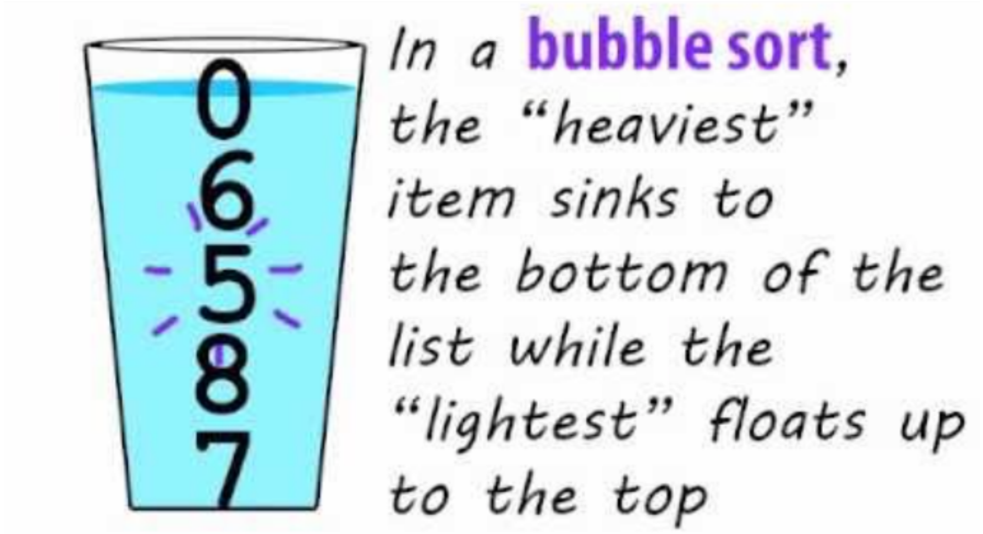
15-112
Fundamentals of Programming

Week 3 - Lecture 2:
Intro to efficiency + Searching and sorting + Big O



June 1, 2016

Principles of good programming

Correctness

Your program does what it is supposed to.

Handles all cases (e.g. invalid user input).

Maintainability

Readability, clarity of the code.

Reusability for yourself and others
(proper use of functions/methods and *objects*)

→ programs that are easy to handle and debug.

Efficiency

In terms of running time and memory space used.

Why care about efficiency?

multiplying two integers
sorting a list
factoring integers
computing Nash Equilibria of games
protein structure prediction
simulating quantum systems
building AI
proving theorems

The Plan

> How to properly measure running time

> Searching a given list

- Linear search
- Binary search

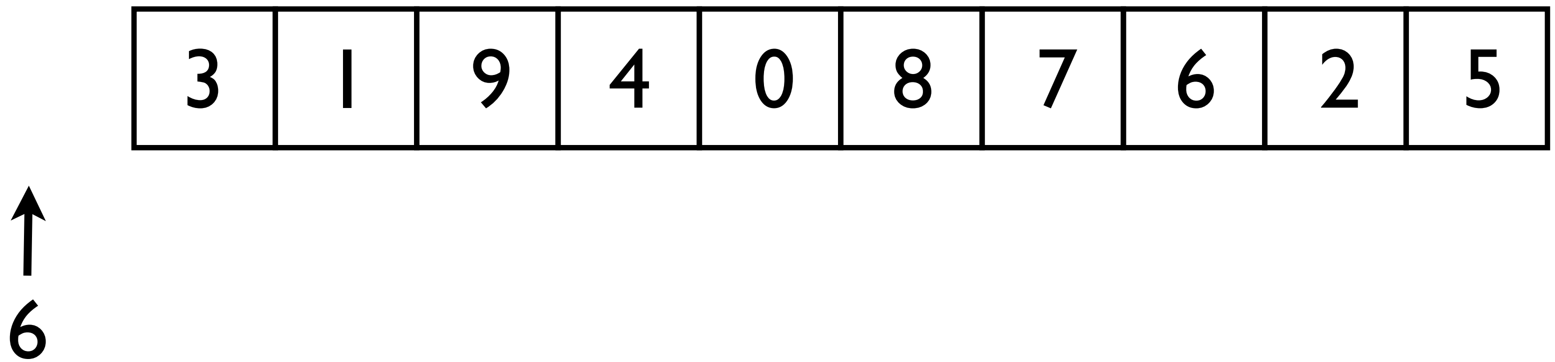
> Big-Oh notation

> Sorting a given list

- Selection sort
- Bubble sort

Motivating example: searching a list

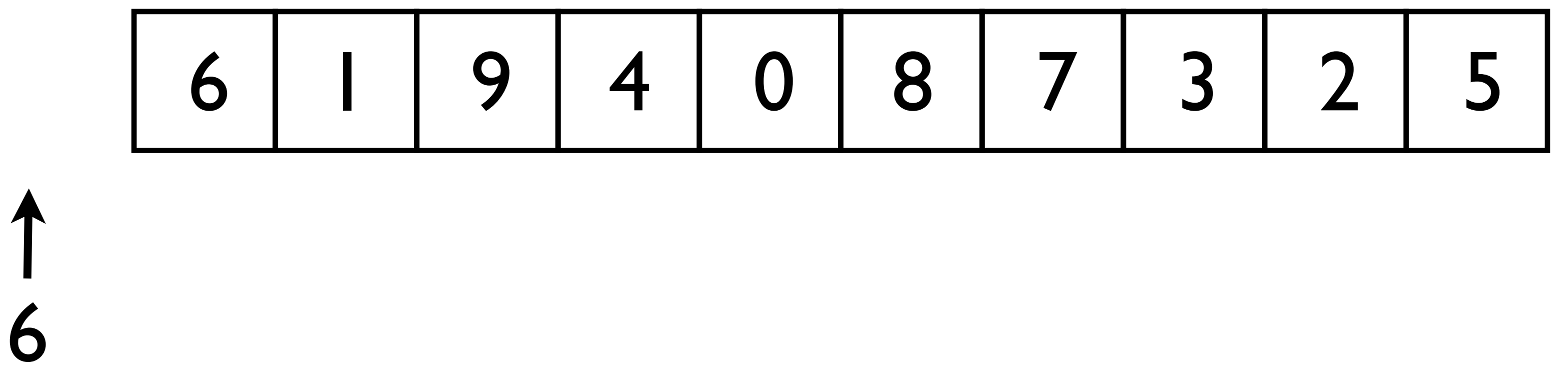
Given a list of integers, and an integer, determine if the integer is in the list.



How many steps in the algorithm?

Motivating example: searching a list

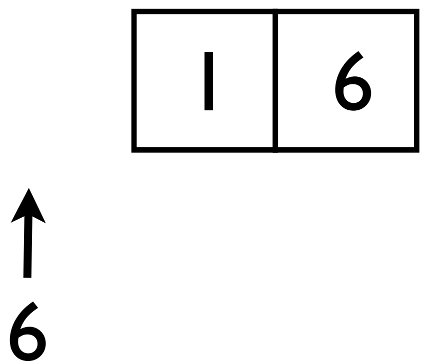
Given a list of integers, and an integer, determine if the integer is in the list.



How many steps in the algorithm?

Motivating example: searching a list

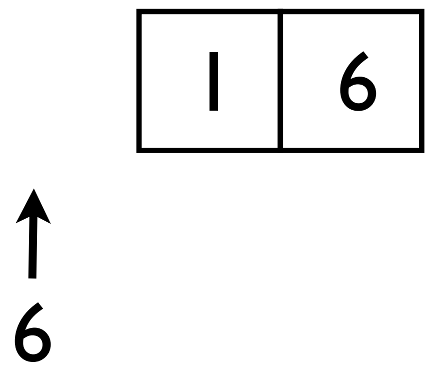
Given a list of integers, and an integer, determine if the integer is in the list.



How many steps in the algorithm?

Motivating example: searching a list

Given a list of integers, and an integer, determine if the integer is in the list.



How many steps in the algorithm?

running time of an algorithm depends on:

- size of input (e.g., size of the list)
- the values in the input

Measuring running time

running time of an algorithm depends on:

- size of input (e.g., size of the list)
- the values in the input

Measuring running time

running time of an algorithm depends on:

- size of input (e.g., size of the list)
- the values in the input

size of the list:

Want to know running time with respect to any list size.

N = list size

Measure running time as a function of N.

Measuring running time

running time of an algorithm depends on:

- size of the list (size of input)
- the values in the input

the values in the input:

Measure running time with respect to *worst input*.

worst input = input that leads to most number of steps

Measuring running time

How to properly measure running time

> Input length/size denoted by N (and sometimes by n)

- for lists: N = number of elements
- for strings: N = number of characters
- for ints: N = number of digits

> Running time is a function of N.

> Look at worst-case scenario/input of length N.

> Count algorithmic steps.

> Ignore constant factors. (e.g. $N^2 \approx 3N^2$)
(use Big-Oh notation)

The Plan

> How to properly measure running time

> Searching a given list

- Linear search
- Binary search

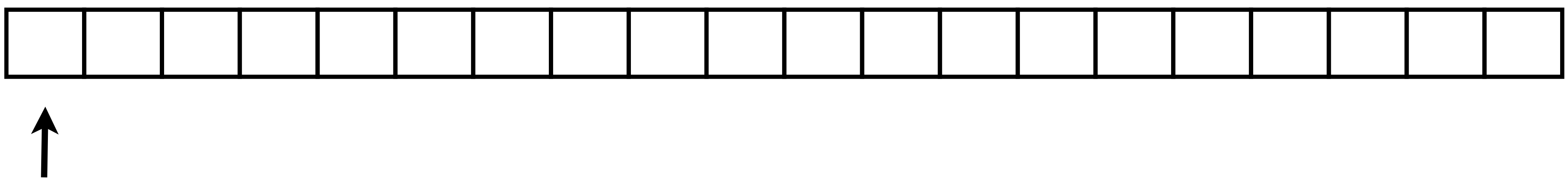
> Big-Oh notation

> Sorting a given list

- Selection sort
- Bubble sort

Searching for an element in a list

Given a list of integers, and an integer, determine if the integer is in the list.



How many steps does this take?

N steps

Can't do better (in the worst case)

This algorithm is called **Linear Search**.

Searching for an element in a sorted list

Given a sorted list of integers, and an integer, determine if the integer is in the list.

0	1	2	4	5	5	6	8	9	9	50	60	99

↑
50

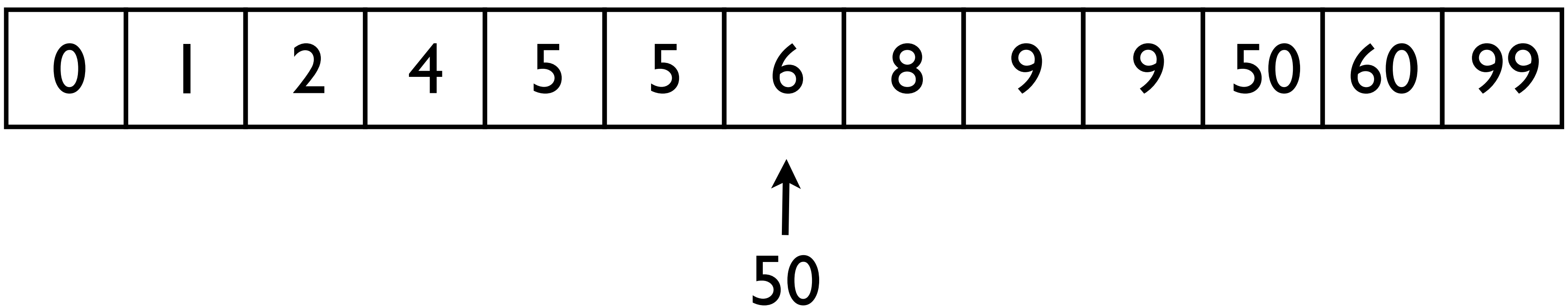
running time: *N* steps

Can we do better?

How would you search for a name in a phonebook?

Searching for an element in a sorted list

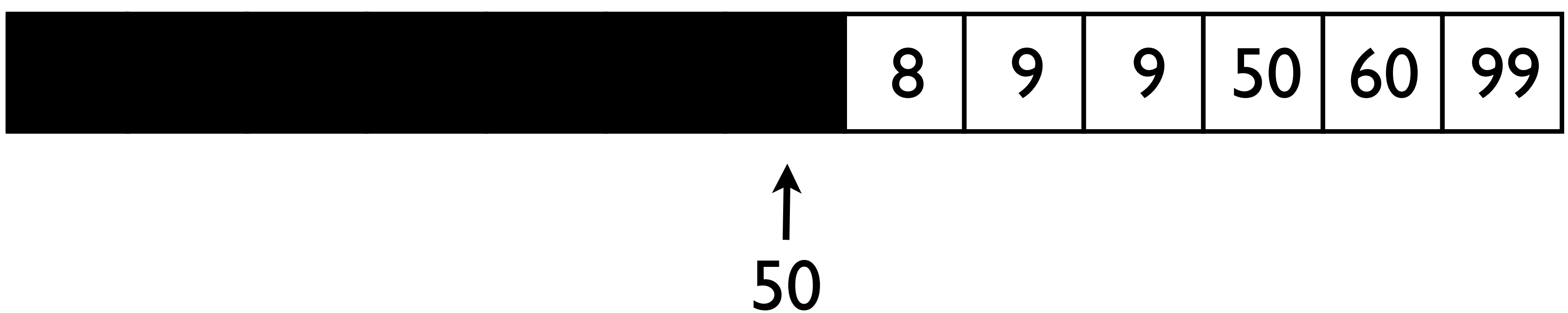
Binary Search



Start in the middle

Searching for an element in a sorted list

Binary Search



Start in the middle

If (element > middle),

    can ignore <u>left</u> half of the list.

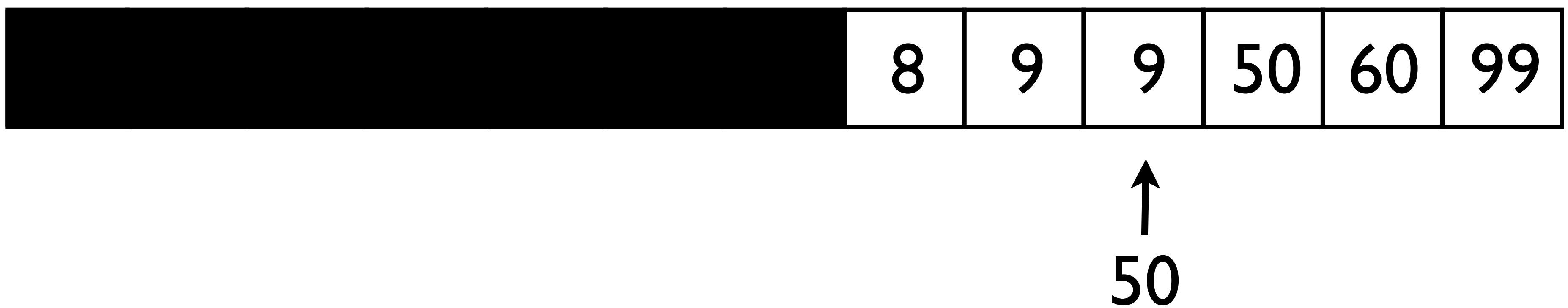
If (element < middle),

    can ignore <u>right</u> half of the list.

Repeat process on the remaining half.

Searching for an element in a sorted list

Binary Search



Start in the middle

If (element > middle),

can ignore left half of the list.

If (element < middle),

can ignore right half of the list.

Repeat process on the remaining half.

Searching for an element in a sorted list

Binary Search

Start in the middle

If (element > middle),

can ignore <u>left</u> half of the list.

If (element < middle),

can ignore <u>right</u> half of the list.

Repeat process on the remaining half.

Searching for an element in a sorted list

Binary Search

Start in the middle

If (element > middle),

can ignore <u>left</u> half of the list.

If (element < middle),

can ignore <u>right</u> half of the list.

Repeat process on the remaining half.

Searching for an element in a sorted list

Binary Search

Start in the middle

If (element > middle),

can ignore <u>left</u> half of the list.

If (element < middle),

can ignore <u>right</u> half of the list.

Repeat process on the remaining half.

Searching for an element in a sorted list

Binary Search

How many steps does this take (in the worst case)?

$$\sim \log_2 N$$

At each step we halve the list.

$$N \to \frac{N}{2} \to \frac{N}{4} \to \frac{N}{8} \to \cdots \to 1$$

After k steps: $\frac{N}{2^k}$ elements left. When is this 1?

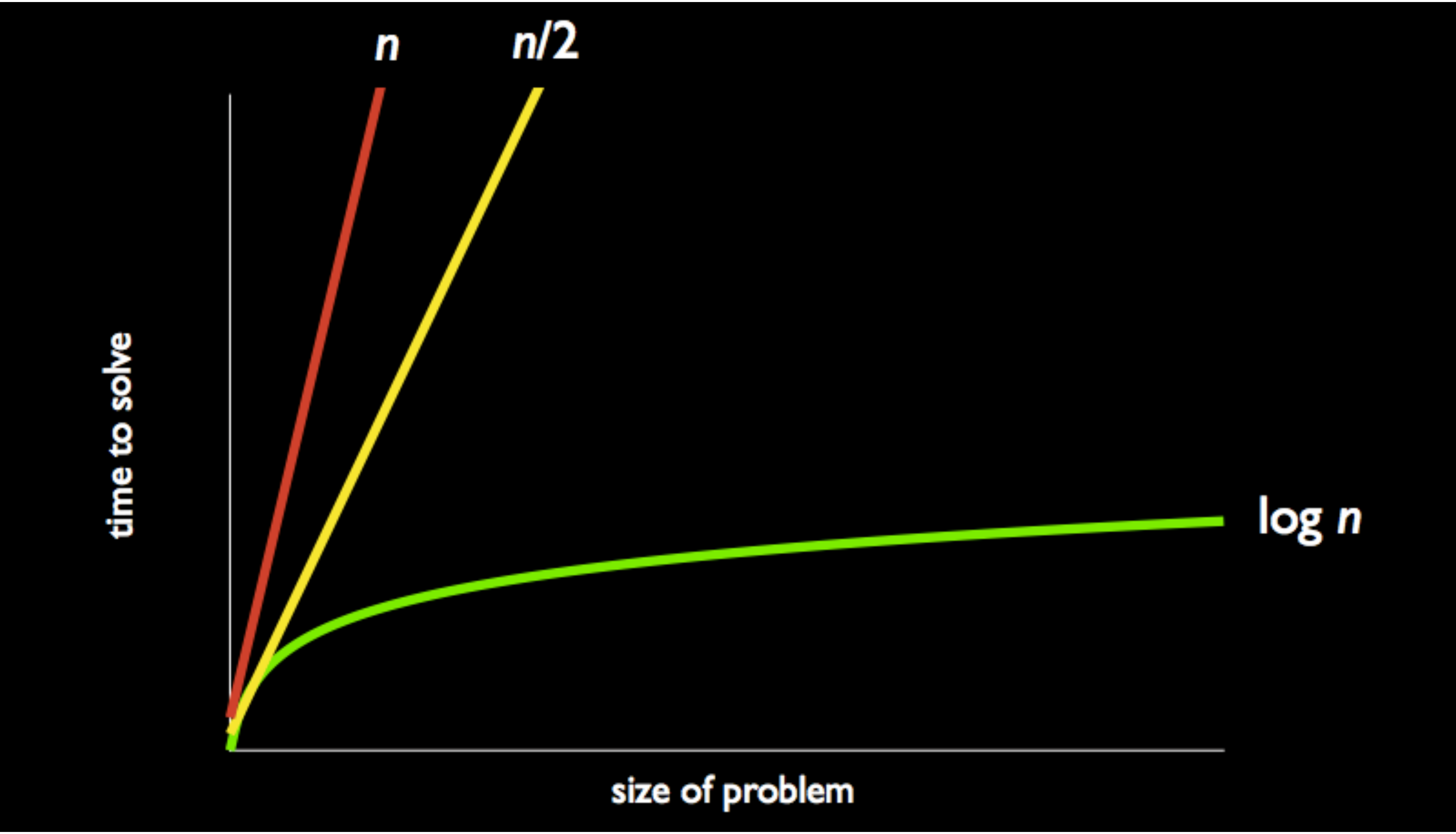
N vs log N

How much better is log N compared to N ?

N	log N
2	1
8	3
128	7
1024	10
1,048,576	20
1,073,741,824	30
1,152,921,504,606,846,976	60

~ 1 quintillion

n vs log n



Linear search vs Binary search

Linear Search

Takes $\sim N$ steps.

Works for both sorted and unsorted lists.

Binary Search

Takes $\sim \log_2 N$ steps.

Works for only sorted lists.

Linear search code

```
def linearSearch(L, target):
    for index in range(len(L)):
        if(L[index] == target):
            return True
    return False
```

How many steps in the worst case?

Binary search code

```
def binarySearch(L, target):
    start = 0
    end = len(L) - 1
    while(start <= end):
        middle = (start + end)//2
        if(L[middle] == target):
            return True
        elif(L[middle] > target):
            end = middle-1
        else:
            start = middle+1
    return False
```

How many steps in the worst case?

The Plan

> How to properly measure running time

> Searching a given list

- Linear search

- Binary search

> Big-Oh notation

> Sorting a given list

- Selection sort

- Bubble sort

The CS way to compare functions:

$$O(\cdot)$$

$$\leq$$

$$f(n) = O(g(n)) \qquad \equiv \qquad f(n) \text{ is } O(g(n))$$

means $f(n) \leq g(n)$, ignoring constant factors and small values of n

The CS way to compare functions:

$$O(\cdot)$$

$$\leq$$

$$10n + 25 = O(n) \qquad \equiv \qquad 10n + 25 \text{ is } O(n)$$

means $10n + 25 \leq n$, ignoring constant factors and small values of n

Big Oh Notation

A notation to ignore constant factors and small n.

$2n$ is $O(n)$

$3n$ is $O(n)$

$1000n$ is $O(n)$

$0.0000001n$ is $O(n)$

n is $O(n^2)$

$0.0000001n^2$ is <u>not</u> $O(n)$

$2\log_2 n$ is $O(\log n)$

$3\log_2 n$ is $O(\log n)$

$1000\log_2 n$ is $O(\log n)$

$0.0000001\log_2 n$ is $O(\log n)$

$\log_9 n$ is $O(\log n)$

$n\log_7 n + 100$ is <u>not</u> $O(n)$

Running time of linear search is $O(N)$

Running time of binary search is $O(\log N)$

Big Oh Notation

Why ignore constant factors and small n?

- We want to capture the essence of an algorithm/problem.

- Technology independent. Language independent.

- Difference in Big Oh → a really fundamental difference.

Big Oh Notation

Ignoring constant factors means ignoring lower order additive terms.

$$n^2 + 100n + 500 \text{ is } O(n^2)$$

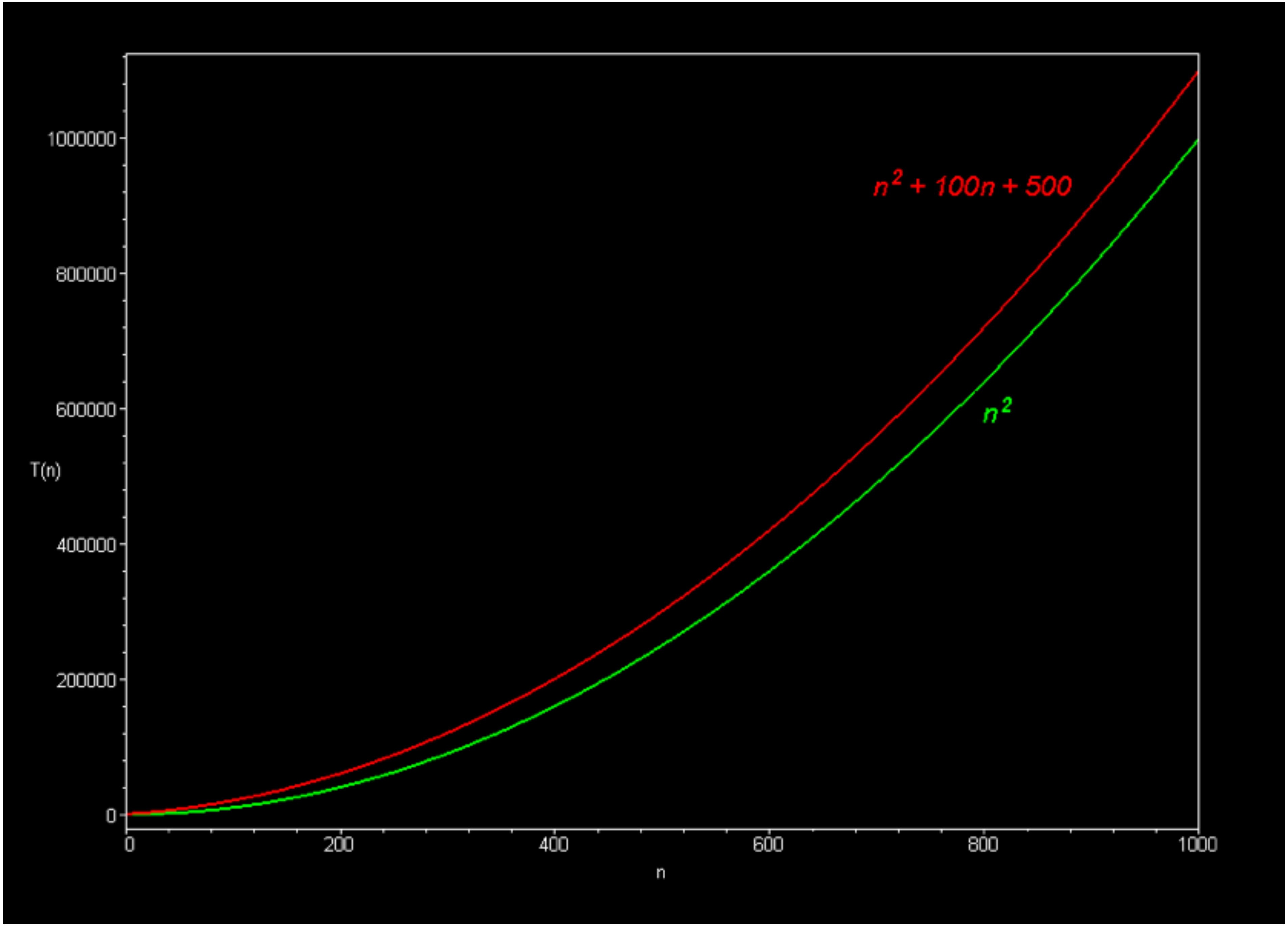
$$601n^2 = n^2 + 100n^2 + 500n^2 > n^2 + 100n + 500$$

Also:

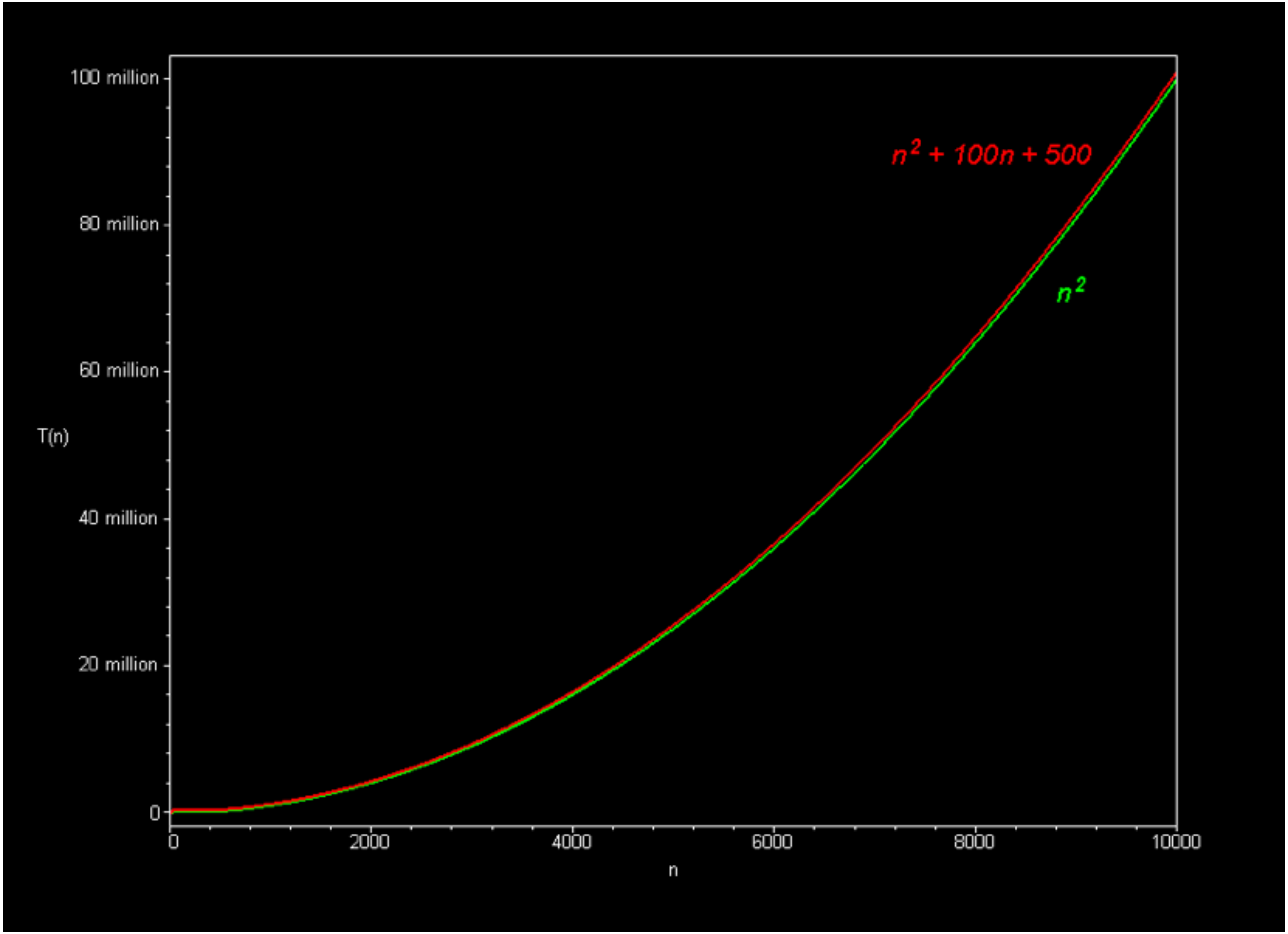
$$\frac{n^2 + 100n + 500}{n^2} = 1 + \frac{100n}{n^2} + \frac{500}{n^2} \quad \longrightarrow 1$$

Lower order terms don't matter!

Big Oh Notation



Big Oh Notation



Important Big Oh Classes

Again, not much interested in the difference between n and $n/2$.

We are **very** interested in the differences between

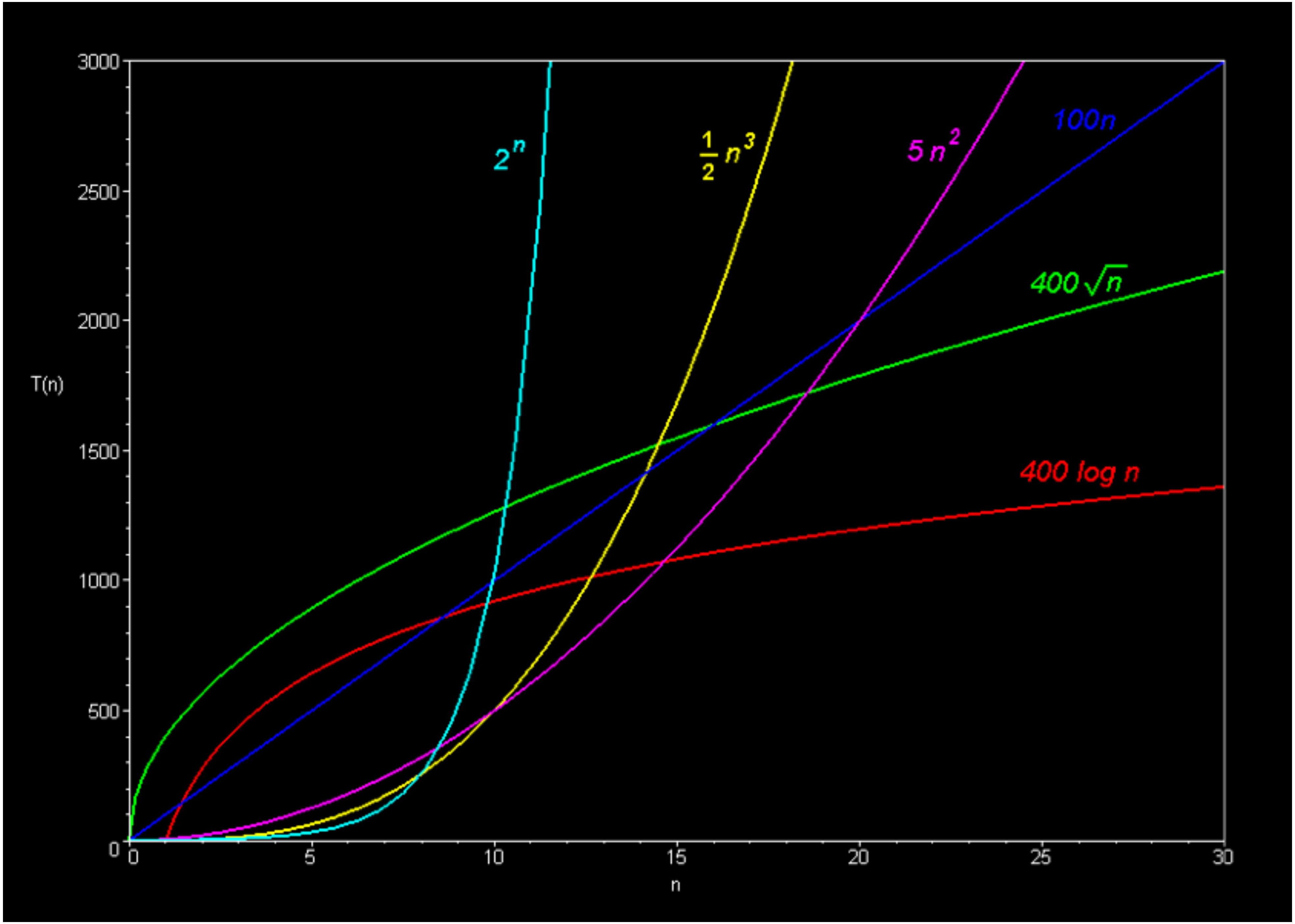
$$\log n \;<<<\; \sqrt{n} \;<<\; n \;<<\; n^2 \;<<\; n^3 \;<<<\; 2^n$$

Important Big Oh Classes

Common function families:

Constant:	$O(1)$
Logarithmic:	$O(\log n)$
Square-root:	$O(\sqrt{n}) = O(n^{0.5})$
Linear:	$O(n)$
Loglinear:	$O(n \log n)$
Quadratic:	$O(n^2)$
Exponential:	$O(k^n)$

Important Big Oh Classes



Exponential running time

If your algorithm has exponential running time
e.g. $\sim 2^n$

No hope of being practical.

n vs 2^n

2^n	n
2	1
8	3
128	7
1024	10
1,048,576	20
1,073,741,824	30
1,152,921,504,606,846,976	60

Exponential running time example

Given a list of integers, determine if there is a subset of the integers that sum to 0.

4	-3	-2	7	99	5	1

Exponential running time example

Given a list of integers, determine if there is a subset of the integers that sum to 0.

4	-3	-2	7	99	5	1

Exhaustive Search

Try every possible subset and see if it sums to 0.

Number of subsets is 2^N

So running time is at least 2^N

The Plan

> How to properly measure running time

> Searching a given list

- Linear search
- Binary search

> Big-Oh notation

> Sorting a given list

- Selection sort
- Bubble sort

1. Algorithm
2. Running time
3. Code

Selection Sort: Algorithm

Sort a given list of integers (from small to large).

4	8	2	7	99	5	0

Selection Sort

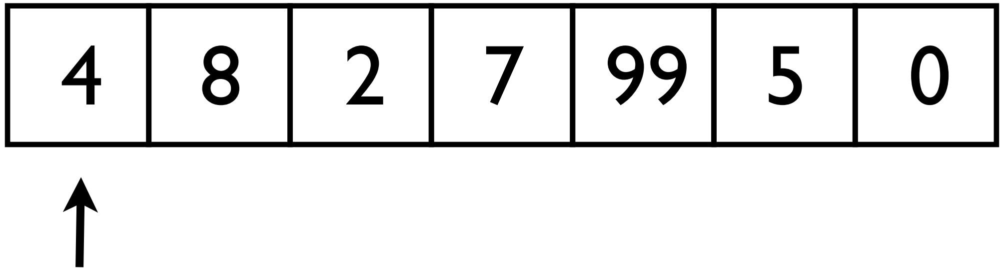
Find the minimum element.

Put it on the left.

Repeat process on the remaining n-1 elements.

Selection Sort: Algorithm

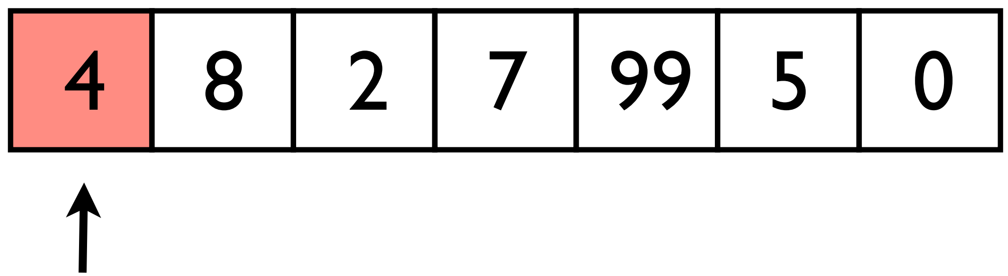
Sort a given list of integers (from small to large).



Selection Sort

Selection Sort: Algorithm

Sort a given list of integers (from small to large).

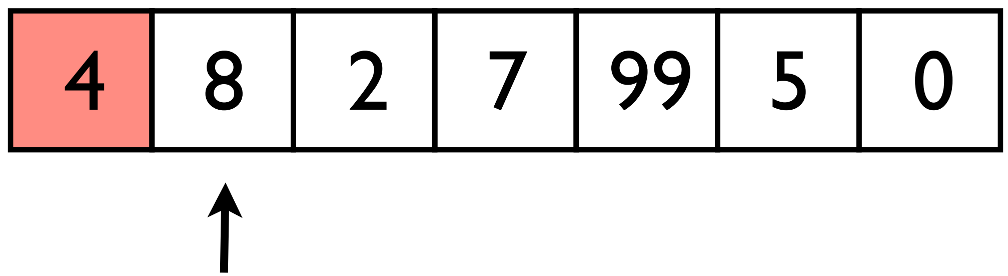


Selection Sort

Current min: 4

Selection Sort: Algorithm

Sort a given list of integers (from small to large).

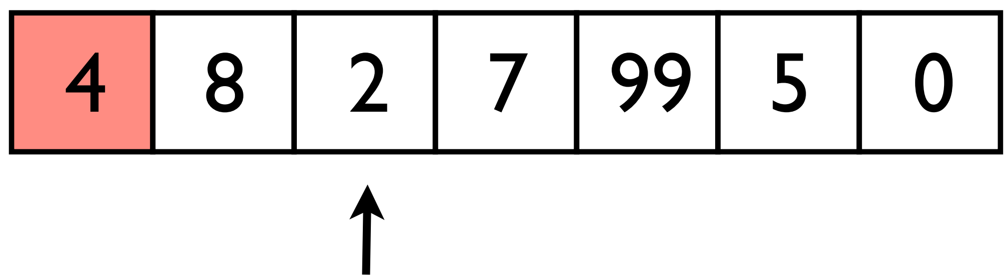


Selection Sort

Current min: 4

Selection Sort: Algorithm

Sort a given list of integers (from small to large).

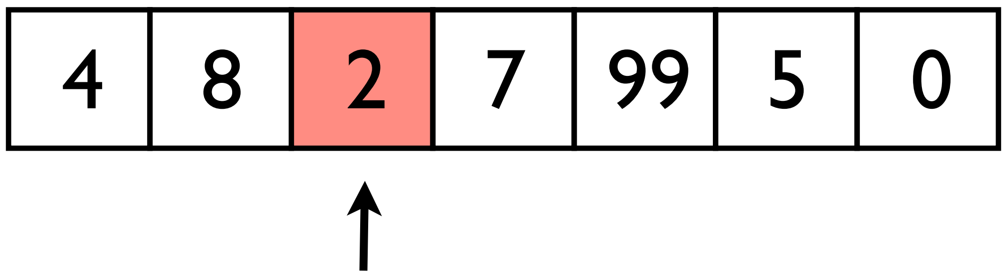


Selection Sort

Current min: 4

Selection Sort: Algorithm

Sort a given list of integers (from small to large).

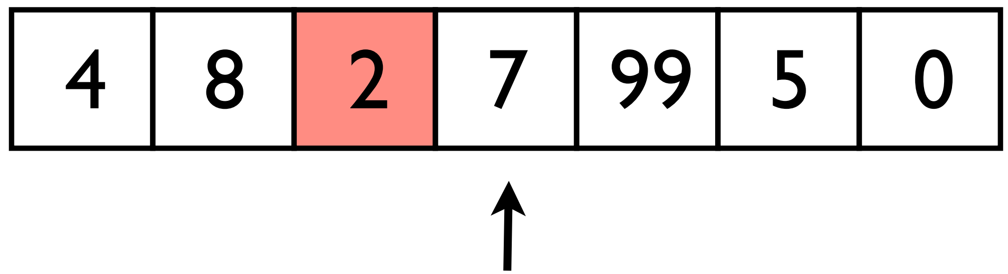


Selection Sort

Current min: 2

Selection Sort: Algorithm

Sort a given list of integers (from small to large).

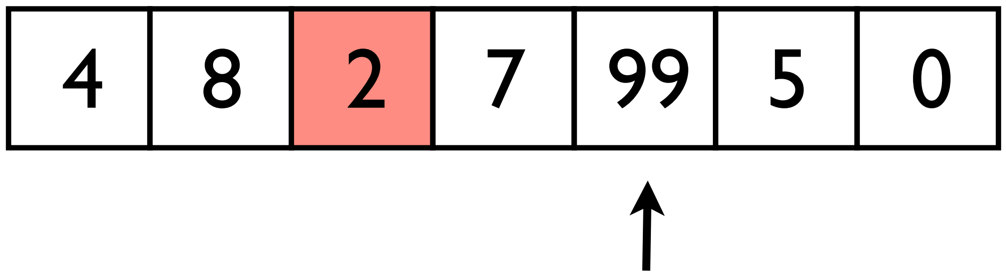


Selection Sort

Current min: 2

Selection Sort: Algorithm

Sort a given list of integers (from small to large).

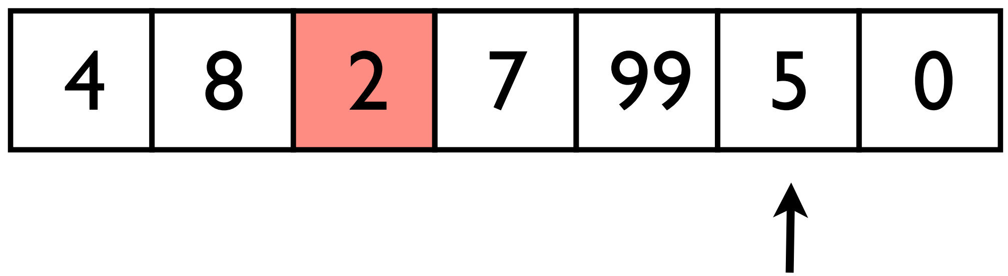


Selection Sort

Current min: 2

Selection Sort: Algorithm

Sort a given list of integers (from small to large).

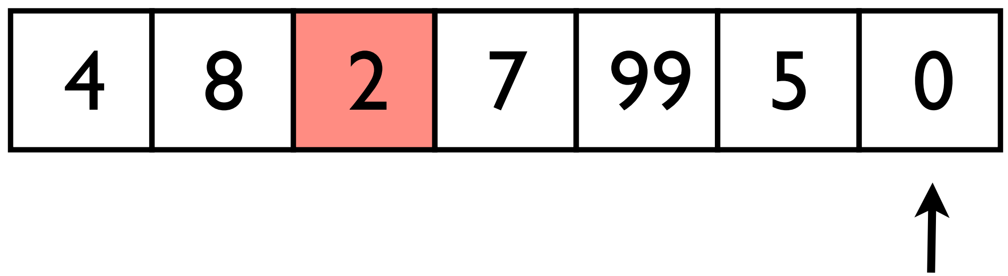


Selection Sort

Current min: 2

Selection Sort: Algorithm

Sort a given list of integers (from small to large).

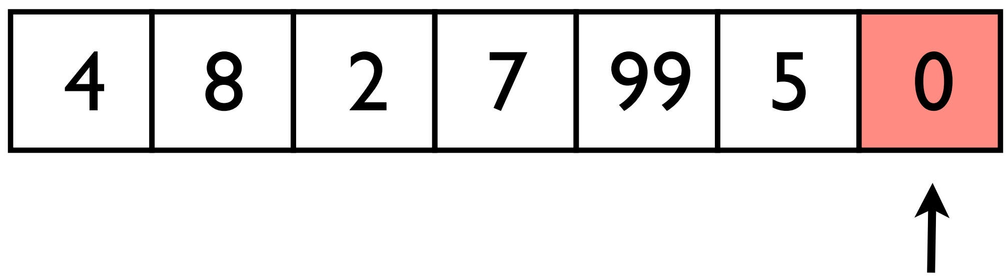


Selection Sort

Current min: 2

Selection Sort: Algorithm

Sort a given list of integers (from small to large).

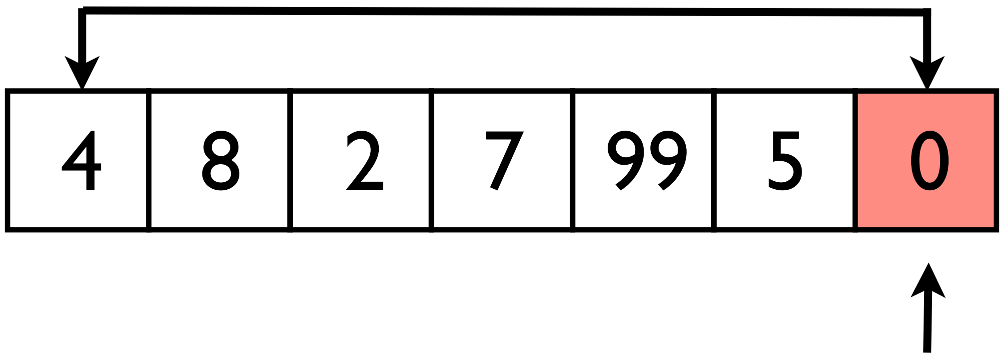


Selection Sort

Current min: 0

Selection Sort: Algorithm

Sort a given list of integers (from small to large).

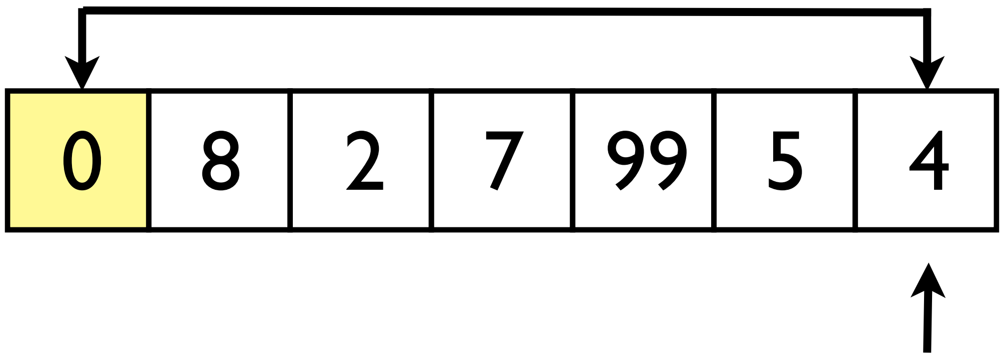


Selection Sort

Swap current min with first element of the array

Selection Sort: Algorithm

Sort a given list of integers (from small to large).



Selection Sort

Swap current min with first element of the array

Selection Sort: Algorithm

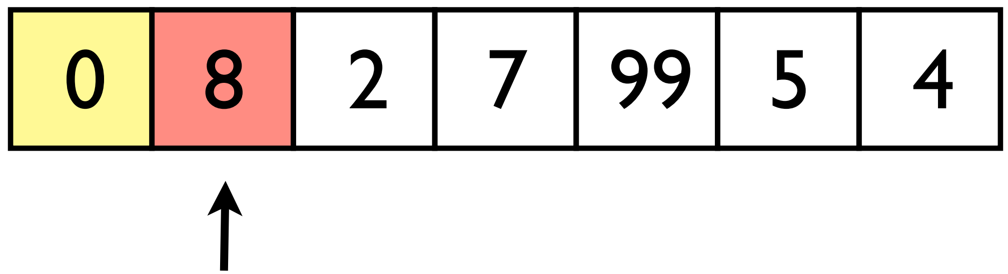
Sort a given list of integers (from small to large).

0	8	2	7	99	5	4

Selection Sort

Selection Sort: Algorithm

Sort a given list of integers (from small to large).

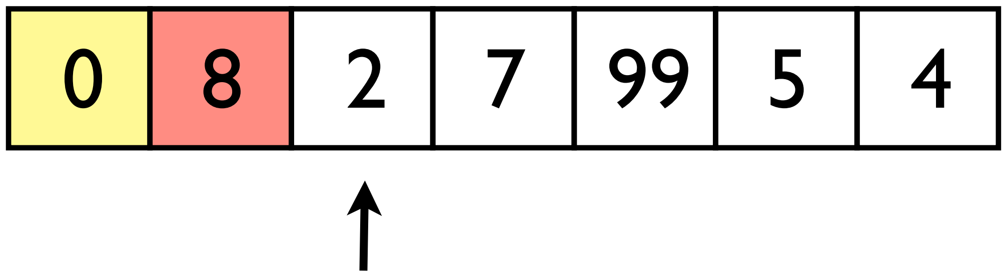


Selection Sort

Current min: 8

Selection Sort: Algorithm

Sort a given list of integers (from small to large).

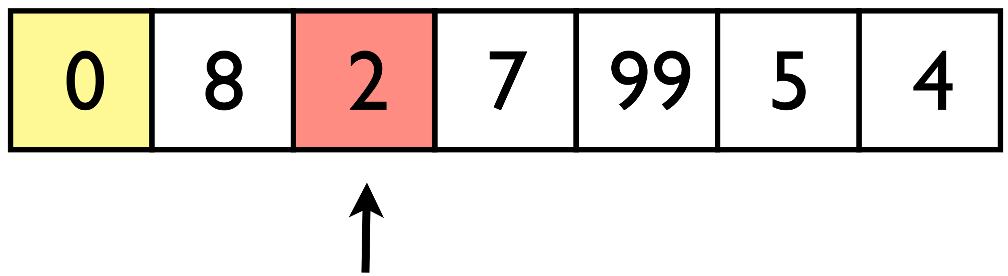


Selection Sort

Current min: 8

Selection Sort: Algorithm

Sort a given list of integers (from small to large).

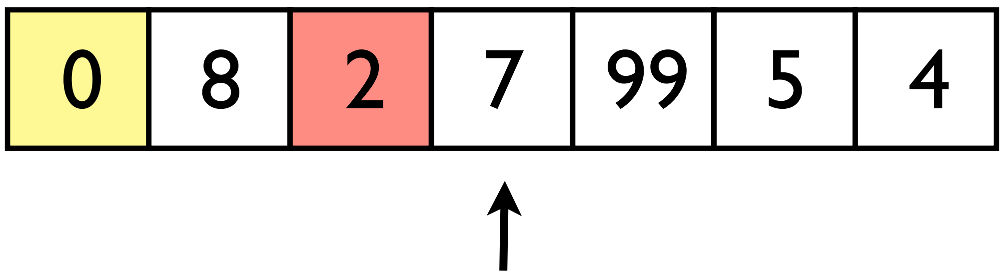


Selection Sort

Current min: 2

Selection Sort: Algorithm

Sort a given list of integers (from small to large).

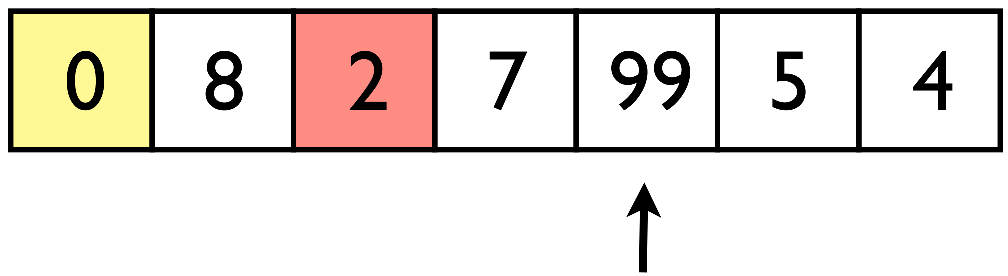


Selection Sort

Current min: 2

Selection Sort: Algorithm

Sort a given list of integers (from small to large).

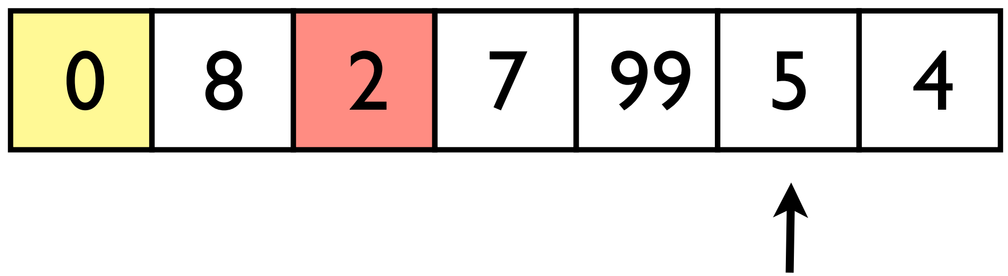


Selection Sort

Current min: 2

Selection Sort: Algorithm

Sort a given list of integers (from small to large).

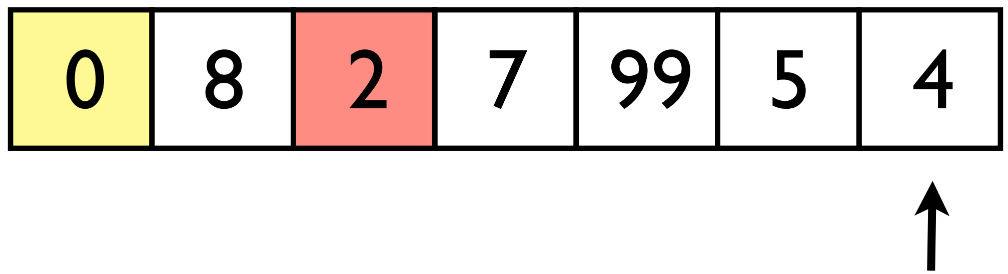


Selection Sort

Current min: 2

Selection Sort: Algorithm

Sort a given list of integers (from small to large).

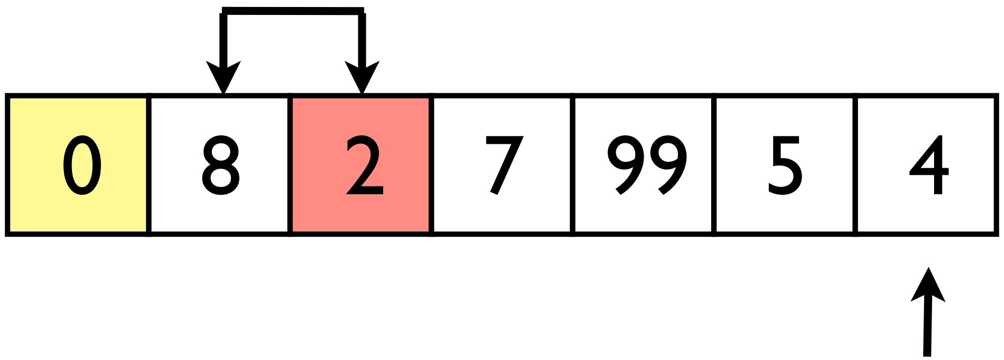


Selection Sort

Current min: 2

Selection Sort: Algorithm

Sort a given list of integers (from small to large).

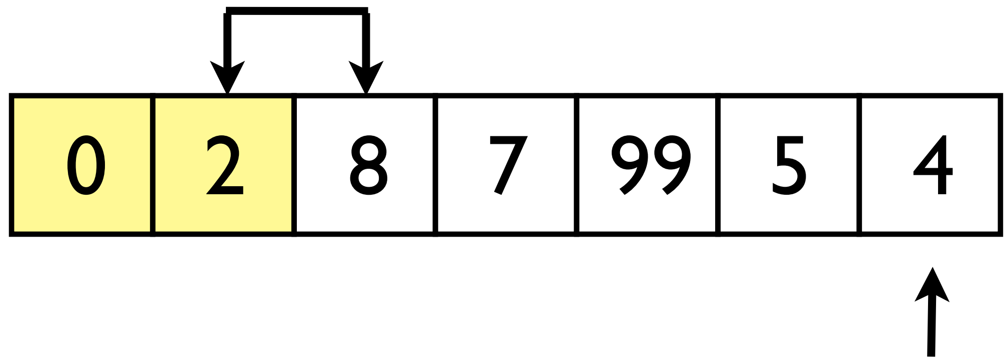


Selection Sort

Swap current min with first element of unsorted part

Selection Sort: Algorithm

Sort a given list of integers (from small to large).

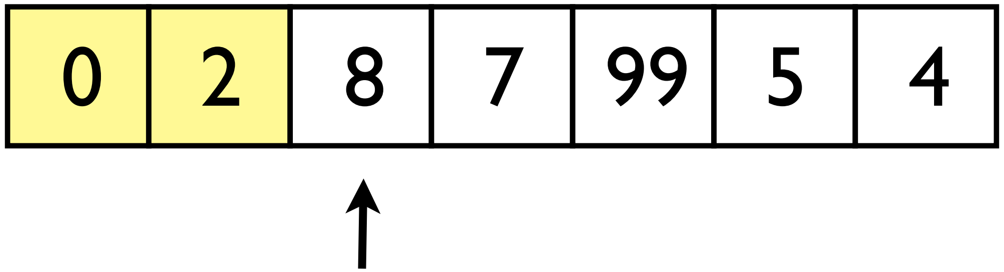


Selection Sort

Swap current min with first element of unsorted part

Selection Sort: Algorithm

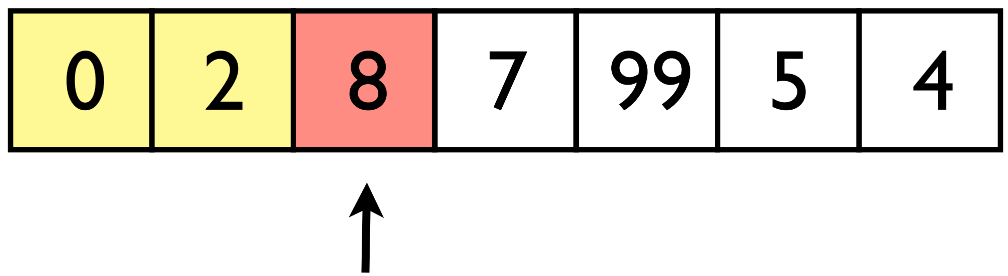
Sort a given list of integers (from small to large).



Selection Sort

Selection Sort: Algorithm

Sort a given list of integers (from small to large).

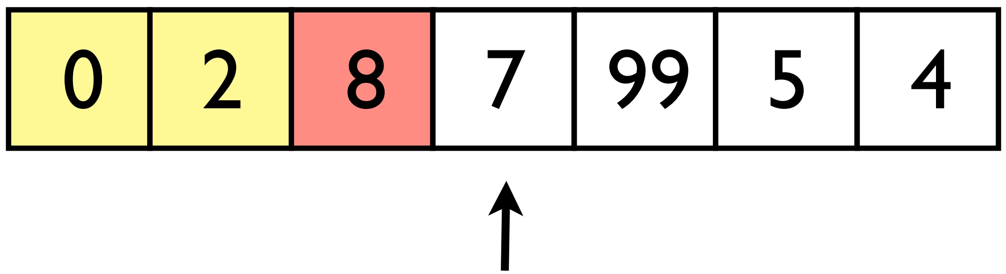


Selection Sort

Current min: 8

Selection Sort: Algorithm

Sort a given list of integers (from small to large).

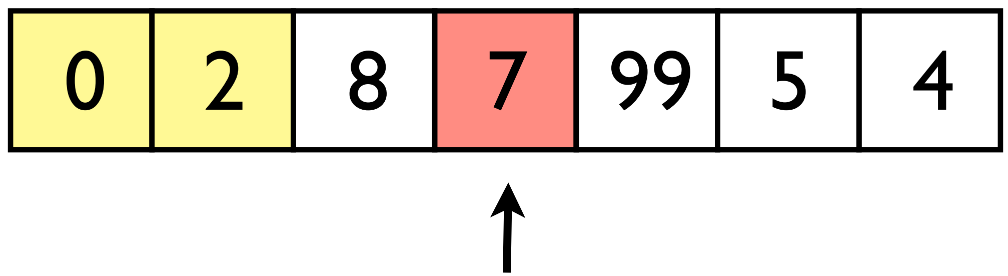


Selection Sort

Current min: 8

Selection Sort: Algorithm

Sort a given list of integers (from small to large).

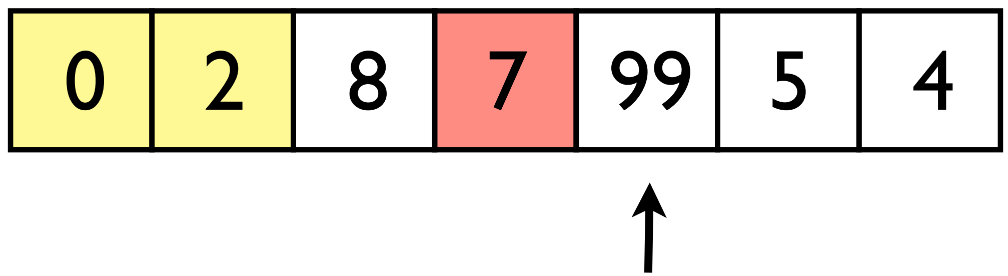


Selection Sort

Current min: 7

Selection Sort: Algorithm

Sort a given list of integers (from small to large).

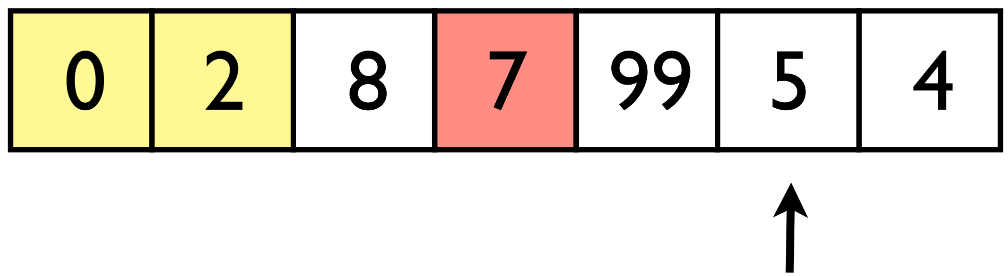


Selection Sort

Current min: 7

Selection Sort: Algorithm

Sort a given list of integers (from small to large).

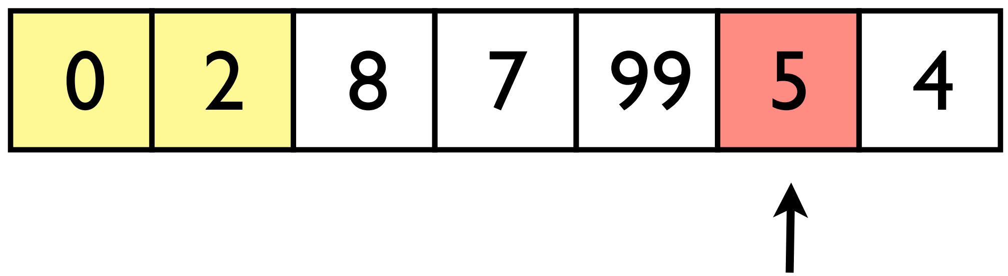


Selection Sort

Current min: 7

Selection Sort: Algorithm

Sort a given list of integers (from small to large).

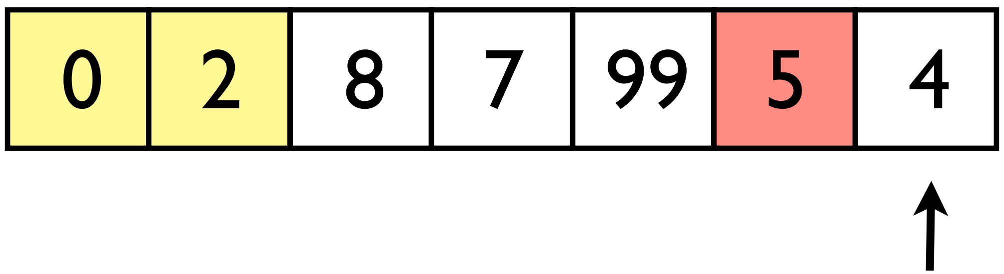


Selection Sort

Current min: 5

Selection Sort: Algorithm

Sort a given list of integers (from small to large).



Selection Sort

Current min: 5

Selection Sort: Algorithm

Sort a given list of integers (from small to large).

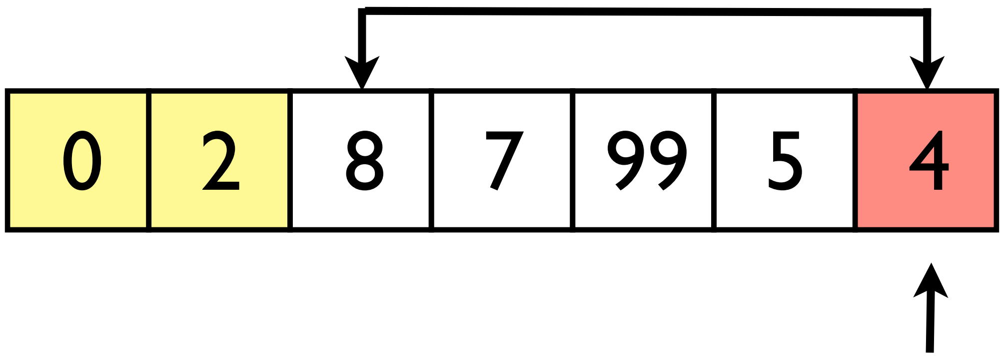
0	2	8	7	99	5	4

Selection Sort

Current min: 4

Selection Sort: Algorithm

Sort a given list of integers (from small to large).

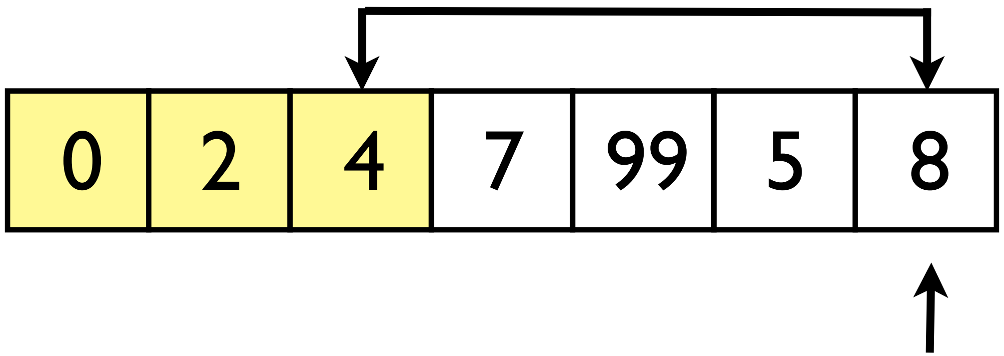


Selection Sort

Swap current min with first element of unsorted part

Selection Sort: Algorithm

Sort a given list of integers (from small to large).

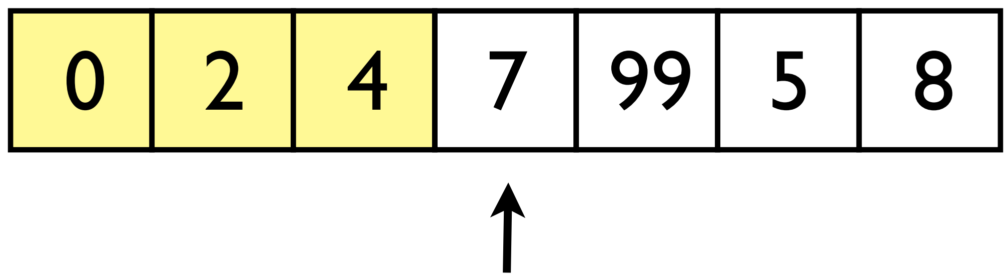


Selection Sort

Swap current min with first element of unsorted part

Selection Sort: Algorithm

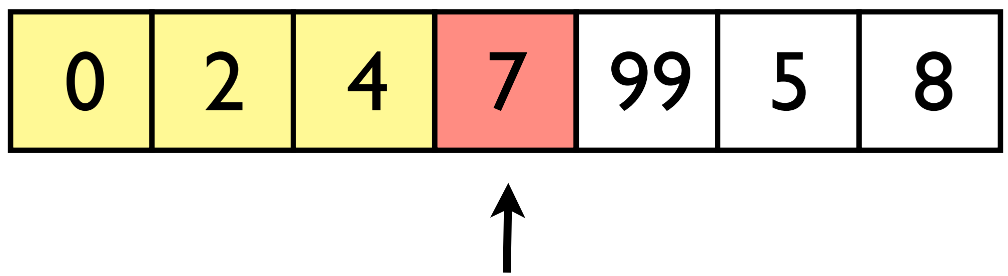
Sort a given list of integers (from small to large).



Selection Sort

Selection Sort: Algorithm

Sort a given list of integers (from small to large).

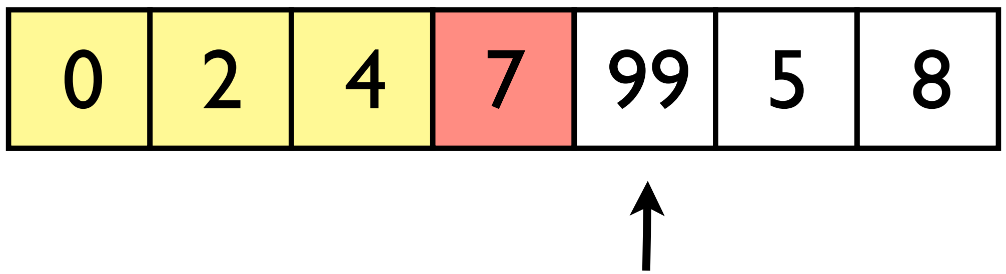


Selection Sort

Current min: 7

Selection Sort: Algorithm

Sort a given list of integers (from small to large).

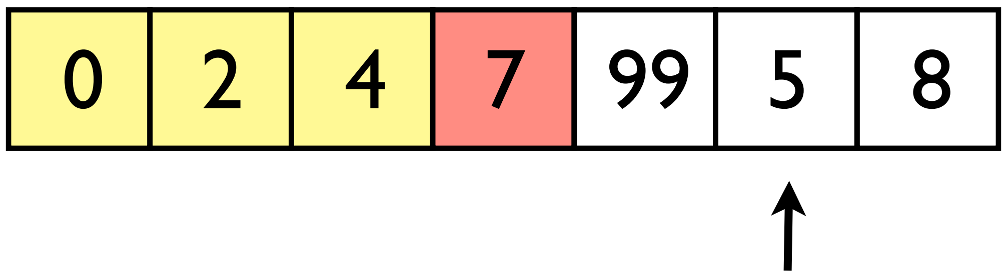


Selection Sort

Current min: 7

Selection Sort: Algorithm

Sort a given list of integers (from small to large).

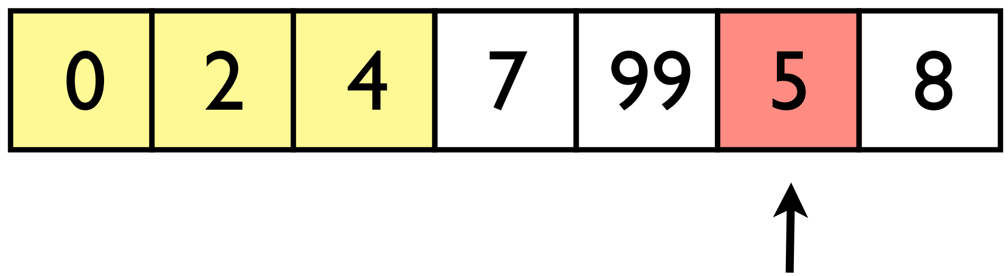


Selection Sort

Current min: 7

Selection Sort: Algorithm

Sort a given list of integers (from small to large).

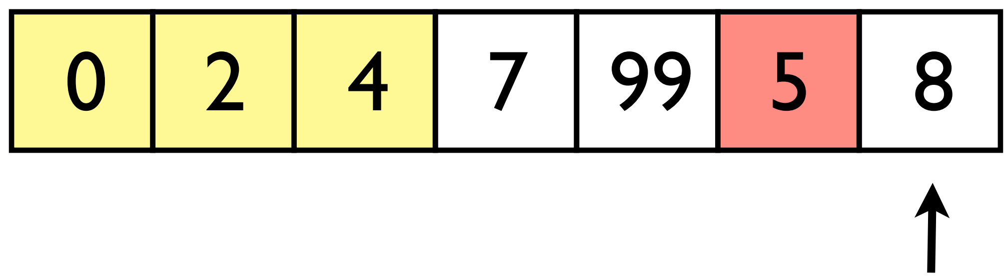


Selection Sort

Current min: 5

Selection Sort: Algorithm

Sort a given list of integers (from small to large).

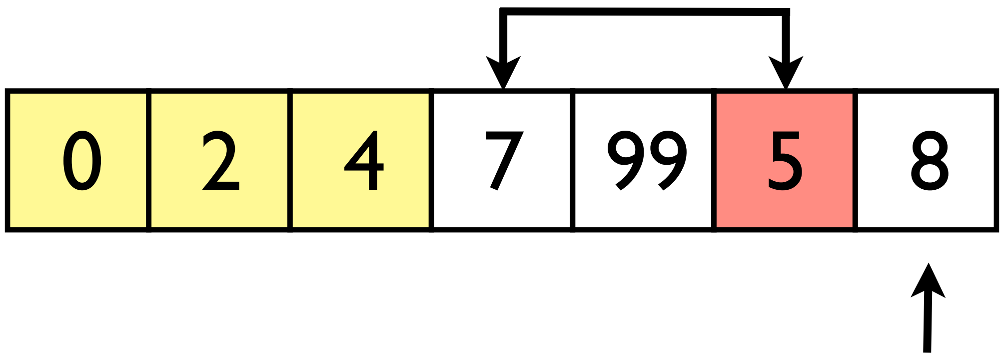


Selection Sort

Current min: 5

Selection Sort: Algorithm

Sort a given list of integers (from small to large).

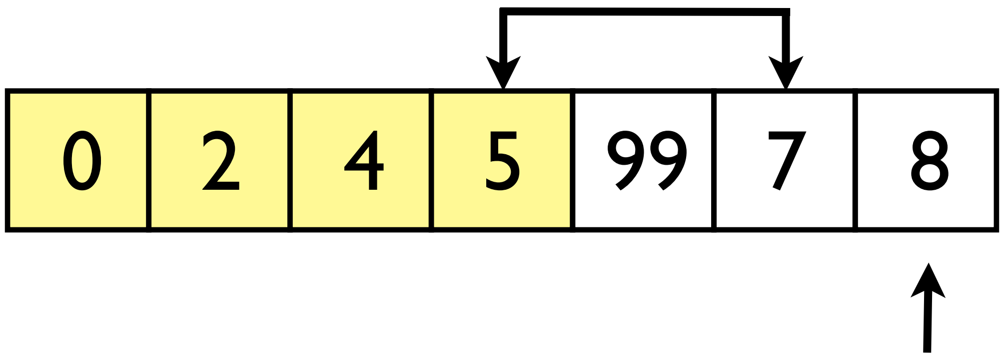


Selection Sort

Swap current min with first element of unsorted part

Selection Sort: Algorithm

Sort a given list of integers (from small to large).

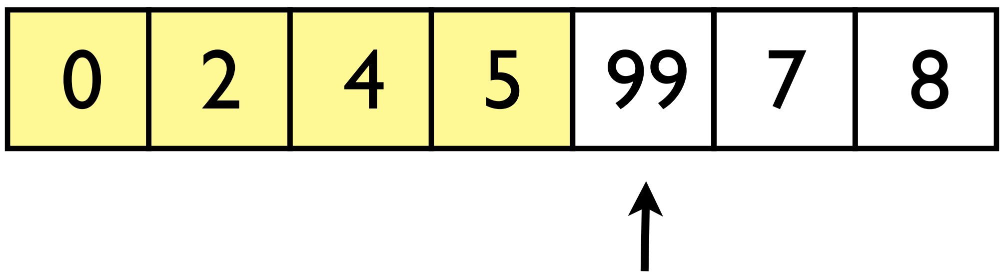


Selection Sort

Swap current min with first element of unsorted part

Selection Sort: Algorithm

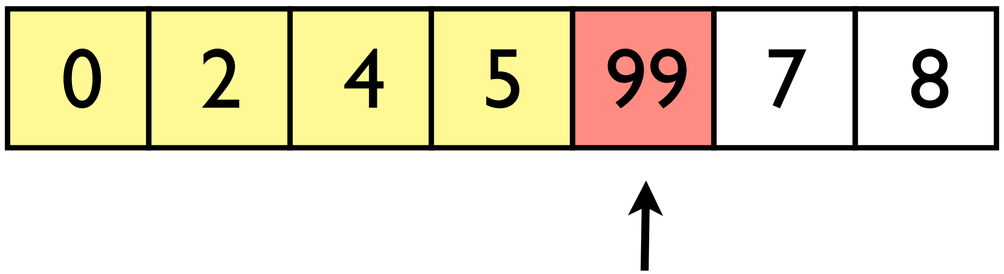
Sort a given list of integers (from small to large).



Selection Sort

Selection Sort: Algorithm

Sort a given list of integers (from small to large).

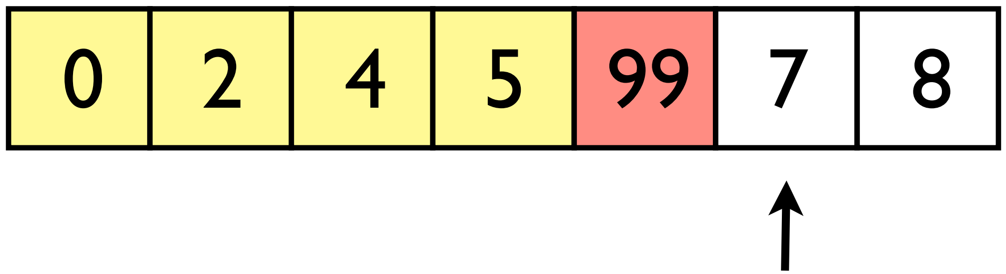


Selection Sort

Current min: 99

Selection Sort: Algorithm

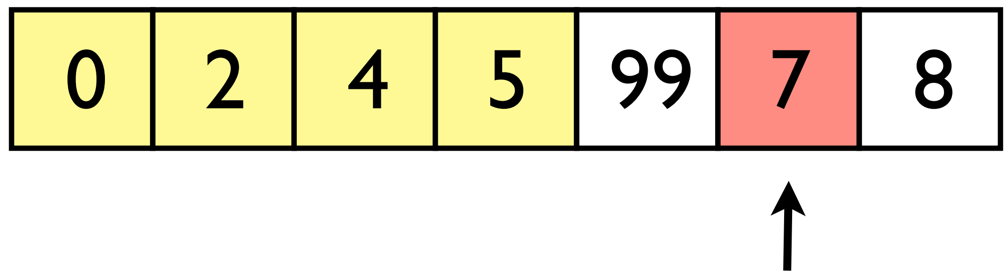
Sort a given list of integers (from small to large).



Selection Sort

Current min: 99

Sort a given list of integers (from small to large).

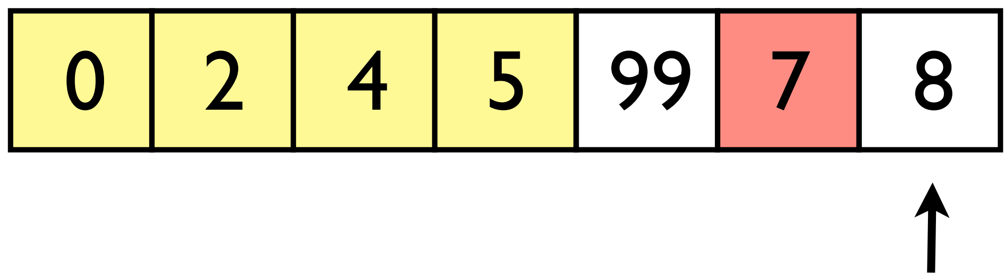


Selection Sort

Current min: 7

Selection Sort: Algorithm

Sort a given list of integers (from small to large).

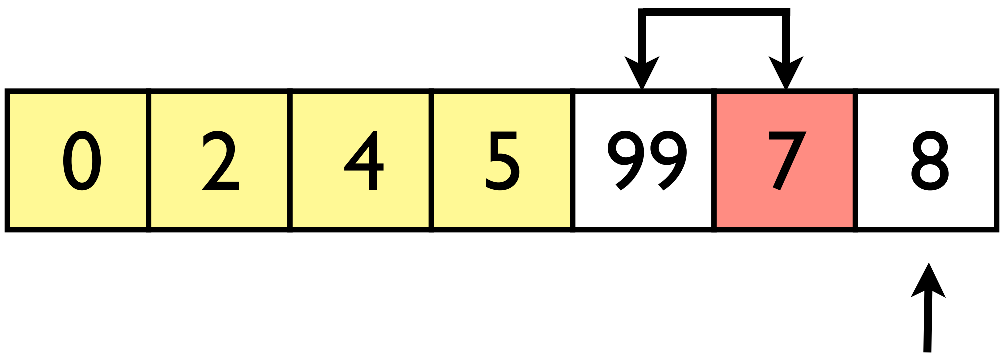


Selection Sort

Current min: 7

Selection Sort: Algorithm

Sort a given list of integers (from small to large).

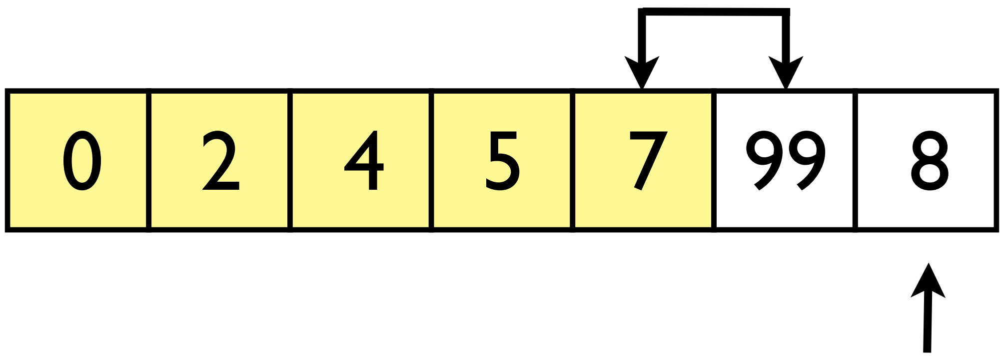


Selection Sort

Swap current min with first element of unsorted part

Selection Sort: Algorithm

Sort a given list of integers (from small to large).

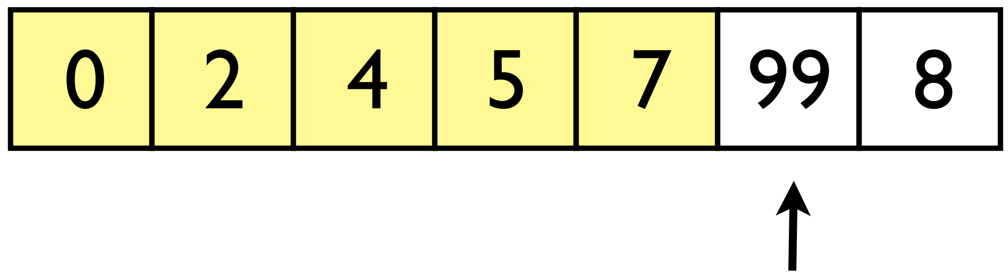


Selection Sort

Swap current min with first element of unsorted part

Selection Sort: Algorithm

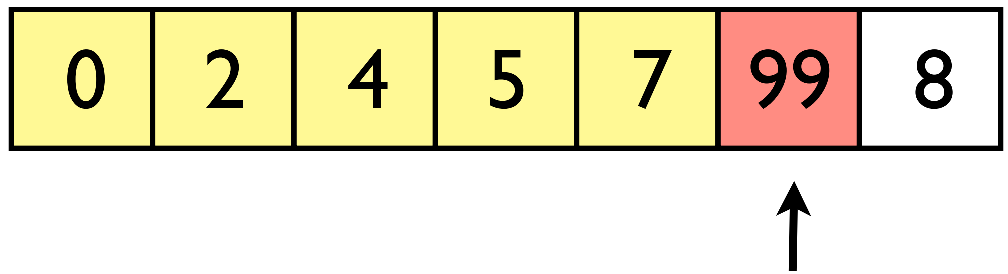
Sort a given list of integers (from small to large).



Selection Sort

Selection Sort: Algorithm

Sort a given list of integers (from small to large).

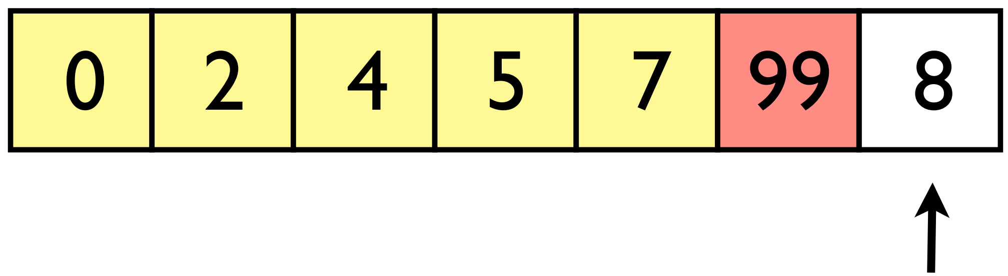


Selection Sort

Current min: 99

Selection Sort: Algorithm

Sort a given list of integers (from small to large).

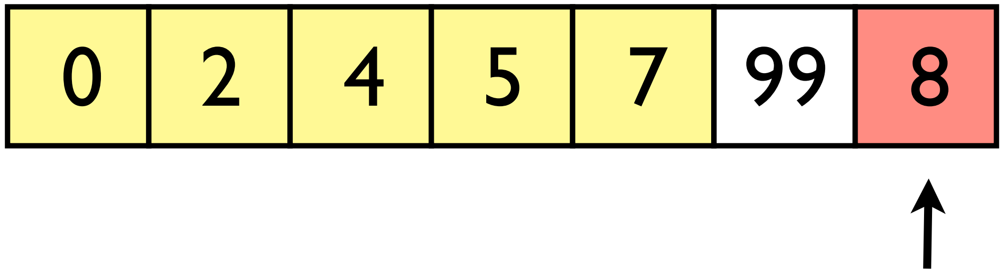


Selection Sort

Current min: 99

Selection Sort: Algorithm

Sort a given list of integers (from small to large).

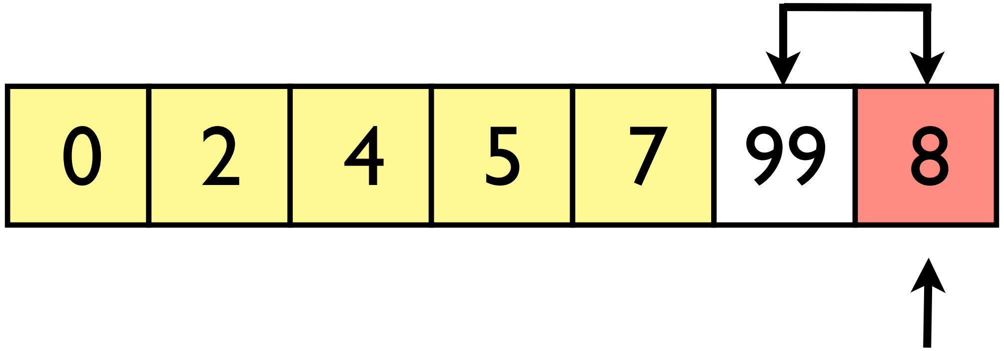


Selection Sort

Current min: 8

Selection Sort: Algorithm

Sort a given list of integers (from small to large).

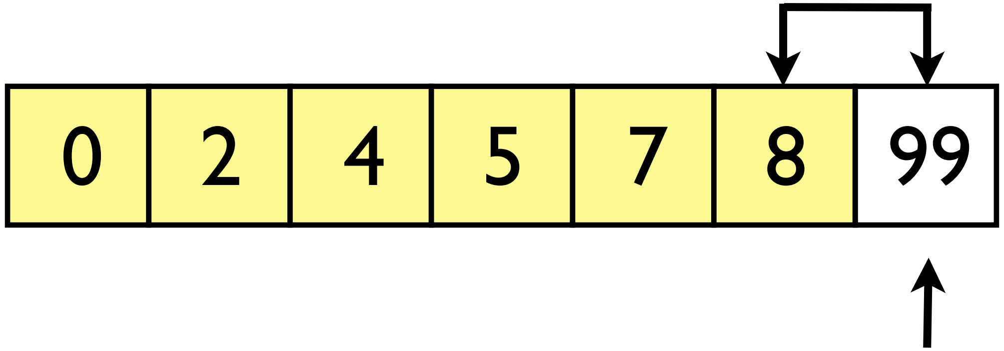


Selection Sort

Swap current min with first element of unsorted part

Selection Sort: Algorithm

Sort a given list of integers (from small to large).



Selection Sort

Swap current min with first element of unsorted part

Selection Sort: Algorithm

Sort a given list of integers (from small to large).

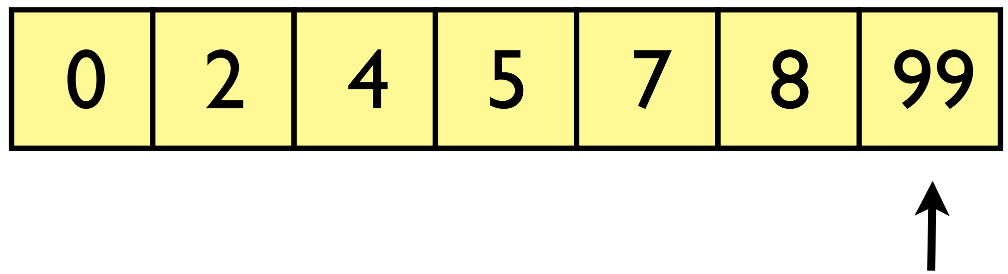
0	2	4	5	7	8	99

Selection Sort

Done!

Selection Sort: Running Time

Sort a given list of integers (from small to large).



Selection Sort

How many steps does this take (in the worst case)?

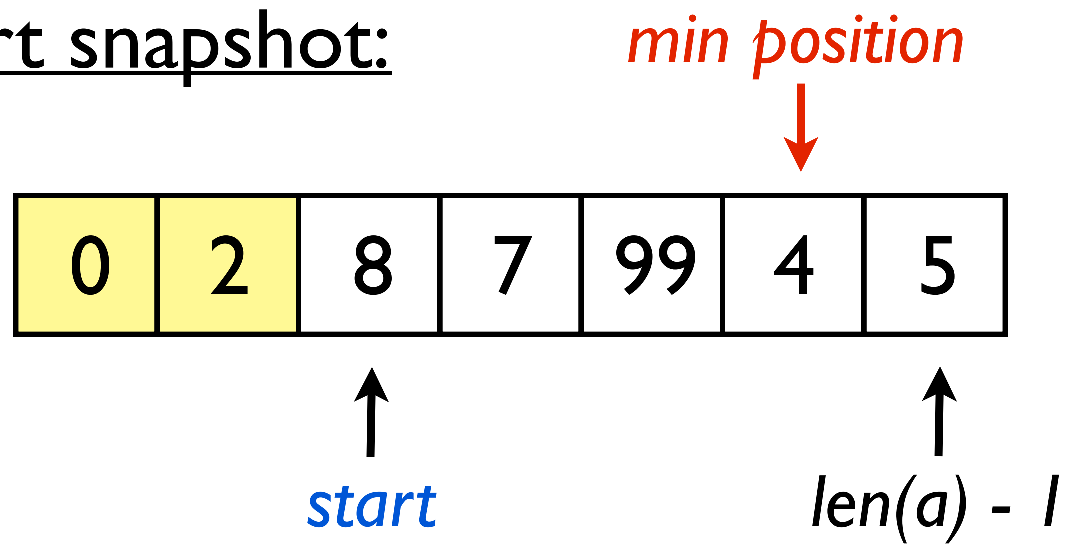
$$\sim N + (N-1) + (N-2) + \cdots + 1 \quad = \frac{N^2}{2} + \frac{N}{2}$$

(As *N* increases, small terms lose significance.)

Running time is $O(N^2)$.

Selection Sort: Code

<u>Selection sort snapshot:</u>



Find the *min position* from *start* to *len(a) - 1*

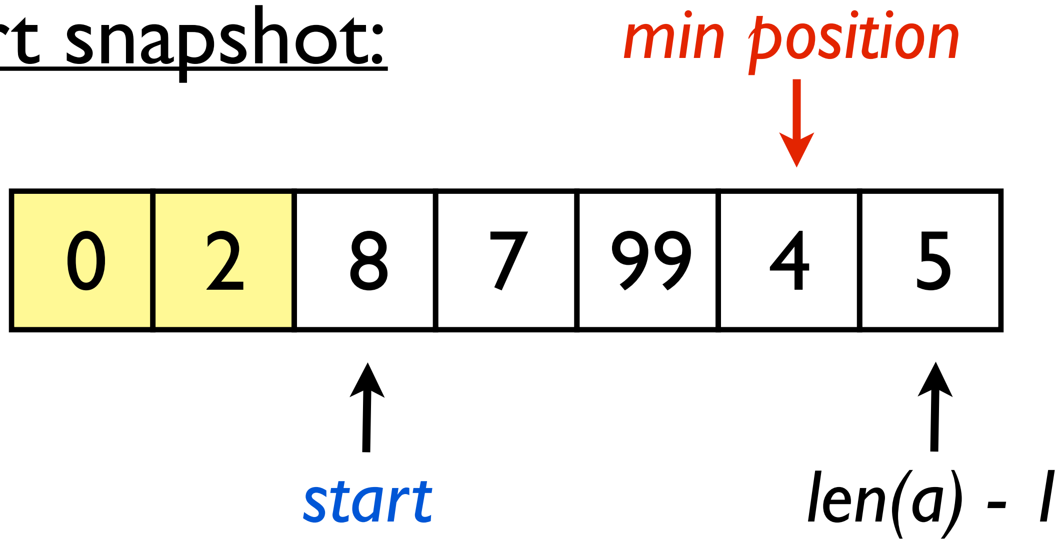
Swap elements in *min position* and *start*

Increment *start*

Repeat

Selection Sort: Code

Selection sort snapshot:



for *start* = 0 to *len(a)-1*:

Find the *min position* from *start* to *len(a) - 1*

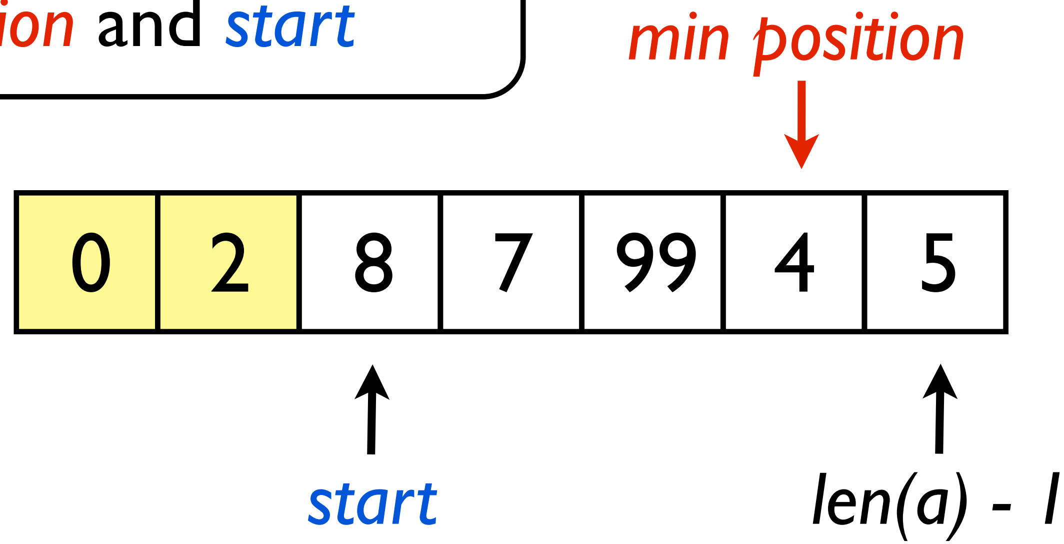
Swap elements in *min position* and *start*

Selection Sort: Code

for *start* = 0 to *len(a)-1*:

  Find the *min position* from *start* to *len(a) - 1*

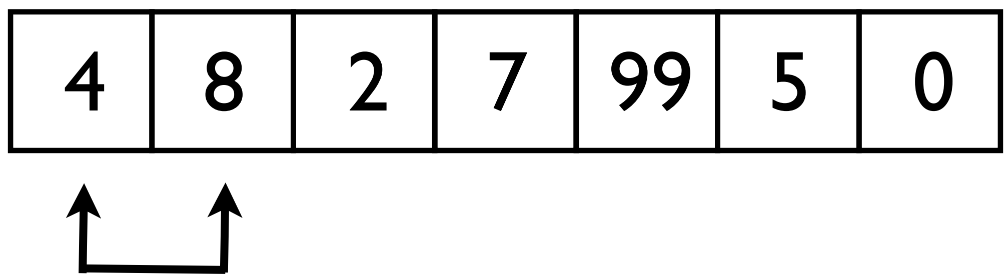
  Swap elements in *min position* and *start*




```
def selectionSort(a):
  for start in range(len(a)):
    currentMinIndex = start
    for i in range(start, len(a)):
      if(a[i] < a[currentMinIndex]):
        currentMinIndex = i
    (a[currentMinIndex], a[start]) = (a[start], a[currentMinIndex])
```

Bubble Sort: Algorithm

Sort a given list of integers (from small to large).



Bubble Sort

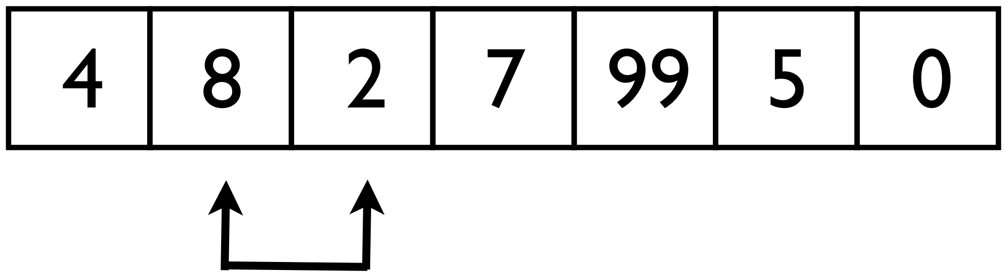
Compare each pair of adjacent items (left to right).

Swap them if they are in the wrong order.

Repeat until no more swaps are needed.

Bubble Sort: Algorithm

Sort a given list of integers (from small to large).



Bubble Sort

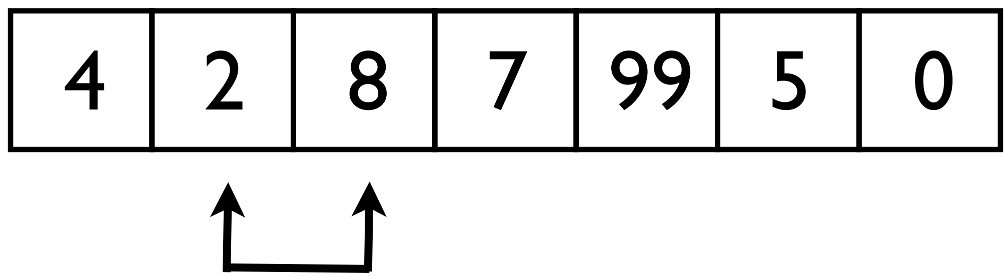
Compare each pair of adjacent items (left to right).

Swap them if they are in the wrong order.

Repeat until no more swaps are needed.

Bubble Sort: Algorithm

Sort a given list of integers (from small to large).



Bubble Sort

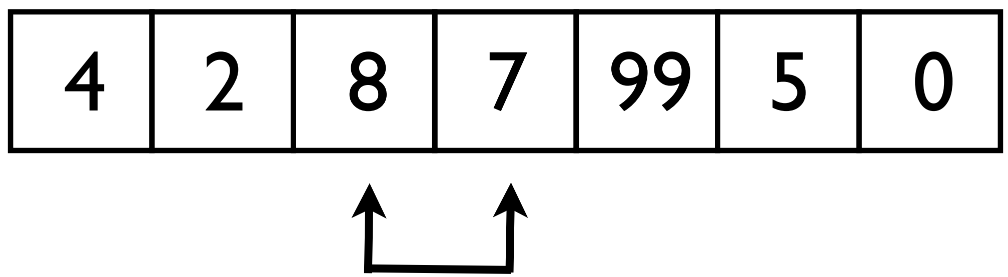
Compare each pair of adjacent items (left to right).

Swap them if they are in the wrong order.

Repeat until no more swaps are needed.

Bubble Sort: Algorithm

Sort a given list of integers (from small to large).



Bubble Sort

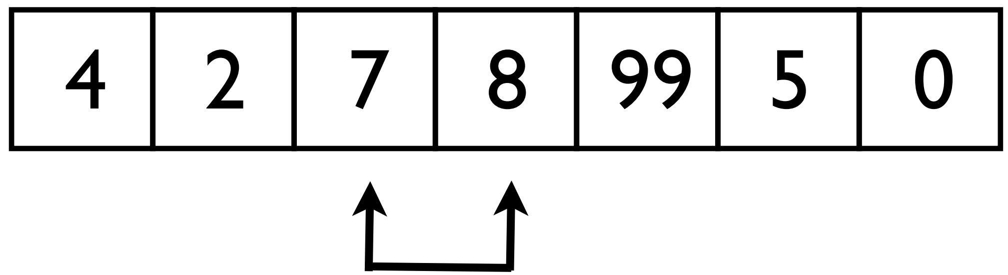
Compare each pair of adjacent items (left to right).

Swap them if they are in the wrong order.

Repeat until no more swaps are needed.

Bubble Sort: Algorithm

Sort a given list of integers (from small to large).



Bubble Sort

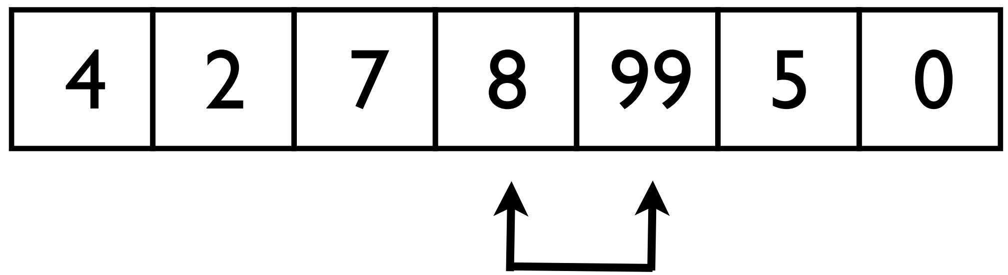
Compare each pair of adjacent items (left to right).

Swap them if they are in the wrong order.

Repeat until no more swaps are needed.

Bubble Sort: Algorithm

Sort a given list of integers (from small to large).



Bubble Sort

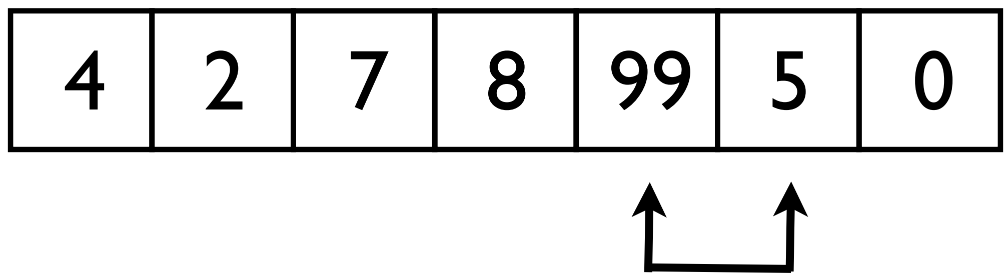
Compare each pair of adjacent items (left to right).

Swap them if they are in the wrong order.

Repeat until no more swaps are needed.

Bubble Sort: Algorithm

Sort a given list of integers (from small to large).



Bubble Sort

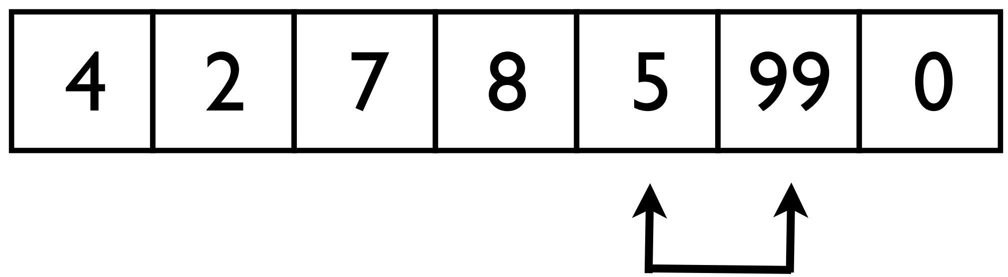
Compare each pair of adjacent items (left to right).

Swap them if they are in the wrong order.

Repeat until no more swaps are needed.

Bubble Sort: Algorithm

Sort a given list of integers (from small to large).



Bubble Sort

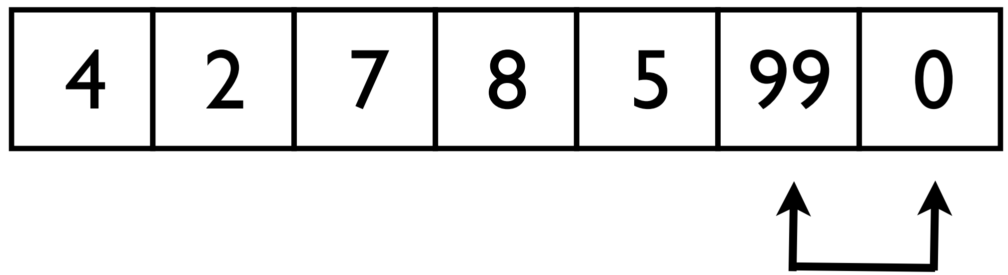
Compare each pair of adjacent items (left to right).

Swap them if they are in the wrong order.

Repeat until no more swaps are needed.

Bubble Sort: Algorithm

Sort a given list of integers (from small to large).



Bubble Sort

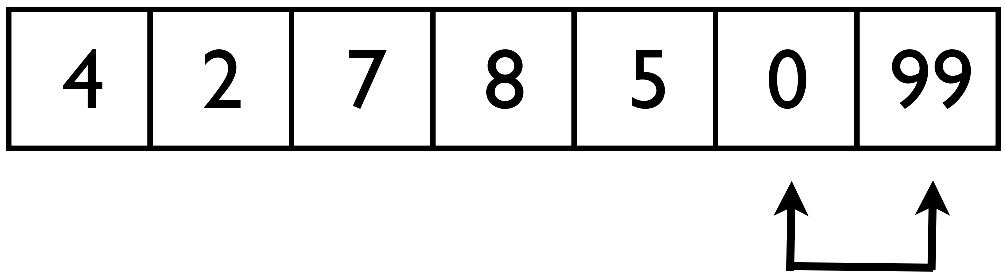
Compare each pair of adjacent items (left to right).

Swap them if they are in the wrong order.

Repeat until no more swaps are needed.

Bubble Sort: Algorithm

Sort a given list of integers (from small to large).



Bubble Sort

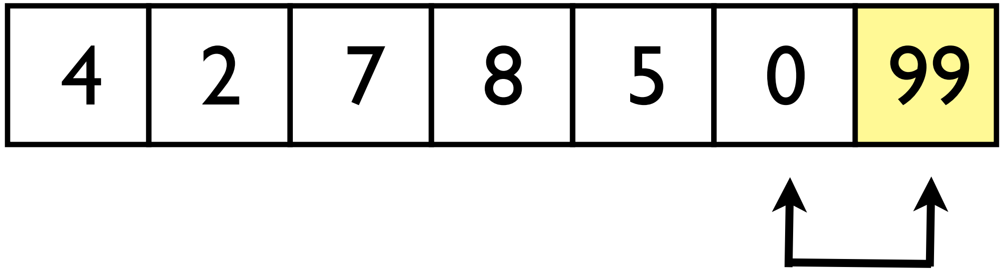
Compare each pair of adjacent items (left to right).

Swap them if they are in the wrong order.

Repeat until no more swaps are needed.

Bubble Sort: Algorithm

Sort a given list of integers (from small to large).



Bubble Sort

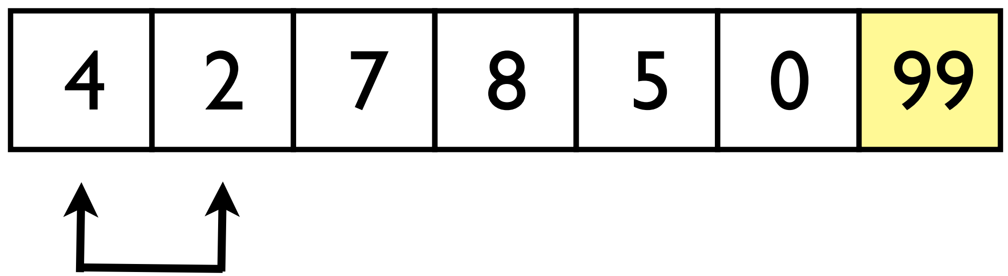
Compare each pair of adjacent items (left to right).

Swap them if they are in the wrong order.

Repeat until no more swaps are needed.

Bubble Sort: Algorithm

Sort a given list of integers (from small to large).



Bubble Sort

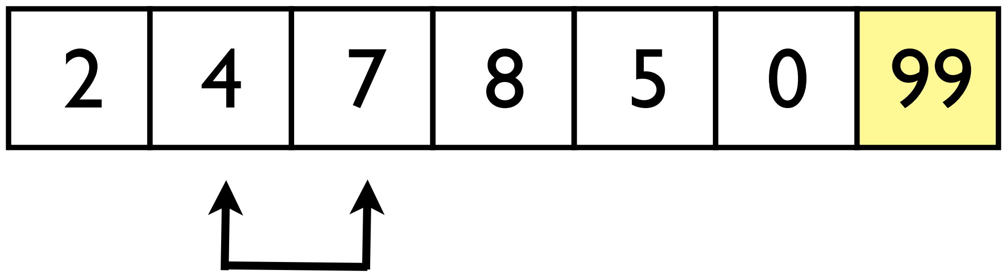
Compare each pair of adjacent items (left to right).

Swap them if they are in the wrong order.

Repeat until no more swaps are needed.

Bubble Sort: Algorithm

Sort a given list of integers (from small to large).



Bubble Sort

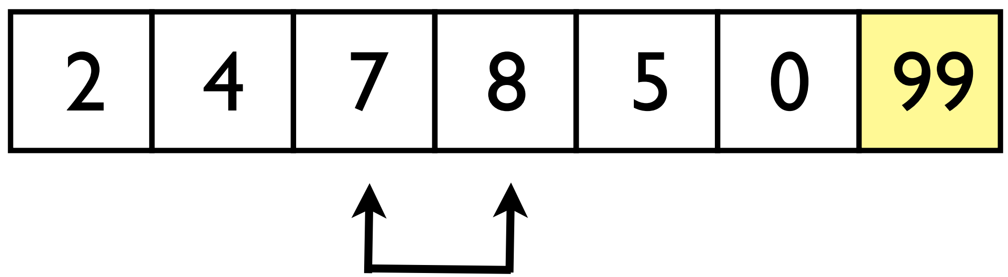
Compare each pair of adjacent items (left to right).

Swap them if they are in the wrong order.

Repeat until no more swaps are needed.

Bubble Sort: Algorithm

Sort a given list of integers (from small to large).



Bubble Sort

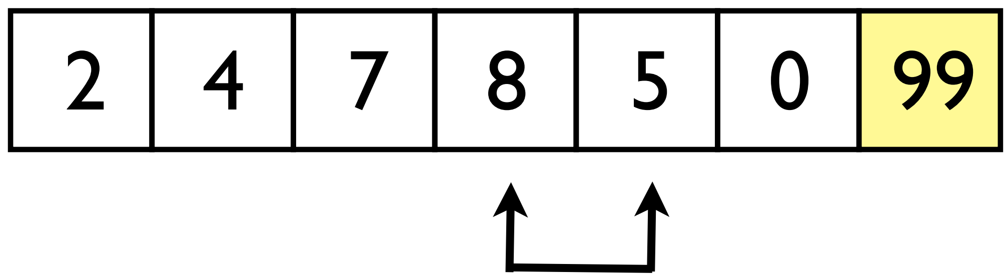
Compare each pair of adjacent items (left to right).

Swap them if they are in the wrong order.

Repeat until no more swaps are needed.

Bubble Sort: Algorithm

Sort a given list of integers (from small to large).



Bubble Sort

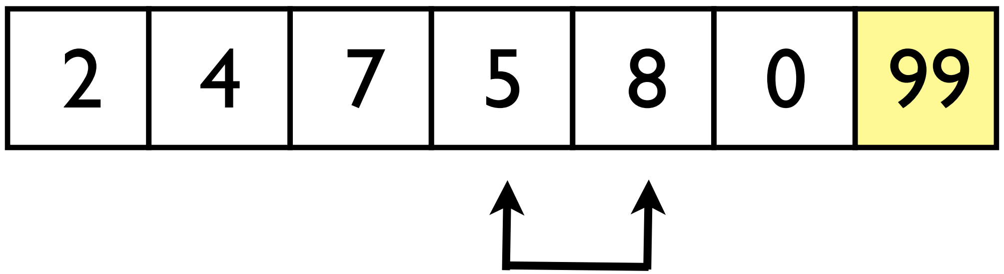
Compare each pair of adjacent items (left to right).

Swap them if they are in the wrong order.

Repeat until no more swaps are needed.

Bubble Sort: Algorithm

Sort a given list of integers (from small to large).



Bubble Sort

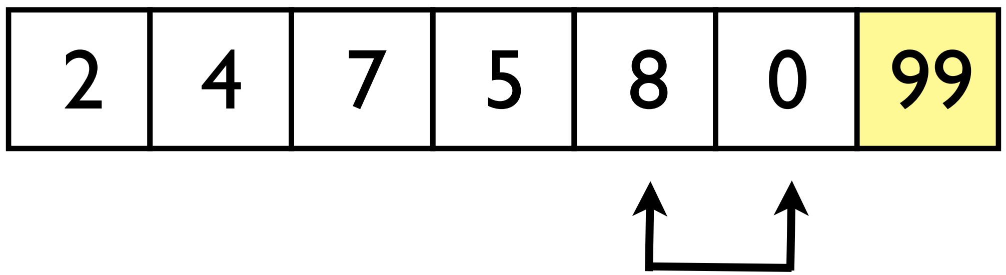
Compare each pair of adjacent items (left to right).

Swap them if they are in the wrong order.

Repeat until no more swaps are needed.

Bubble Sort: Algorithm

Sort a given list of integers (from small to large).



Bubble Sort

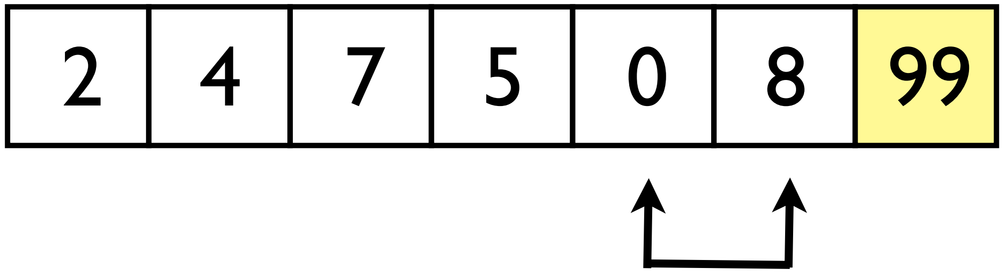
Compare each pair of adjacent items (left to right).

Swap them if they are in the wrong order.

Repeat until no more swaps are needed.

Bubble Sort: Algorithm

Sort a given list of integers (from small to large).



Bubble Sort

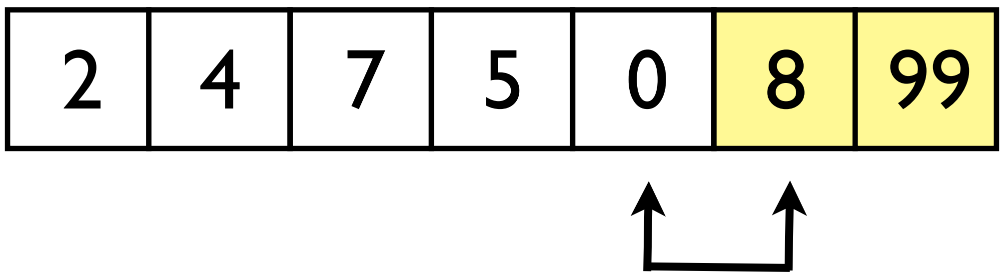
Compare each pair of adjacent items (left to right).

Swap them if they are in the wrong order.

Repeat until no more swaps are needed.

Bubble Sort: Algorithm

Sort a given list of integers (from small to large).



Bubble Sort

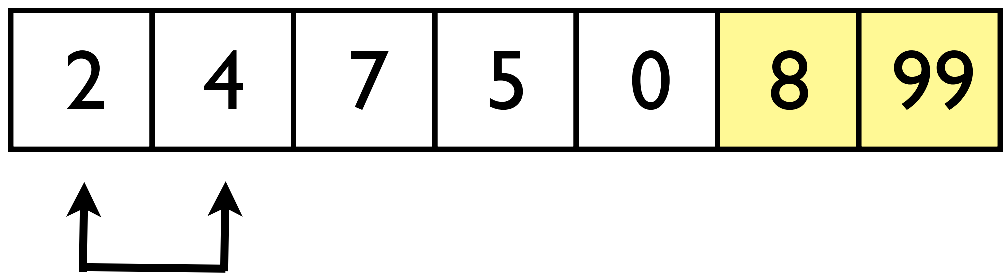
Compare each pair of adjacent items (left to right).

Swap them if they are in the wrong order.

Repeat until no more swaps are needed.

Bubble Sort: Algorithm

Sort a given list of integers (from small to large).



Bubble Sort

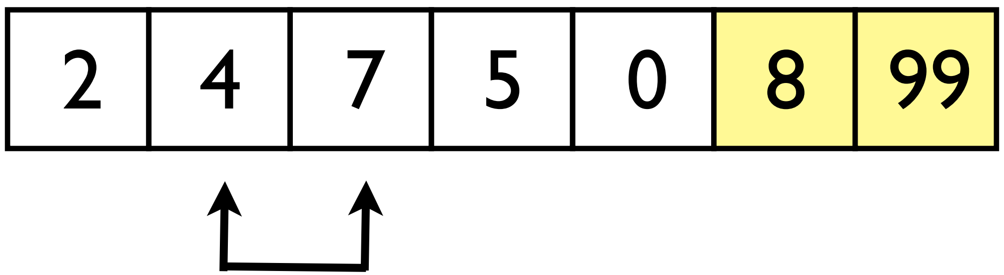
Compare each pair of adjacent items (left to right).

Swap them if they are in the wrong order.

Repeat until no more swaps are needed.

Bubble Sort: Algorithm

Sort a given list of integers (from small to large).



Bubble Sort

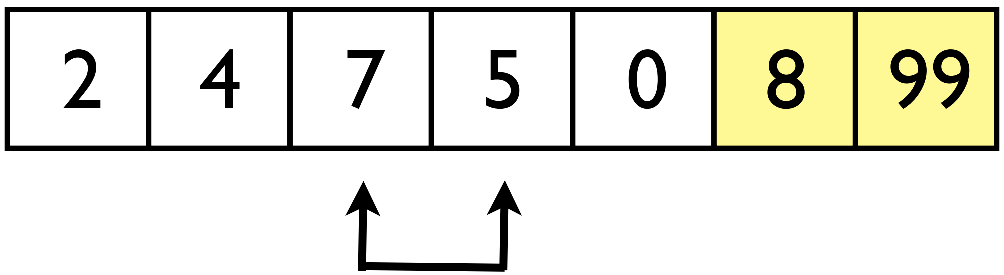
Compare each pair of adjacent items (left to right).

Swap them if they are in the wrong order.

Repeat until no more swaps are needed.

Bubble Sort: Algorithm

Sort a given list of integers (from small to large).



Bubble Sort

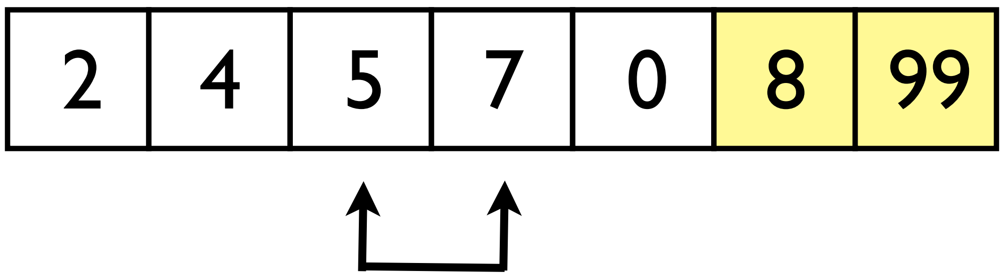
Compare each pair of adjacent items (left to right).

Swap them if they are in the wrong order.

Repeat until no more swaps are needed.

Bubble Sort: Algorithm

Sort a given list of integers (from small to large).



Bubble Sort

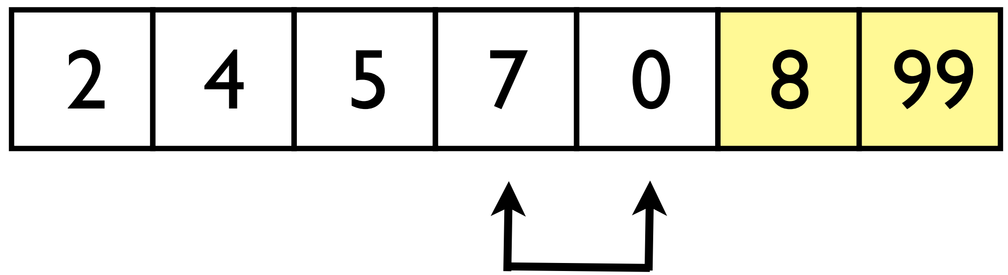
Compare each pair of adjacent items (left to right).

Swap them if they are in the wrong order.

Repeat until no more swaps are needed.

Bubble Sort: Algorithm

Sort a given list of integers (from small to large).



Bubble Sort

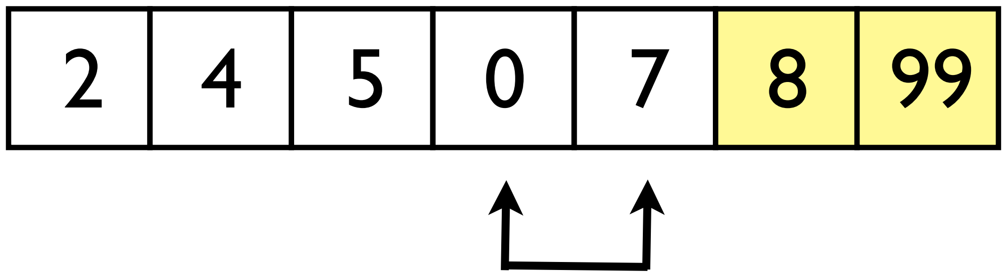
Compare each pair of adjacent items (left to right).

Swap them if they are in the wrong order.

Repeat until no more swaps are needed.

Bubble Sort: Algorithm

Sort a given list of integers (from small to large).



Bubble Sort

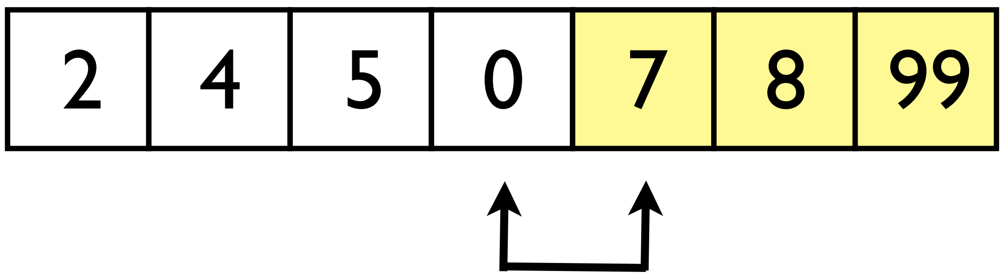
Compare each pair of adjacent items (left to right).

Swap them if they are in the wrong order.

Repeat until no more swaps are needed.

Bubble Sort: Algorithm

Sort a given list of integers (from small to large).



Bubble Sort

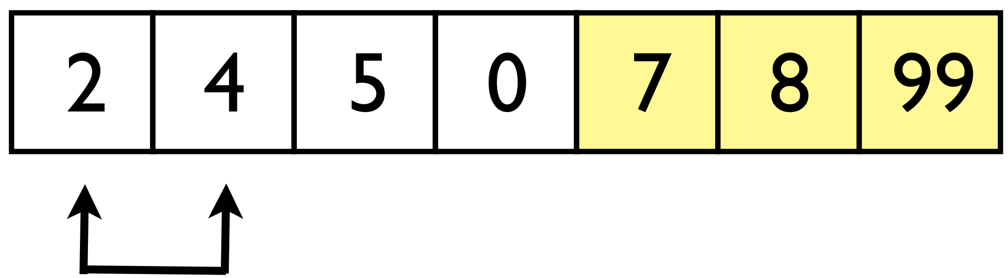
Compare each pair of adjacent items (left to right).

Swap them if they are in the wrong order.

Repeat until no more swaps are needed.

Bubble Sort: Algorithm

Sort a given list of integers (from small to large).



Bubble Sort

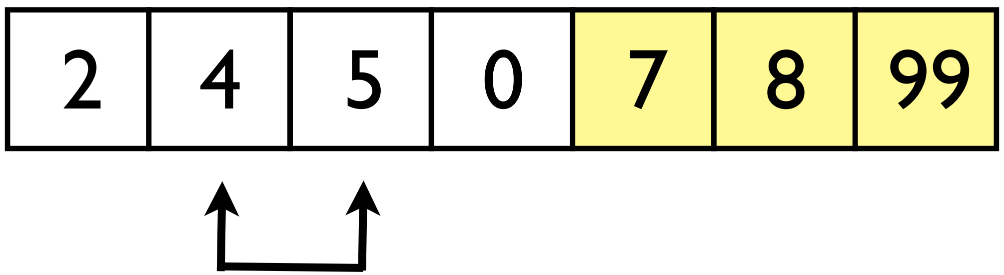
Compare each pair of adjacent items (left to right).

Swap them if they are in the wrong order.

Repeat until no more swaps are needed.

Bubble Sort: Algorithm

Sort a given list of integers (from small to large).



Bubble Sort

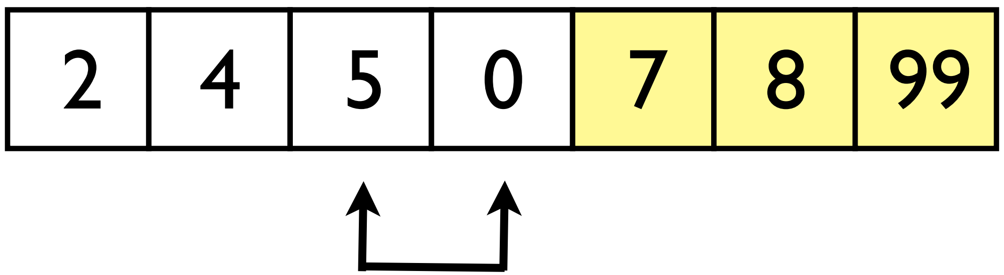
Compare each pair of adjacent items (left to right).

Swap them if they are in the wrong order.

Repeat until no more swaps are needed.

Bubble Sort: Algorithm

Sort a given list of integers (from small to large).



Bubble Sort

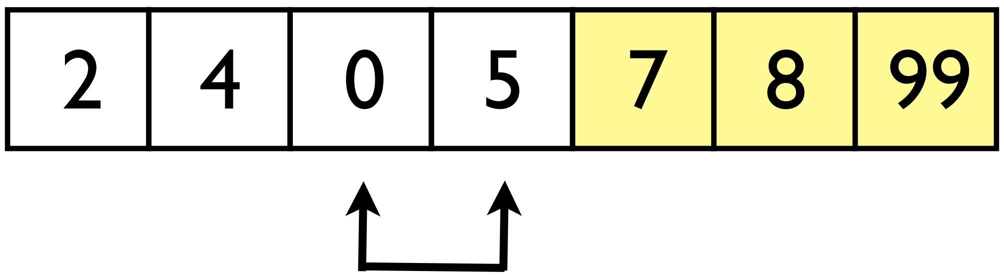
Compare each pair of adjacent items (left to right).

Swap them if they are in the wrong order.

Repeat until no more swaps are needed.

Bubble Sort: Algorithm

Sort a given list of integers (from small to large).



Bubble Sort

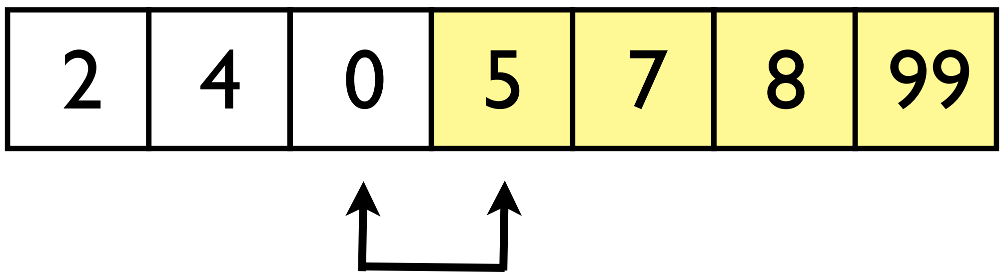
Compare each pair of adjacent items (left to right).

Swap them if they are in the wrong order.

Repeat until no more swaps are needed.

Bubble Sort: Algorithm

Sort a given list of integers (from small to large).



Bubble Sort

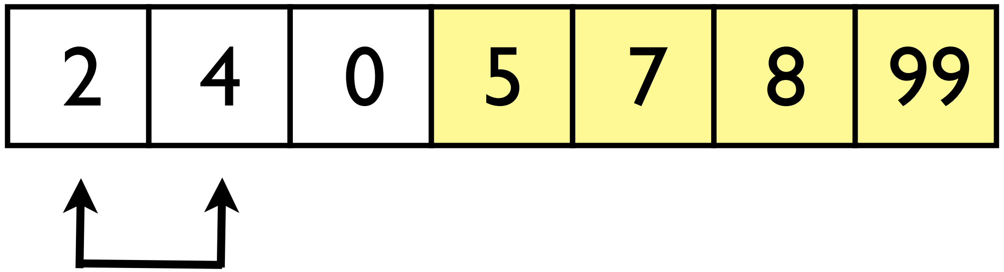
Compare each pair of adjacent items (left to right).

Swap them if they are in the wrong order.

Repeat until no more swaps are needed.

Bubble Sort: Algorithm

Sort a given list of integers (from small to large).



Bubble Sort

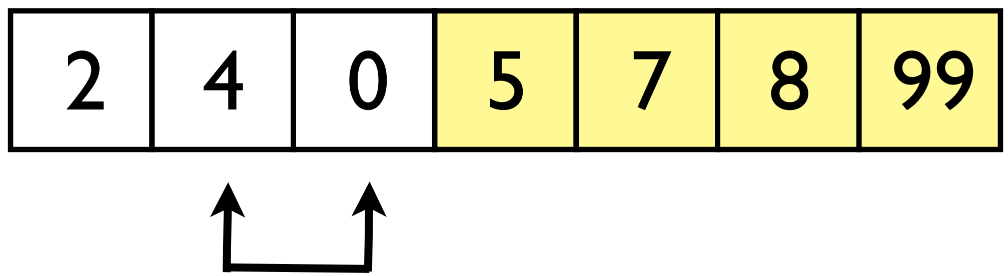
Compare each pair of adjacent items (left to right).

Swap them if they are in the wrong order.

Repeat until no more swaps are needed.

Bubble Sort: Algorithm

Sort a given list of integers (from small to large).



Bubble Sort

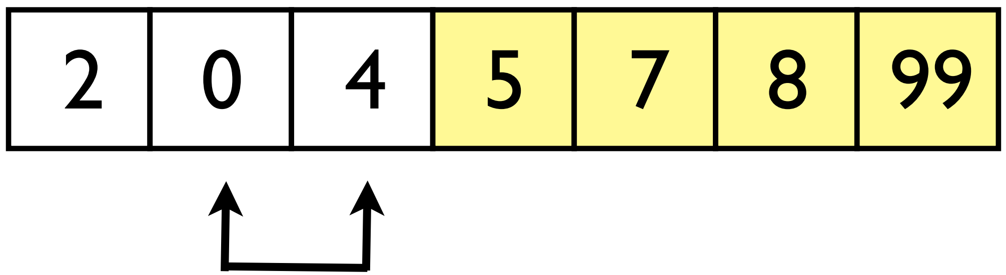
Compare each pair of adjacent items (left to right).

Swap them if they are in the wrong order.

Repeat until no more swaps are needed.

Bubble Sort: Algorithm

Sort a given list of integers (from small to large).



Bubble Sort

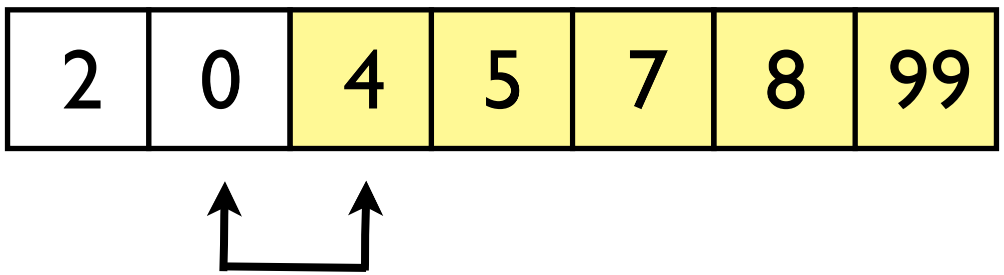
Compare each pair of adjacent items (left to right).

Swap them if they are in the wrong order.

Repeat until no more swaps are needed.

Bubble Sort: Algorithm

Sort a given list of integers (from small to large).



Bubble Sort

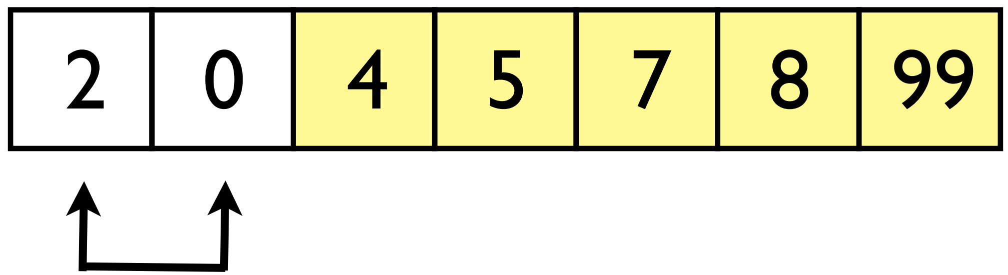
Compare each pair of adjacent items (left to right).

Swap them if they are in the wrong order.

Repeat until no more swaps are needed.

Bubble Sort: Algorithm

Sort a given list of integers (from small to large).



Bubble Sort

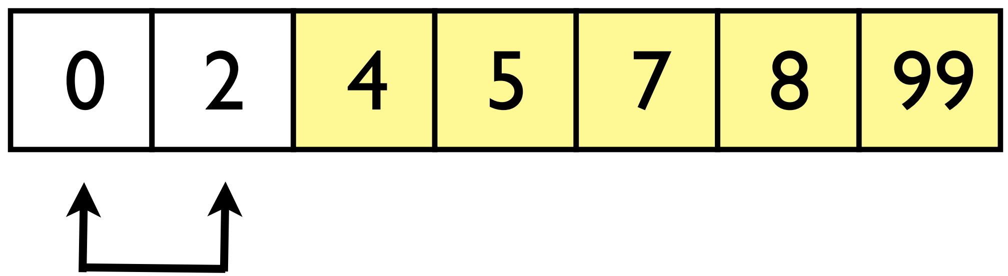
Compare each pair of adjacent items (left to right).

Swap them if they are in the wrong order.

Repeat until no more swaps are needed.

Bubble Sort: Algorithm

Sort a given list of integers (from small to large).



Bubble Sort

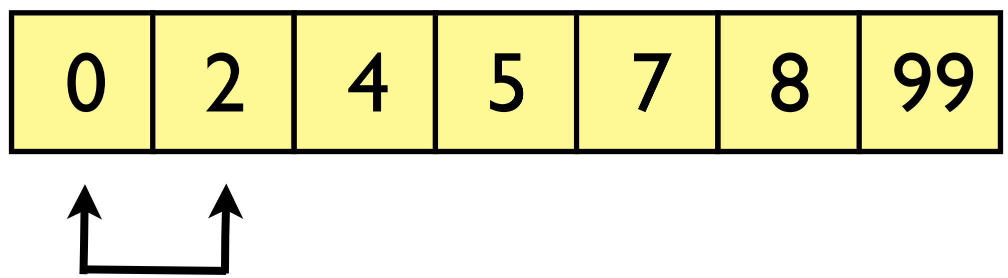
Compare each pair of adjacent items (left to right).

Swap them if they are in the wrong order.

Repeat until no more swaps are needed.

Bubble Sort: Algorithm

Sort a given list of integers (from small to large).



Bubble Sort

Compare each pair of adjacent items (left to right).

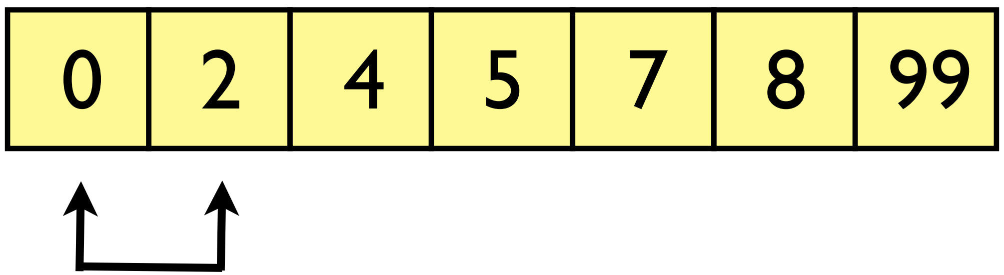
Swap them if they are in the wrong order.

Repeat until no more swaps are needed.

Large elements "bubble up"

Bubble Sort: Running Time

Sort a given list of integers (from small to large).



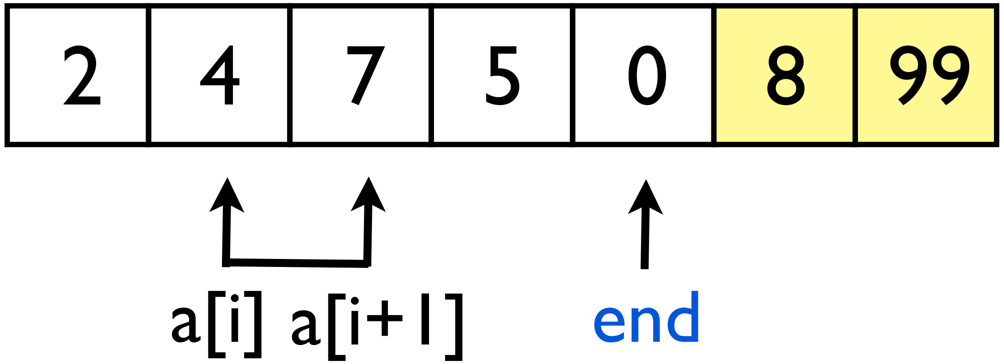
Bubble Sort

How many steps does this take (in the worst case)?

$O(N^2)$

Bubble Sort: Code

Bubble sort snapshot



repeat until no more swaps:

for i = 0 to end:

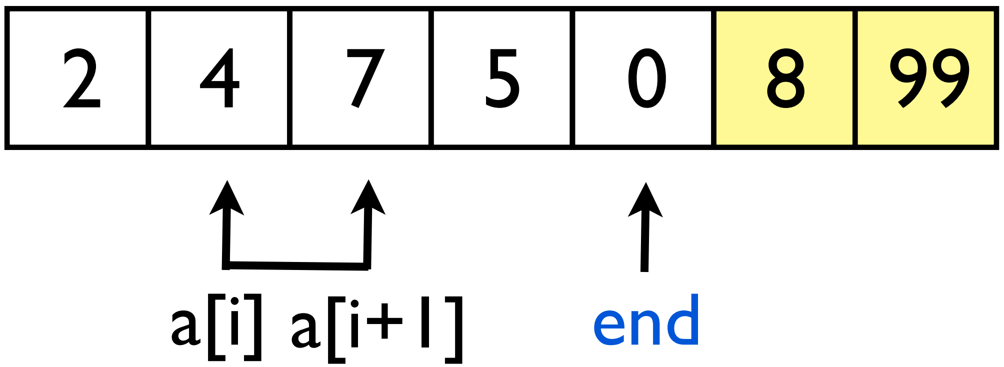
if a[i] > a[i+1], swap a[i] and a[i+1]

decrement end

Bubble Sort: Code

```
repeat until no more swaps:
    for i = 0 to end:
        if a[i] > a[i+1], swap a[i] and a[i+1]
    decrement end
```

```
def bubbleSort(a):
    swapped = True
    end = len(a)-1
    while(swapped):
        swapped = False
        for i in range(end):
            if(a[i] > a[i+1]):
                (a[i], a[i+1]) = (a[i+1], a[i])
                swapped = True
        end -= 1
```



Comparison: Selection Sort vs Bubble Sort

Worst case both take $O(N^2)$ steps.

How about best case?

Selection sort: $O(N^2)$

Bubble sort: $O(N)$

If your list is close to being sorted, bubble sort can be better.

Is there a better way?

Exercise

Write the code yourself:

linear search
binary search
selection sort
bubble sort