Plan For Today

Review Quiz

Insertion Sort

How should we measure the complexity of an algorithm? (Big Oh)

Properties of sorting algorithms

Mergesort

Quicksort (if time)

Big Oh

Big Oh Motivation

We want a way to talk about how much time an algorithm takes to run as a function of the input size.

How much data can we process with this algorithm?

Can we sort a vector of a million elements?

Simple Definition

We could say that one algorithm is faster than another if it always runs in less time, for every input length.

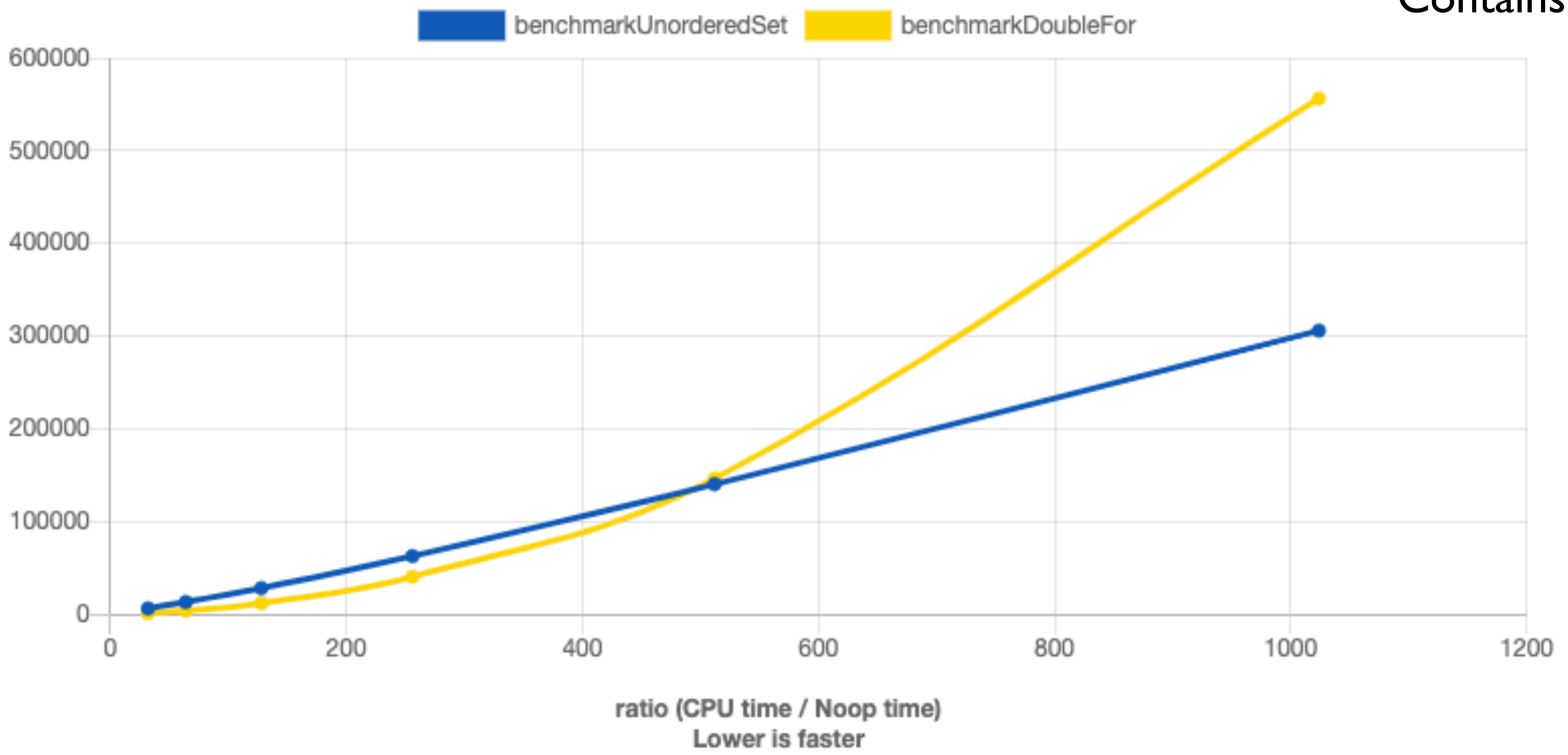
If $f, g : \{1, 2, 3, \ldots\} \to \{1, 2, 3, \ldots\}$ describe the running times as a function of the input length

$$f \leq g \iff f(n) \leq g(n) \text{ for every } n = 1, 2, 3, \ldots$$

What is the shortcoming of this definition?

Small Size Effects

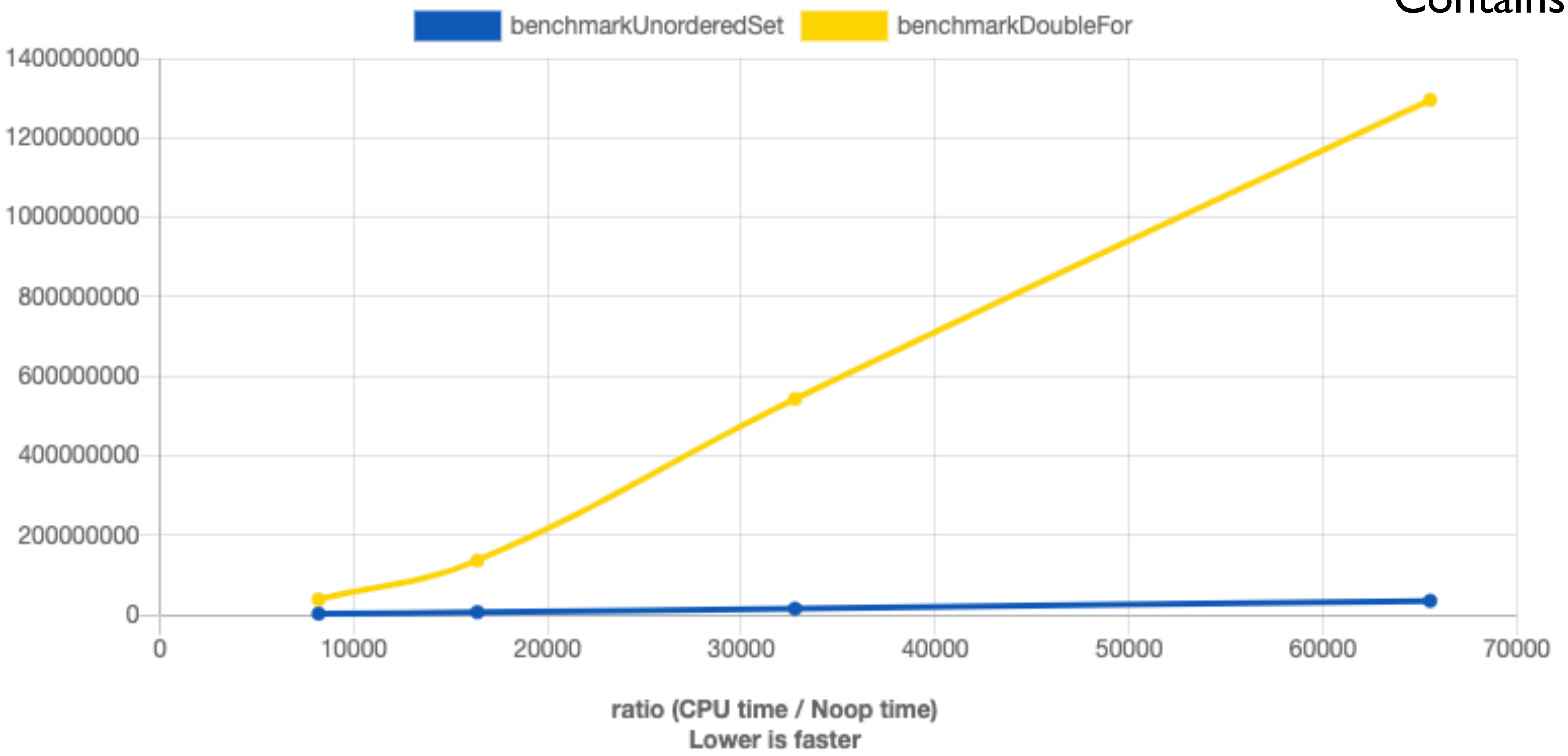
Leetcode 217:
Contains Duplicate



Which algorithm is better?

Small Size Effects

Leetcode 217:
Contains Duplicate



How about now?

LARGE problem sizes

Big Oh ignores small size effects.

It only cares about running time as the problem size becomes VERY big (goes to infinity).

New Attempt:

If $f, g : \{1, 2, 3, \ldots\} \to \{1, 2, 3, \ldots\}$ describe the running times as a function of the input length, say that

$f \leq g \iff$ there is some size n_0 such that $f(n) \leq g(n)$ for all $n \geq n_0$.

Ignore Constant factors

The second simplification big Oh makes is that it ignores constant factors.

$$1 + 2 + \cdots + n/2 + (n/2 + 1) + \cdots + (n - 1) + n$$

We had to do some work to compute this sum

It is much easier to see that it is at most n^2 and at least $n^2/4$.

Without knowing tiny details of an algorithm's implementation and the machine it is running on we can't hope to predict running time better than up to a constant factor.

Relaxation 2

If $f, g : \{1, 2, 3, \ldots\} \to \{1, 2, 3, \ldots\}$ describe the running times as a function of the input length

We forgive constant factors: $f \leq g$ if and only if for some constant $c > 0$

$f(n) \leq c \cdot g(n)$ for all large enough n .

This is exactly the definition of big Oh. In this case we say $f(n) = O(g(n))$.

Sufficient Condition

Look at the ratio $\frac{f(n)}{g(n)}$. If there is a constant c such that

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} \leq c$$

then $f(n) = O(g(n))$.

Questions

True or False: $100n = O(n)$

Questions

True or False: $\log n = O(n)$

Questions

True or False: $$\frac{n^2}{1000} = O(n)$$

Questions

True or False: $n \log n = O(n)$

Lower Bounds

> Unfortunately, people have occasionally been using the O-notation for lower bounds, for example when they reject a particular sorting method "because its running time is $O(n^2)$." I have seen instances of this in print quite often, and finally it has prompted me to sit down and write a Letter to the Editor about the situation.

Donald E. Knuth, "Big Omicron and Big Omega and Big Theta", 1976.

Lower Bounds

Unfortunately, people have occasionally been using the O-notation for lower bounds, for example when they reject a particular sorting method "because its running time is $O(n^2)$." I have seen instances of this in print quite often, and finally it has prompted me to sit down and write a Letter to the Editor about the situation.

Donald E. Knuth, "Big Omicron and Big Omega and Big Theta", 1976.

In computer science, **selection sort** is an in-place comparison sorting algorithm. It has an $O(n^2)$ time complexity, which makes it inefficient on large lists, and generally performs worse than the similar insertion sort. Selection sort is noted for its simplicity and has performance advantages over more complicated algorithms in certain situations, particularly where auxiliary memory is limited.

Wikipedia, Today

Big Omega

We need a way to talk about lower bounds on running time.

$f \geq g$ if and only if there is constant $c > 0$ such that

$f(n) \geq c \cdot g(n)$ for all large enough n .

This is exactly the definition of big Omega. In this case we say $f(n) = \Omega(g(n))$.

If $f(n) = \Omega(g(n))$ and $f(n) = O(g(n))$ then we say $f(n) = \Theta(g(n))$.

Questions

True or False: $n \log n = \Omega(n)$

Questions

True or False: $n = \Omega(n^2)$

Usage

cheat sheet

$f(n) = O(g(n))$	"$f(n) \leq g(n)$"
$f(n) = \Omega(g(n))$	"$f(n) \geq g(n)$"
$f(n) = \Theta(g(n))$	"$f(n) = g(n)$"

Caveats:

1) Don't care about constant factors

2) Don't care about behaviour for small n.

On the internet, when people use $O(\cdot)$ they almost always mean $\Theta(\cdot)$.

Common functions

Complexity	Example
$\Theta(1)$	
$\Theta(\log n)$	
$\Theta(n)$	
$\Theta(n \log n)$	
$\Theta(n^2)$	

Theta vs. Problem Size

n	$\Theta(1)$	$\Theta(\log n)$	$\Theta(n)$	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(n^3)$	$\Theta(2^n)$
10	1 ns	3 ns	10 ns	30 ns	100 ns	1 microsec	1 microsec
100	1 ns	6 ns	100 ns	600 ns	10 microsec	1 ms	40 trillion yrs
1,000	1 ns	10 ns	1 microsec	10 microsec	1 ms	1 sec	
10,000	1 ns	13 ns	10 microsec	130 microsec	100 ms	16 min	
100,000	1 ns	16 ns	100 microsec	1.6 ms	10 sec	277 hours	
1,000,000	1 ns	20 ns	1 ms	20 ms	16 min	32 yrs	

one operation per nanosecond

Sorting

Properties of Sorting Algos

Comparison Based:

In Place:

Stable:

Mergesort

Mergesort

Mergesort is a comparison based sorting algorithm with worst-case running time $\Theta(n \log n)$.

This is optimal for a comparison-based method.

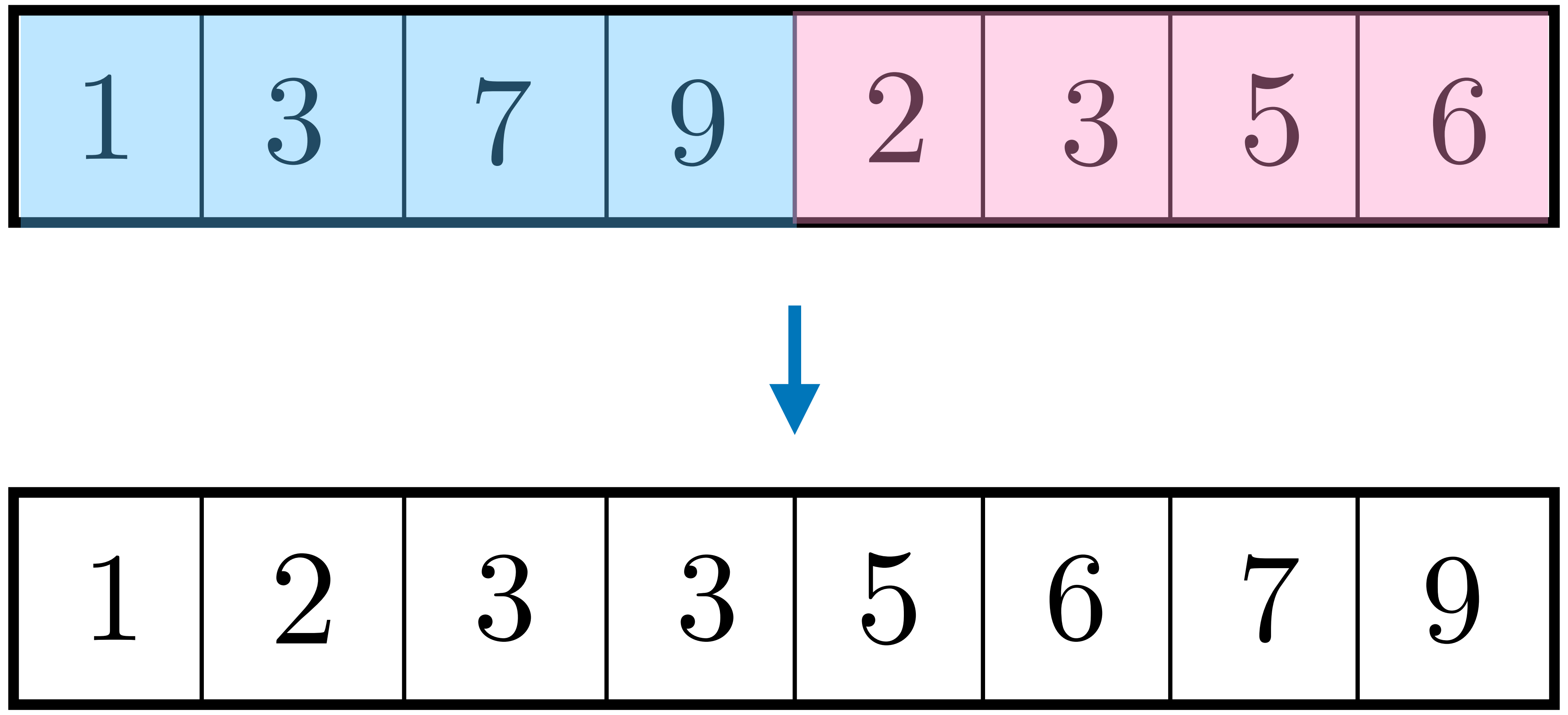
Mergesort is stable but is not in place.

Mergesort is a great example of a divide and conquer algorithm.

Merge Function

The heart of mergesort is merging together two sorted arrays.

Say we have an array of size n where the first half is sorted and the second half is sorted.



We want to merge these to completely sort the array.

Mergesort: Code

```
void mergesort(vecIt begin, vecIt end) {
  if (end - begin <= 1) {
    return;
  }
  vecIt mid = begin + (end - begin)/2;
  mergesort(begin, mid);
  mergesort(mid, end);
  merge(begin, mid, end);
}
```

Mergesort: Running Time

Let us assume the size of the original vector is a power of 2.

Then we have the recurrence

$$T(n) = 2T(n/2) + O(n)$$

$$T(1) = O(1)$$ base case

This is the exact same recurrence we had for the buy and sell stock problem.

The running time of mergesort is $O(n \log n)$.