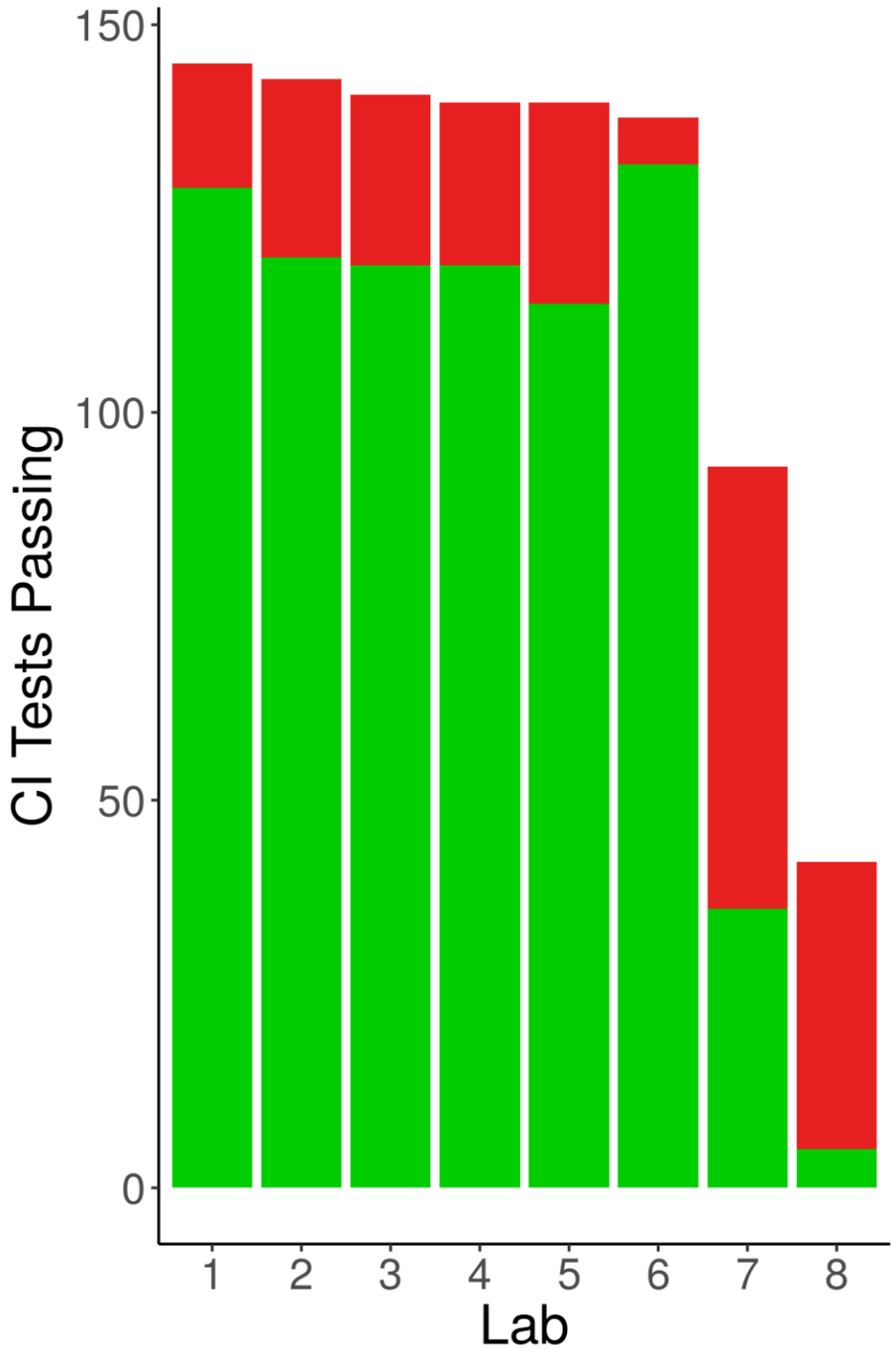
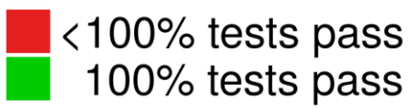
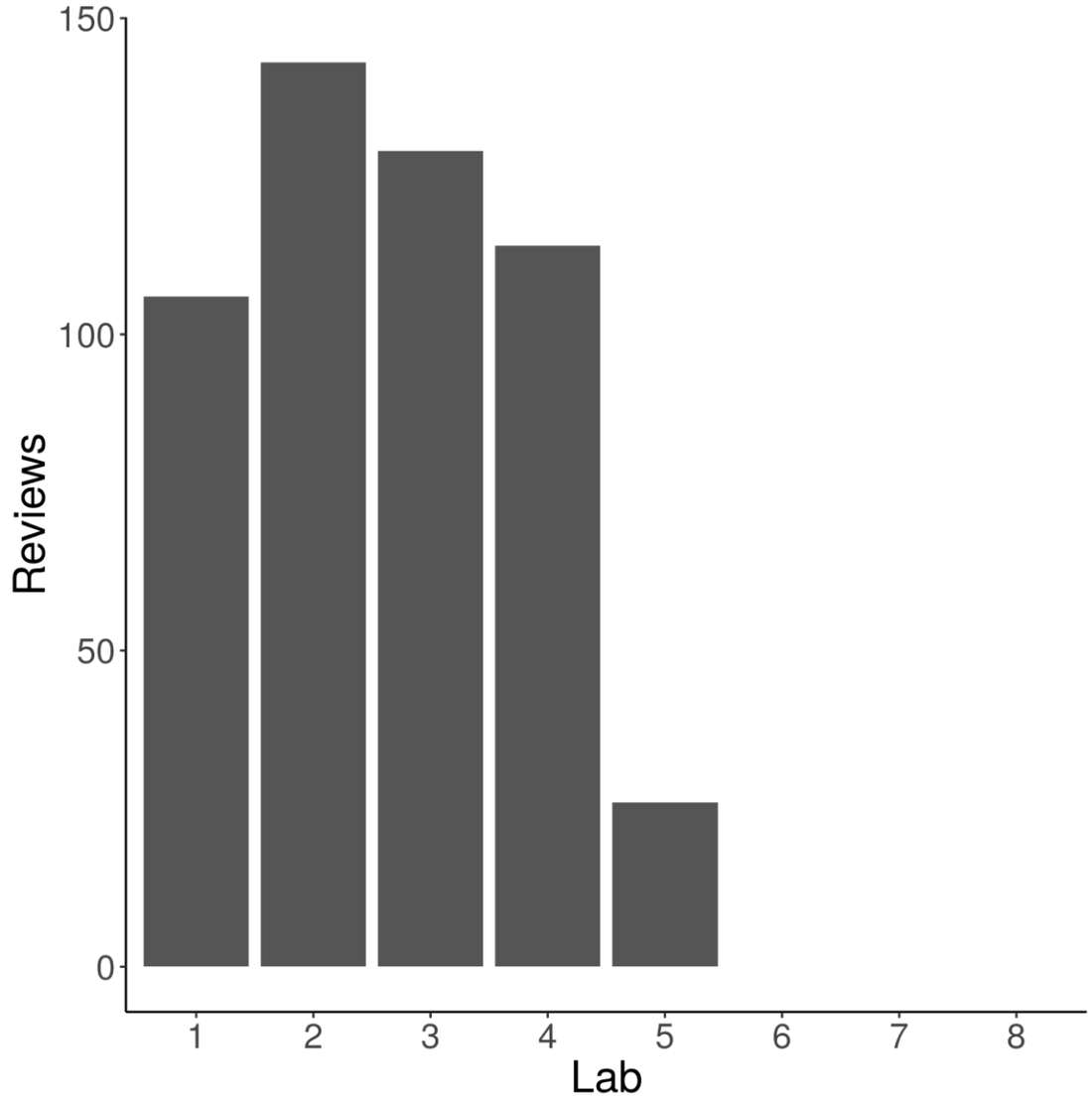

150
100
50
0
CI Tests Passing
1
2
3
4
5
6
7
8
Lab


<100% tests pass
100% tests pass


150
100
50
0
Reviews
1
2
3
4
5
6
7
8
Lab

- The deadlines for labs 1 - 6 code reviews:
  - Group 1: Friday 15 March
  - Group 2: Monday 18 March

- Friday 8th, group 1
  - Lab 8
  - Lab 7 deadline
  - Lab 6 late submission
- Monday 11th, group 2
  - Lab 8
  - Lab 7 deadline
  - Lab 6 late submission

# Today's lab (group 1)

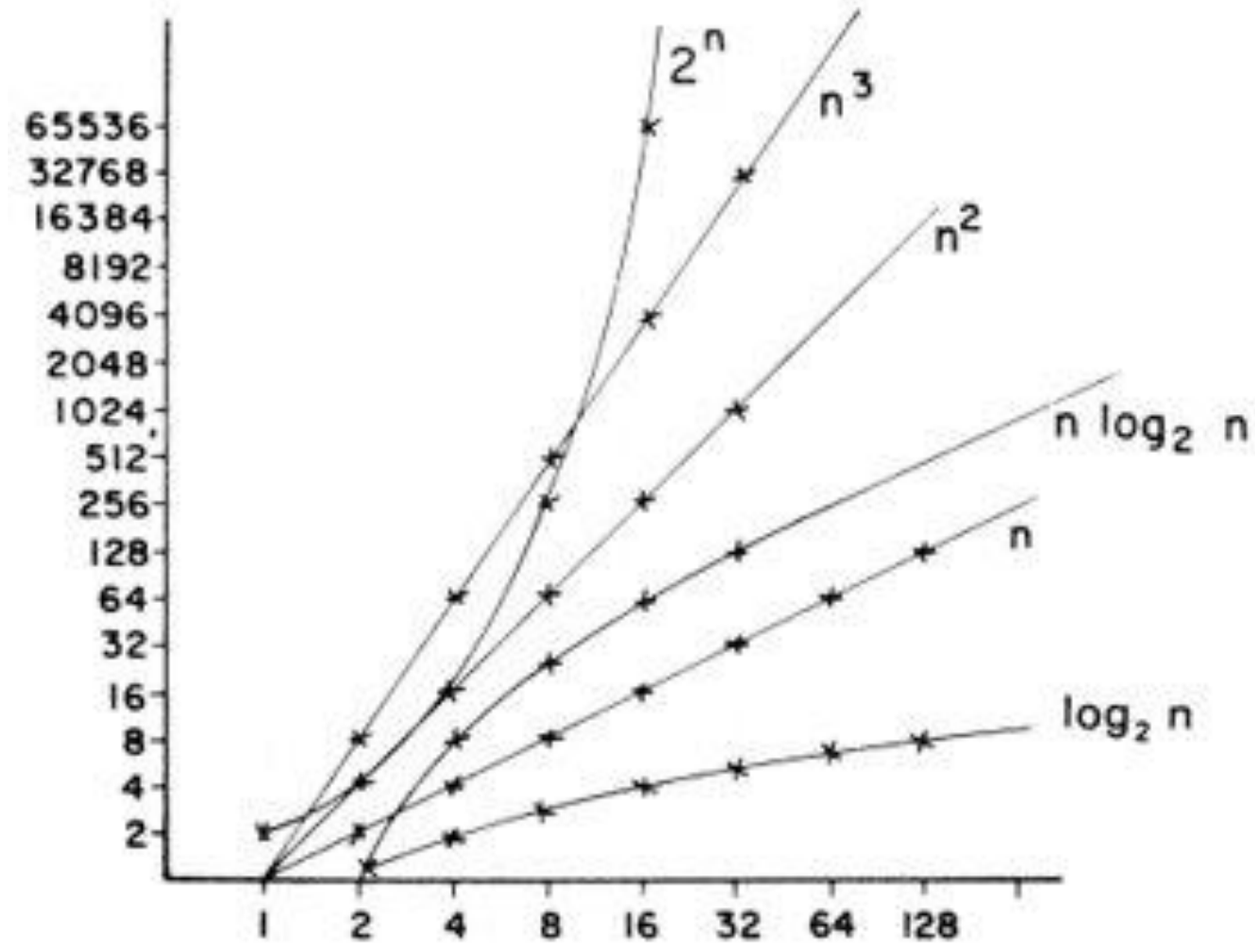
Software Development 3 (F27SG)

# Lecture 16

# Basic Sorting

Rob Stewart

# Sorting Algorithms



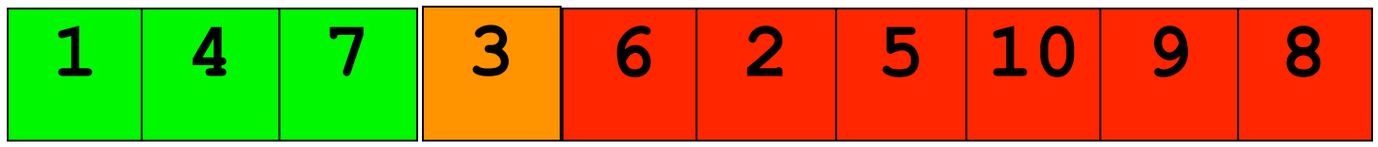

- Many sorting algorithms
- some books say 50+ algorithms
- Trade off between:
  - **simplicity** and **speed**
- **Simple algorithms**
  - fine for small data
  - easier to understand/implement
  - too slow for large data (as we will see they are $O(n^2)$)
  - examples include **Insertion-Sort** and **Bubble-Sort**
- **Speedy algorithms** harder to understand but faster
  - examples include: **Merge-Sort** and **Quick-Sort** (next lecture)

# Overview

- cover some of the **simpler** sorting algorithms
  - the next lecture will look at the **quicker** ones..
- By the end of this lecture you should
  - know the concept of sorting
  - know the **Insertion-Sort** algorithm
  - know the *quadratic* function: $O(N^2)$
  - be able to analyse Insertion-Sort using Big-O
  - be familiar with **Bubble-Sort**

# Insertion-Sort

- Iterates through an array, start at beginning

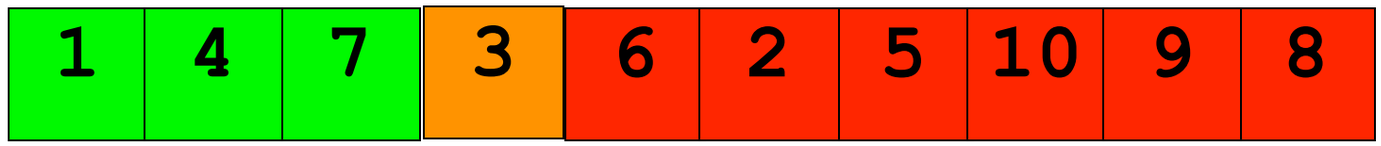



  - at step **n**:
    1. $n^{th}$ element moved to correct place on left side
    2. all elements larger are shifted one place to right
  - in ***nth*** iteration:
    - the 0..n-1 indexes of the array are sorted (the left of **n**)
    - while the n..length-1 indexes are not (the right of **n**)
  - this continues until the end of the array is reached

# Insertion-Sort

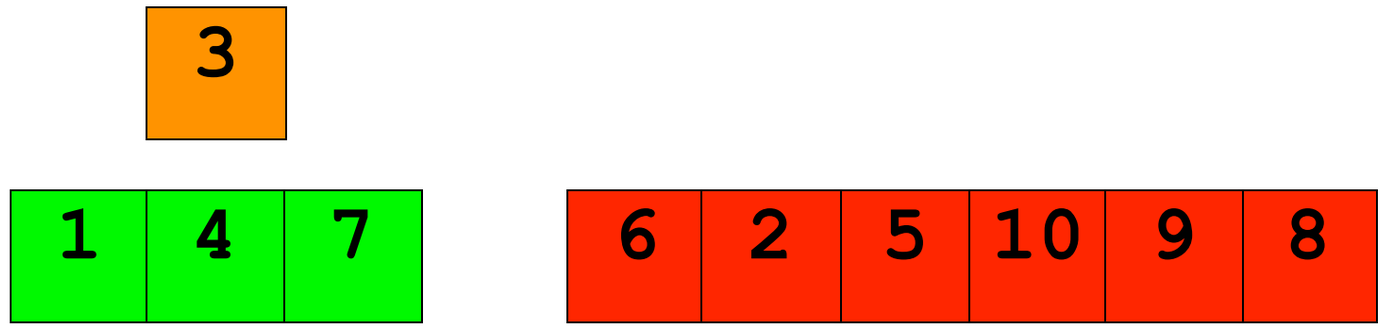
- illustrate how the $4^{th}$ iteration works

- 3 < 7 so it has to be moved to the right location



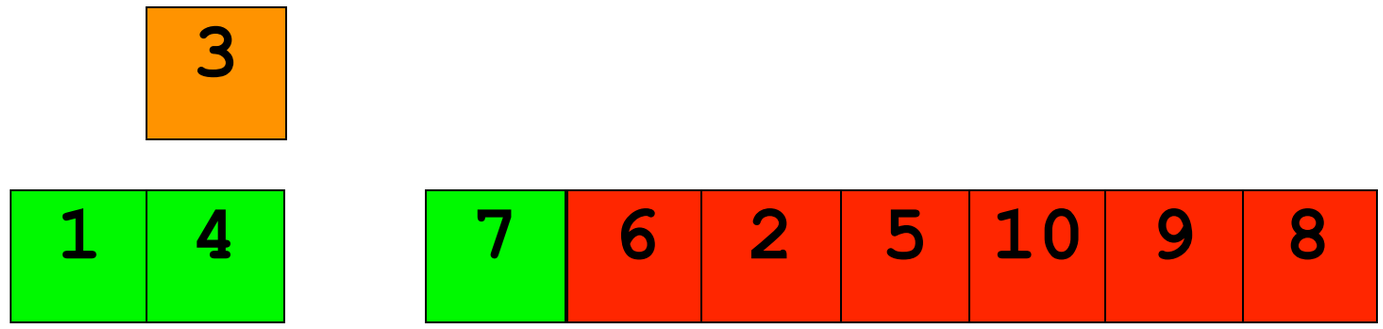
# Insertion-Sort

- illustrate how the 4th iteration works
  - next it shifts the elements between the new position and the old position one to the right



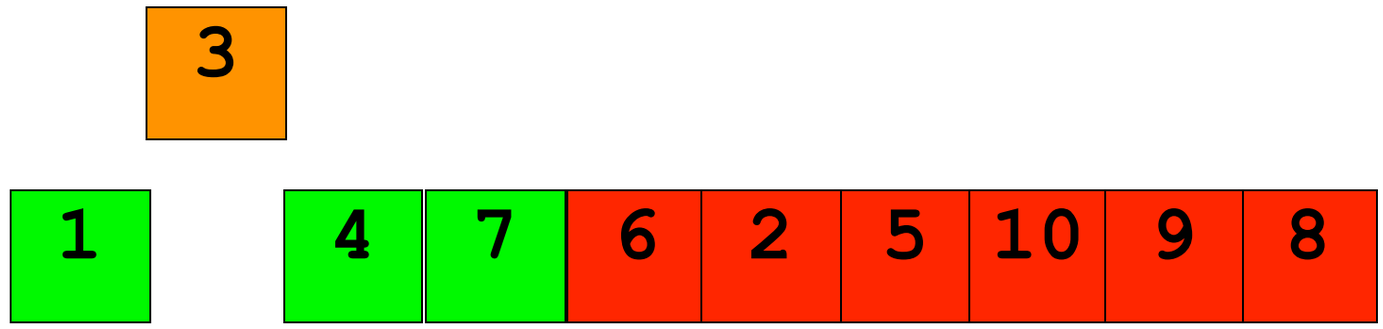
# Insertion-Sort

- illustrate how the 4th iteration works
  - next it shifts the elements between the new position and the old position one to the right



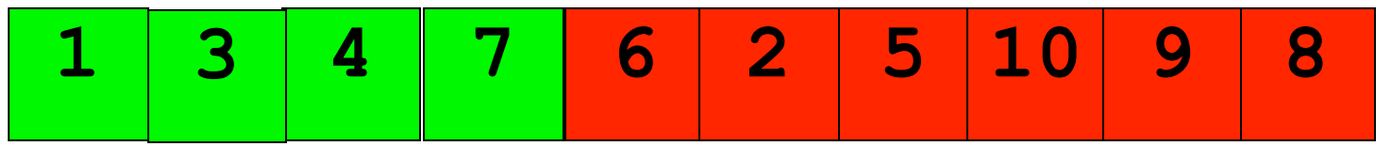
# Insertion-Sort

- illustrate how the 4th iteration works
  - next it shifts the elements between the new position and the old position one to the right



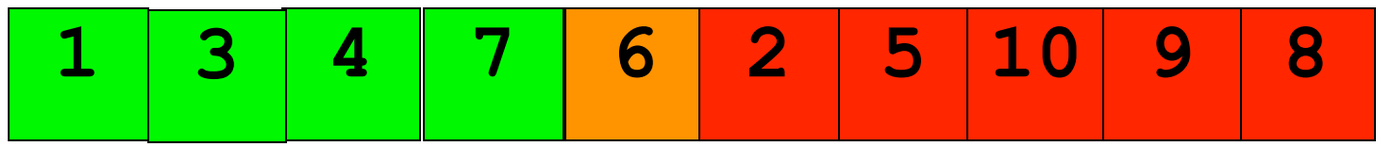
# Insertion-Sort

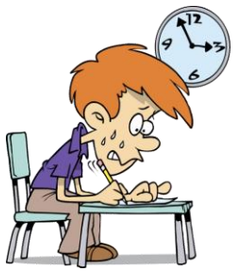
- illustrate how the 4th iteration works
  - finally, 3 is inserted in the correct place



# Insertion-Sort

- illustrate how the 4th iteration works
  - .. and we continue with the next element





# Exercise

Insertion-Sort iterates through an array, starting at the beginning
- in the $n^{th}$ iteration:
  - the 0..n-1 indexes of the array are sorted (the left of n)
  - while the n..length-1 indexes are not (the right of n)
- at this step the process is:
  - the $n^{th}$ element is moved to the correct place on the left side
    - so this remains sorted
  - all elements larger than this are shifted one place to the right
- this continues until the end of the list is reached

Use Insertion-Sort to sort the list {4,3,6,5,2,1}.

- Show the value of the list for each step
- Count the number of times you have to move an element in the array. To get you started:
  - we start at the first position
  - we swap 4 and 3 (position 0 and position 1): {3,4,6,5,2,1} [1 move]
  - …

# Insertion-Sort in Java

## sorting integers

Eclipse demo

# Insertion-Sort in Java

## sorting integers

```
public static void insertionSort(int [] list){
   // iterates the array
  for(int i = 1; i < list.length; i++){
     int cur = list[i]; // select current elements
     int j = i - 1; // start at element before current element
     // find the correct place and move each element forward
     while(j >= 0 && list[j] > cur) {
        list[j+1] = list[j];
        // add the element to the correct place
        list[j] = cur;
        j--;
      }
  }
}
```

# Analysis of Insertion-Sort

- The insertion sort algorithm has such nested loops

```
for(int i = 0; i < list.length; i++) // outer loop
    ....
    while(j >= 0 && list[j] > cur) // inner loop
     ...
```

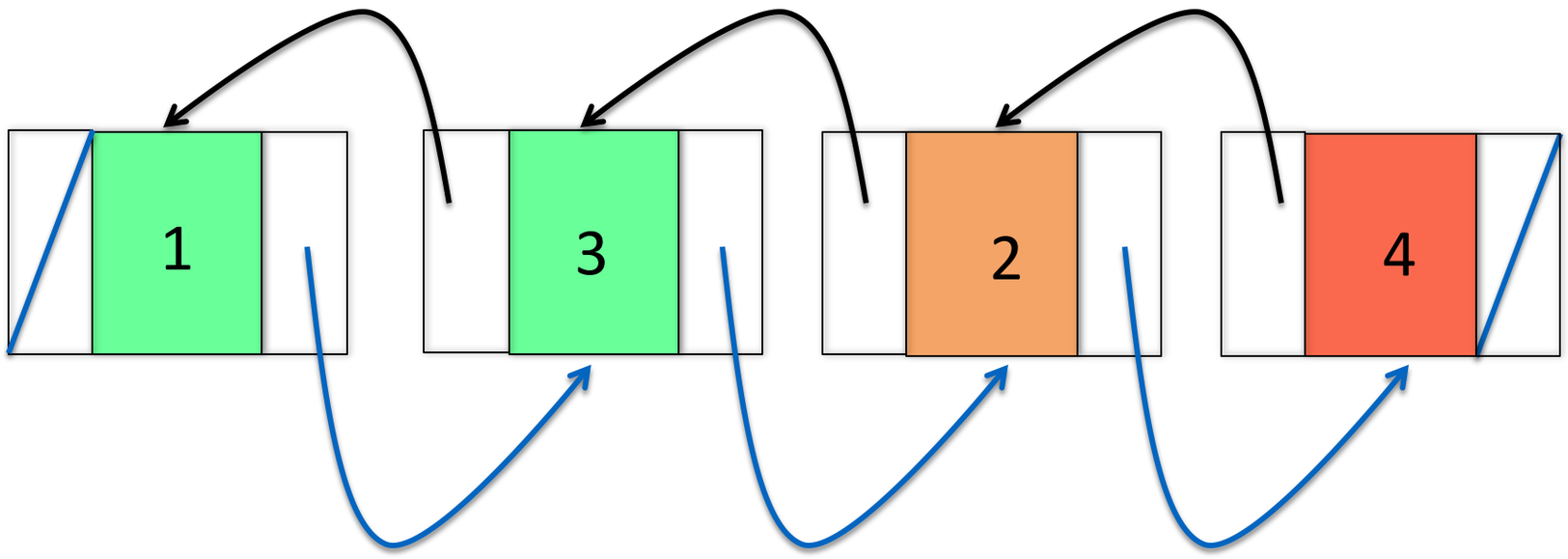
- The outer loop is linear O(N)
  - It will iterate through the entire array
- In the worse case the inner loop always reaches **j=0**
  - Is also linear O(N)
- Thus, it is a **quadratic growth** rate
  - O(N) operations, N number of times

# Big-O: The Quadratic Function O($N^2$)

- The quadratic function:
  $f(n) = n^2$
  returns the input (n) squared.
- Primitive operations increases at squared rate of input size
- Often appear in analysis of *nested loop* (loop within a loop)
  - *both* inner and outer loops perform operations *linear* number of times
  - outer loop applies inner loop **n** times
  - each inner loop applies **n** operations
  - hence the growth rate is: ***n (outer) times n (inner) = n x n = $n^2$***

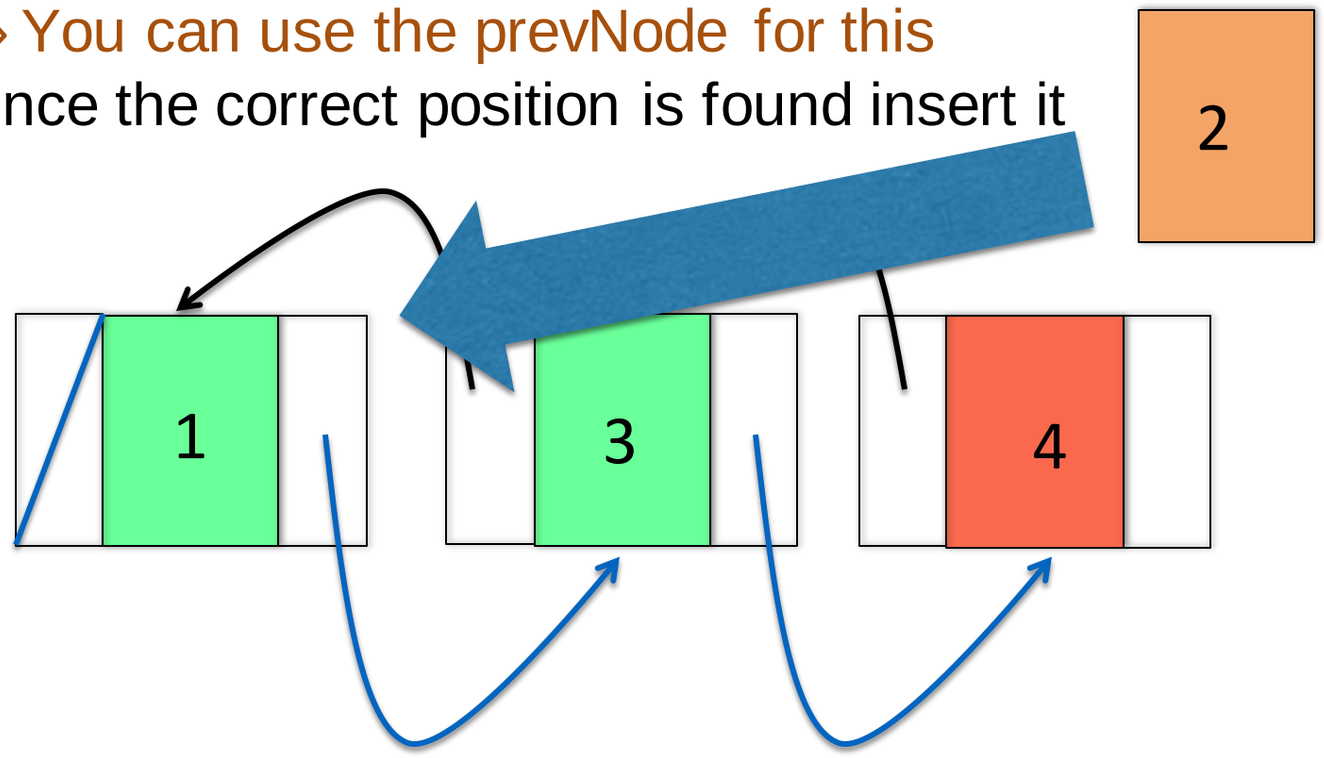
# Sorting Doubly Linked Lists

- We can use the same insertion sort algorithm for linked lists as we used for arrays
- Why?
  - We linearly traverse the list
    - You can use the nextNode for this
    - For each node
  - Remove the node (but remember it's value)
  - Go backwards to find the correct place to have the current node
    - » You can use the prevNode for this
  - Once the correct position is found insert it



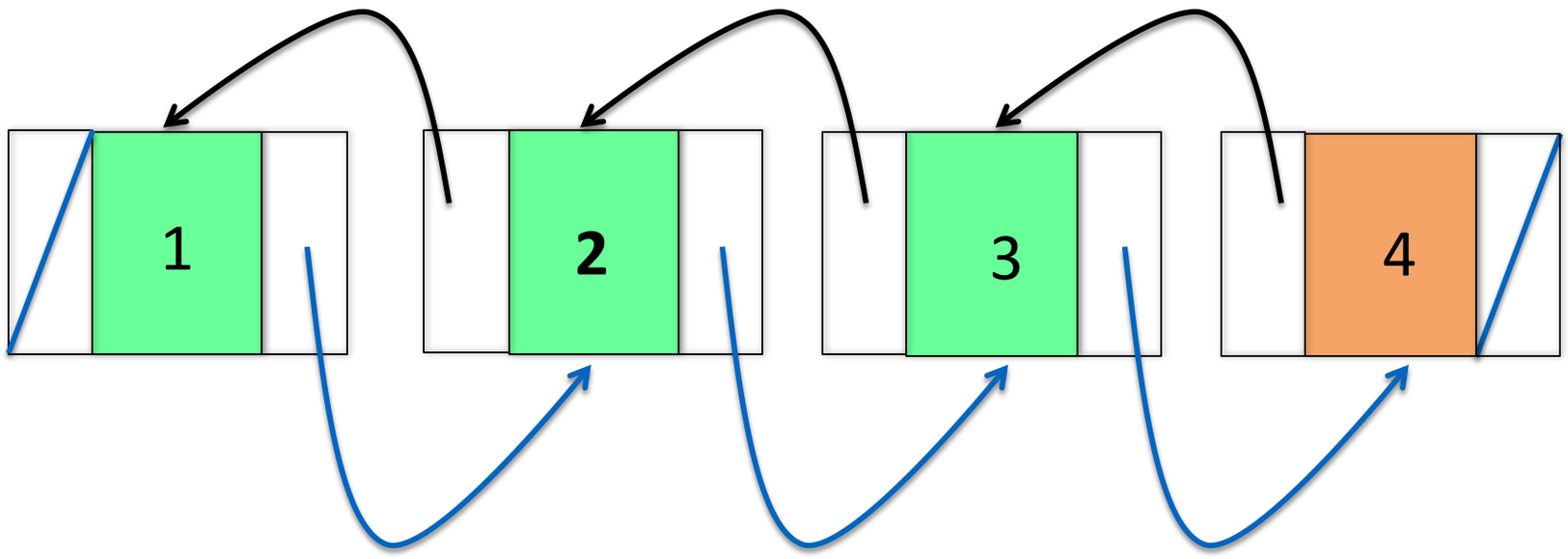
# Sorting Doubly Linked Lists

- We can use the same insertion sort algorithm for linked lists as we used for arrays
- Why?
  - We linearly traverse the list
    - You can use the nextNode for this
    - For each node
  - Remove the node (but remember it's value)
  - Go backwards to find the correct place to have the current node
    - » You can use the prevNode for this
  - Once the correct position is found insert it



# Sorting Doubly Linked Lists

- We can use the same insertion sort algorithm for linked lists as we used for arrays
- Why?
  - We linearly traverse the list
    - You can use the nextNode for this
    - For each node
  - Remove the node (but remember it's value)
  - Go backwards to find the correct place to have the current node
    - » You can use the prevNode for this
  - Once the correct position is found insert it



# Similar Sorting Algorithms
# Bubble-Sort

- Start at beginning of list and iterate through the list
  1. First set a boolean variable **swaps** to **false**
  2. For each step we compare with the next element
     - if next element is smaller then we swap them and set **swaps** to **true**
  3. When we reach the end of the array then
     - if **swaps** is **true** we go to step 1
     - else if **swaps** is **false** we terminate
- The end, rather than the beginning **will be sorted first**
- You will implement this algorithm in the lab 8 Q3

# Exercise

In **Bubble-Sort** we start at the beginning of the list and iterate through the list

1. First we set a boolean variable **swaps** to **false**
2. For each step we compare with the next element
   - if the next element is smaller then swap and set **swaps** to **true**
3. When we reach the end of the array then
   - if **swaps** is **true** we go to 1
   - else if **swaps** is **false** we terminate

Use **Bubblesort** to sort the list {4,3,6,5,2,1}; illustrate each step/move of the algorithm. To get you started, here are the first moves:

- we start with the first element (4) and compare it with the next (3), since 4 is larger we swap giving the list {3,4,6,5,2,1} and set swaps to true
- next we compare 4 with 6, which is left unchanged
- then we compare 6 with 5 and swap since 6 is larger giving the list {3,4,5,6,2,1} (swaps is already set to true)
- ….

# Lab 8 exercises

- **Insertion sort**
  - This lecture: pen and paper, Java implementation (arrays)
  - ***Lab 8 Q2:*** *implement JUnit tests*
  - ***Lab 8 Q4:*** *Java implementation (doubly linked lists)*
- **Bubble sort**
  - This lecture: pen and paper
  - ***Lab 8 Q2:*** *implement JUnit tests*
  - ***Lab 8 Q3:*** *Java implementation (arrays)*
- **Quick sort**
  - Next lecture: pen and paper
  - ***Lab 8 Q2:*** *implement JUnit tests*
  - ***Lab 8 Q5:*** *Java implementation (array list)*
- **Merge sort**
  - Next lecture: pen and paper, Java implementation (arrays)

# Summary

- You should
  - know a basic sorting algorithm called Insertion-Sort, and be familiar with it's relative Bubble-Sort
  - the quadratic function: $O(N^2)$
  - … and to analyse the compexity of Insertion-Sort

- The Sorting Algorithms webpage contains descriptions and animations of many common sorting algorithms
  - http://www.sorting-algorithms.com