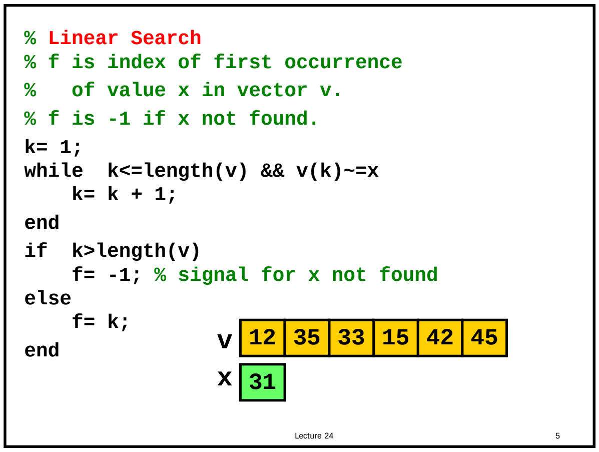
- Previous Lecture:
  - Acoustic data: frequency computation
  - Touchtone phone
- Today's Lecture:
  - Search: Linear Search
  - Sort: Bubble Sort and Insertion Sort
  - Efficiency Analysis
- Announcements:
  - Prelim 3 will be returned on Tues, 11/23
  - Thanksgiving break begins Wednesday afternoon, so attendance at next week's discussion is optional. Do the discussion exercise whether or not you attend!



Searching for an item in an unorganized collection?

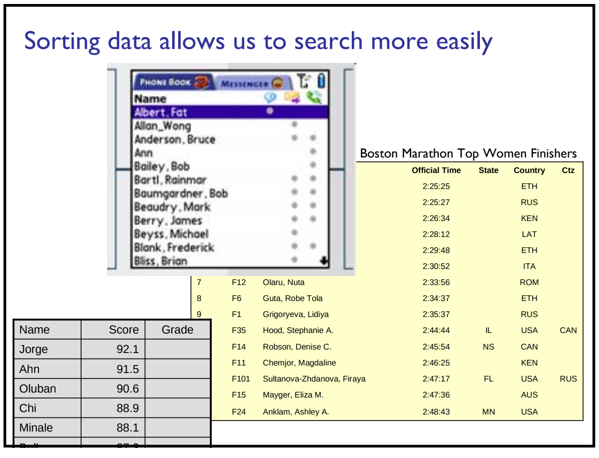
- May need to look through the whole collection to find the target item
- E.g., find value x in vector v

- Linear search



```
% 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
```

v	12	35	33	15	42	45
x	31					



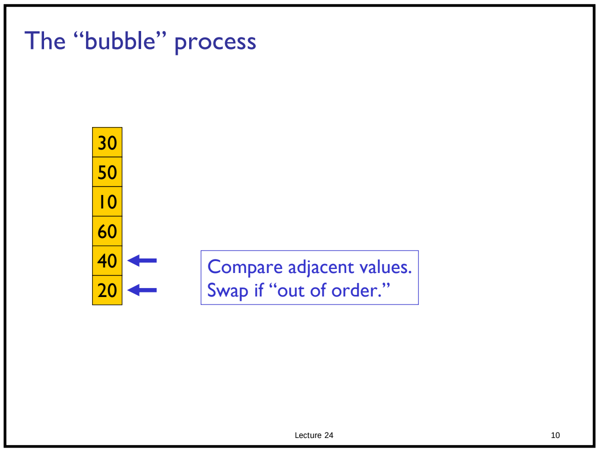
Sorting data allows us to search more easily

Name
Albert, Fat
Allan_Wong
Anderson, Bruce
Ann
Bailey, Bob
Bartl, Rainmar
Baumgardner, Bob
Beaudry, Mark
Berry, James
Beyss, Michael
Blank, Frederick
Bliss, Brian

Boston Marathon Top Women Finishers

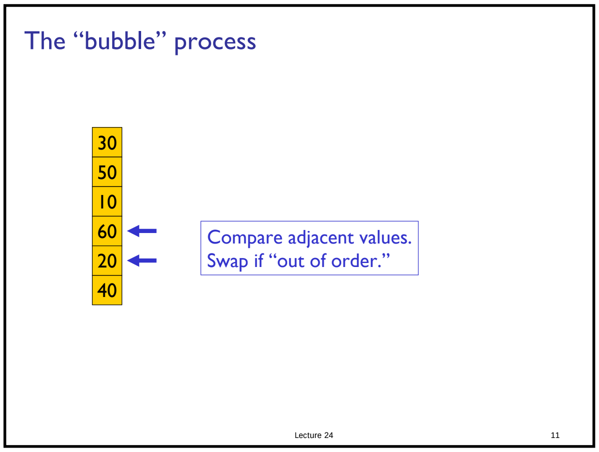
			Official Time	State	Country	Ctz
			2:25:25		ETH	
			2:25:27		RUS	
			2:26:34		KEN	
			2:28:12		LAT	
			2:29:48		ETH	
			2:30:52		ITA	
7	F12	Olaru, Nuta	2:33:56		ROM	
8	F6	Guta, Robe Tola	2:34:37		ETH	
9	F1	Grigoryeva, Lidiya	2:35:37		RUS	
	F35	Hood, Stephanie A.	2:44:44	IL	USA	CAN
	F14	Robson, Denise C.	2:45:54	NS	CAN	
	F11	Chemjor, Magdaline	2:46:25		KEN	
	F101	Sultanova-Zhdanova, Firaya	2:47:17	FL	USA	RUS
	F15	Mayger, Eliza M.	2:47:36		AUS	
	F24	Anklam, Ashley A.	2:48:43	MN	USA	

Name	Score	Grade
Jorge	92.1	
Ahn	91.5	
Oluban	90.6	
Chi	88.9	
Minale	88.1	



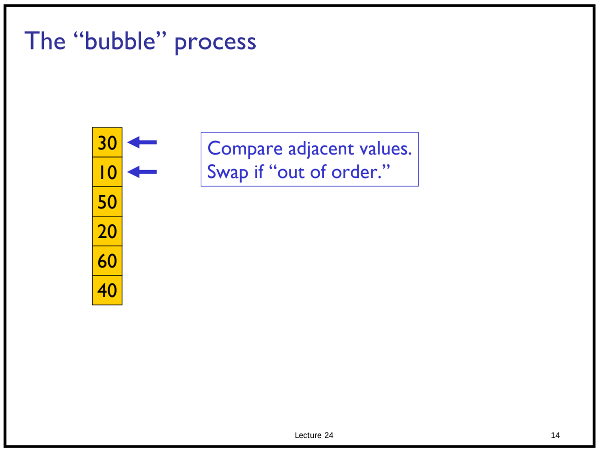
The "bubble" process

Compare adjacent values.
Swap if "out of order."



The "bubble" process

Compare adjacent values.
Swap if "out of order."

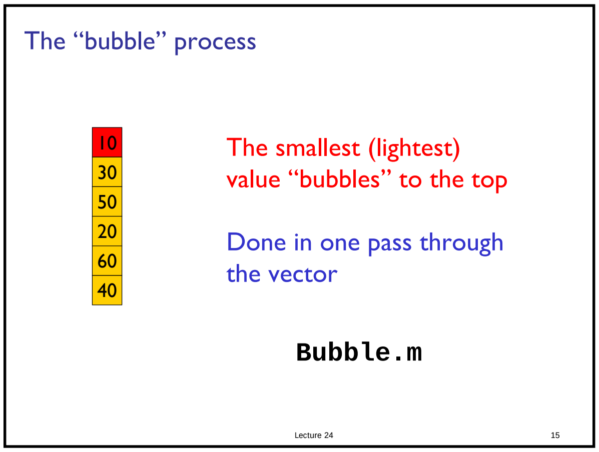


The "bubble" process

30 ←
10 ←
50
20
60
40

Compare adjacent values.
Swap if "out of order."

Lecture 24 14



The "bubble" process

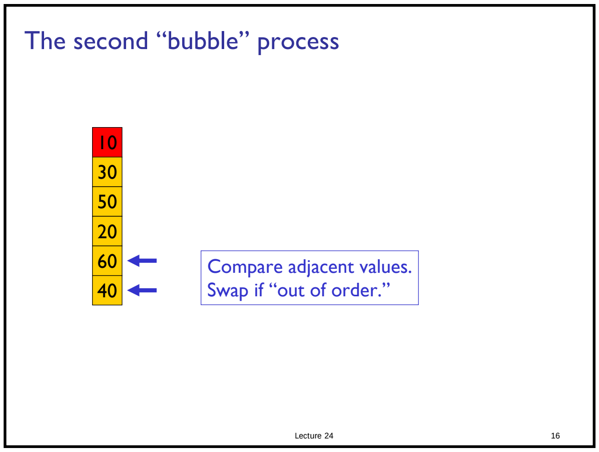
10
30
50
20
60
40

The smallest (lightest) value "bubbles" to the top

Done in one pass through the vector

Bubble.m

Lecture 24 15

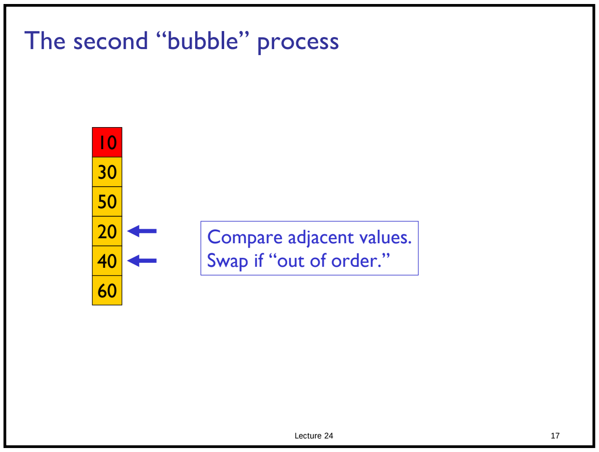


The second "bubble" process

10
30
50
20
60 ←
40 ←

Compare adjacent values.
Swap if "out of order."

Lecture 24 16

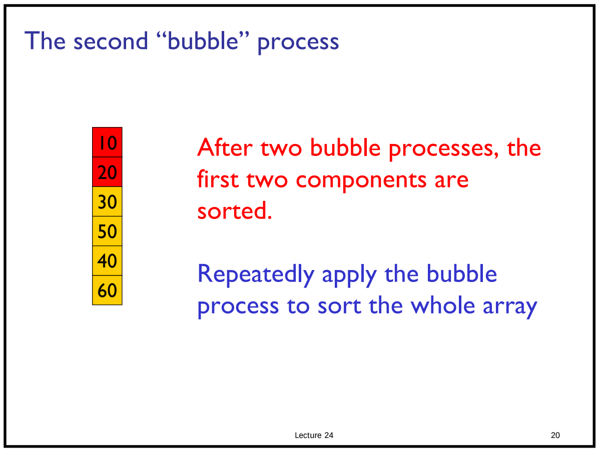


The second "bubble" process

10
30
50
20 ←
40 ←
60

Compare adjacent values.
Swap if "out of order."

Lecture 24 17



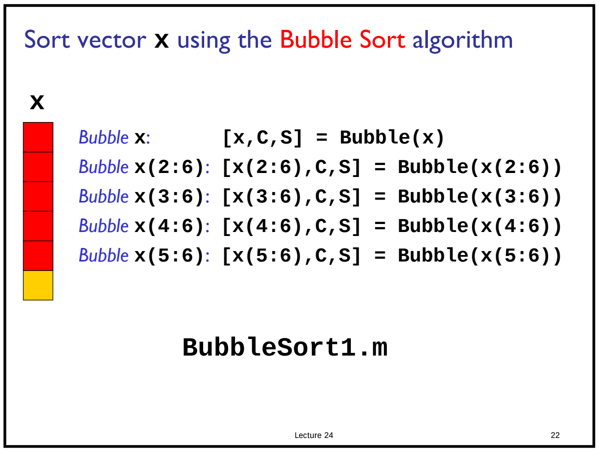
The second "bubble" process

10
20
30
50
40
60

After two bubble processes, the first two components are sorted.

Repeatedly apply the bubble process to sort the whole array

Lecture 24 20



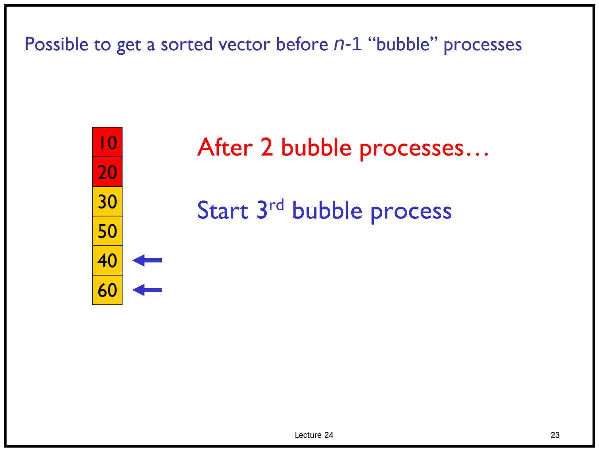
Sort vector **x** using the Bubble Sort algorithm

x

Bubble **x**: `[x,C,S] = Bubble(x)`
Bubble **x(2:6)**: `[x(2:6),C,S] = Bubble(x(2:6))`
Bubble **x(3:6)**: `[x(3:6),C,S] = Bubble(x(3:6))`
Bubble **x(4:6)**: `[x(4:6),C,S] = Bubble(x(4:6))`
Bubble **x(5:6)**: `[x(5:6),C,S] = Bubble(x(5:6))`

BubbleSort1.m

Lecture 24 22

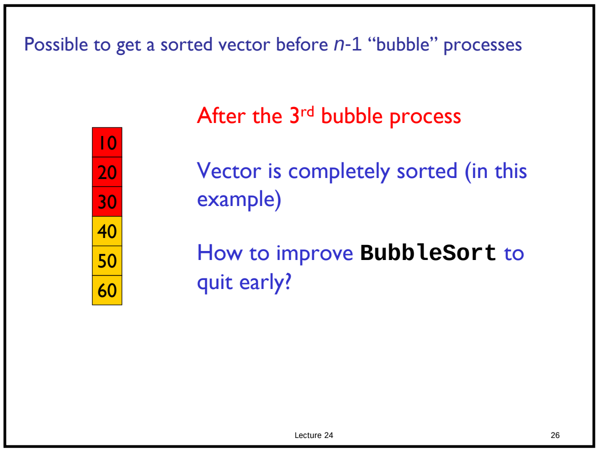


Possible to get a sorted vector before *n*-1 "bubble" processes

10
20
30
50
40 ←
60 ←

After 2 bubble processes…

Start 3rd bubble process



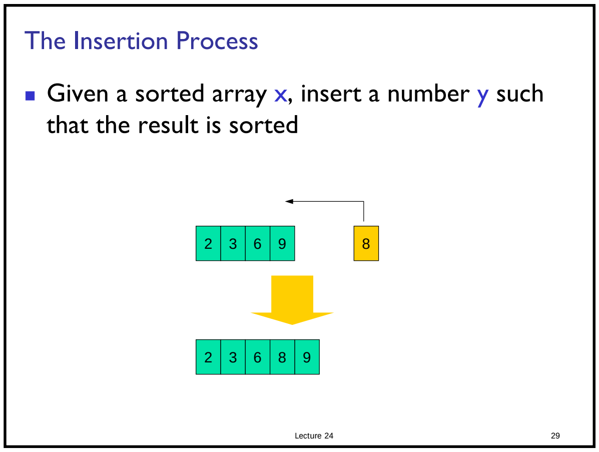
Possible to get a sorted vector before *n*-1 "bubble" processes

10
20
30
40
50
60

After the 3rd bubble process

Vector is completely sorted (in this example)

How to improve **`BubbleSort`** to quit early?

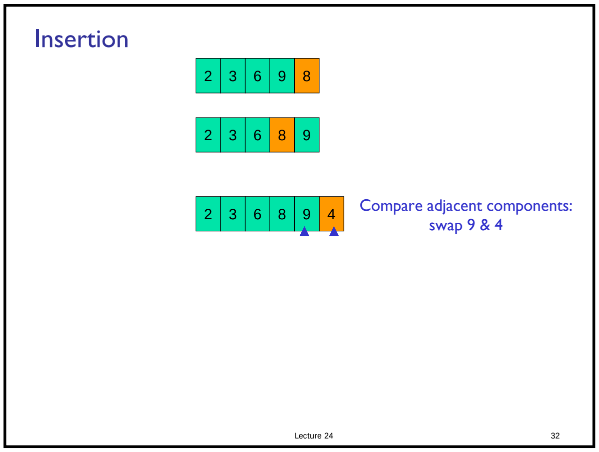


The Insertion Process

- Given a sorted array x, insert a number y such that the result is sorted

2 3 6 9   8

2 3 6 8 9



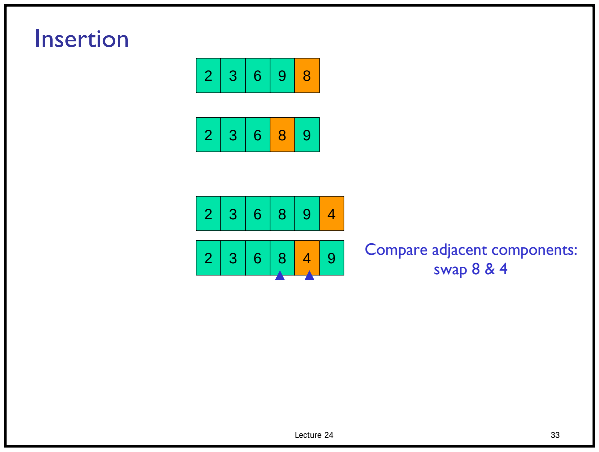
Insertion

2 3 6 9 8

2 3 6 8 9

2 3 6 8 9 4

Compare adjacent components: swap 9 & 4



Insertion

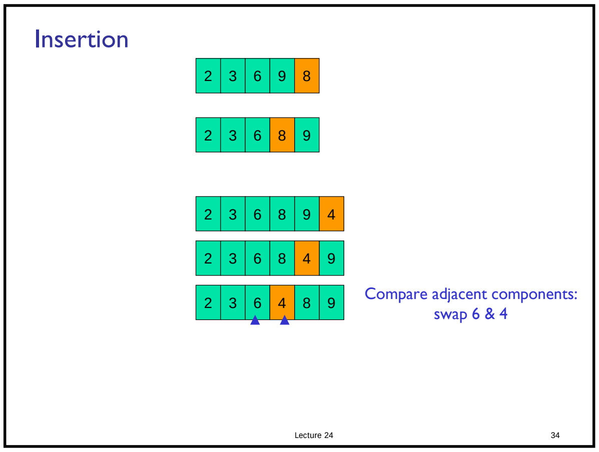
2 3 6 9 8

2 3 6 8 9

2 3 6 8 9 4

2 3 6 8 4 9

Compare adjacent components: swap 8 & 4



Insertion

2 3 6 9 8

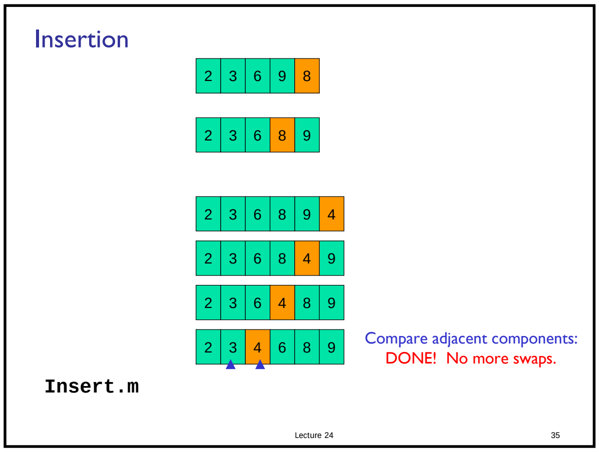
2 3 6 8 9

2 3 6 8 9 4

2 3 6 8 4 9

2 3 6 4 8 9

Compare adjacent components: swap 6 & 4



Insertion

2 3 6 9 8

2 3 6 8 9

2 3 6 8 9 4

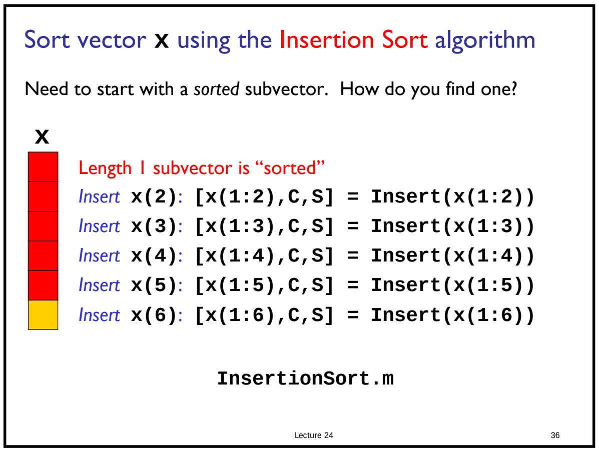
2 3 6 8 4 9

2 3 6 4 8 9

2 3 4 6 8 9

Compare adjacent components:
DONE! No more swaps.

`Insert.m`



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"

Insert `x(2)`: `[x(1:2),C,S] = Insert(x(1:2))`

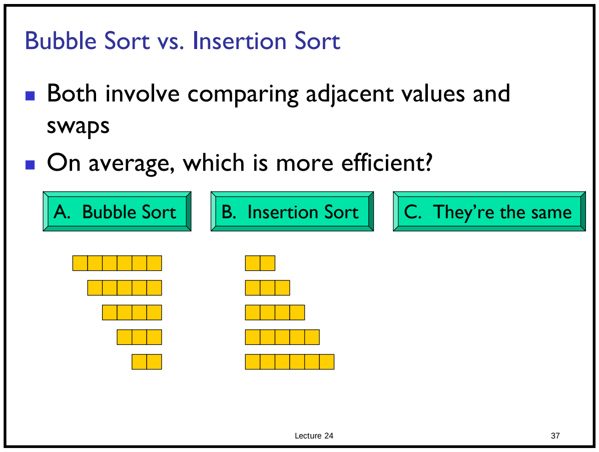
Insert `x(3)`: `[x(1:3),C,S] = Insert(x(1:3))`

Insert `x(4)`: `[x(1:4),C,S] = Insert(x(1:4))`

Insert `x(5)`: `[x(1:5),C,S] = Insert(x(1:5))`

Insert `x(6)`: `[x(1:6),C,S] = Insert(x(1:6))`

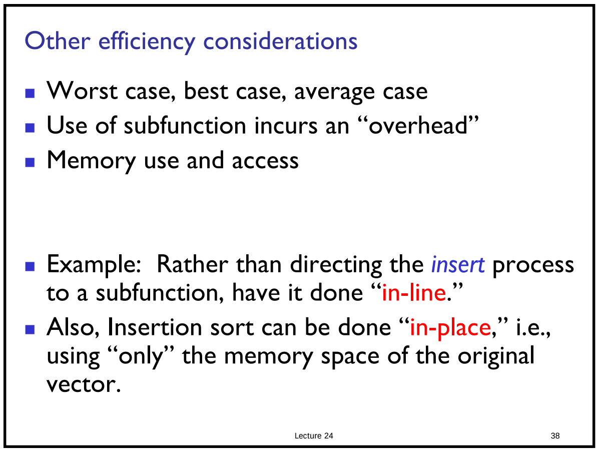
`InsertionSort.m`



Bubble Sort vs. Insertion Sort

- Both involve comparing adjacent values and swaps
- On average, which is more efficient?

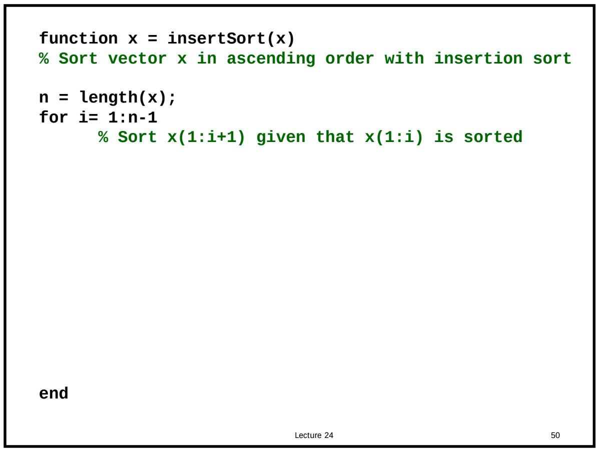
A. Bubble Sort | B. Insertion Sort | C. They're the same



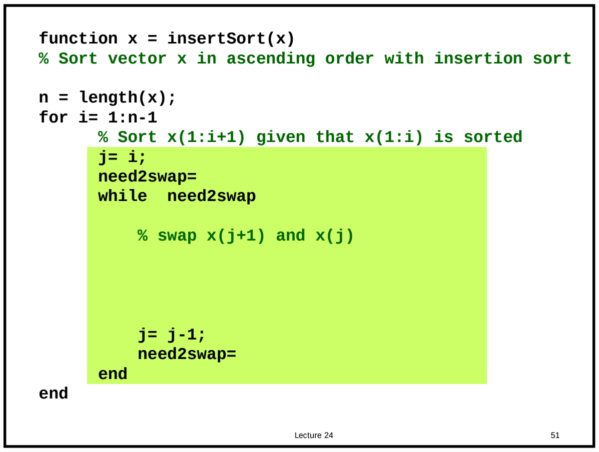
Other efficiency considerations

- 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.



```
function x = insertSort(x)
% Sort vector x in ascending order with insertion sort

n = length(x);
for i= 1:n-1
      % Sort x(1:i+1) given that x(1:i) is sorted







end
```



```
function x = insertSort(x)
% Sort vector x in ascending order with insertion sort

n = length(x);
for i= 1:n-1
      % Sort x(1:i+1) given that x(1:i) is sorted
      j= i;
      need2swap=
      while  need2swap

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




          j= j-1;
          need2swap=
      end
end
```