CST 370 – ADVANCED ALGORITHMS

3. Sorting pt 1

Objectives

You will be able to…

1. Explain how each of the following sorting algorithms work: Bubble sort, Selection sort, Insertion sort, Heap sort.
2. Implement an iterative algorithm in code given an informal written description of it.
3. Implement: Bubble sort, Selection sort, Insertion sort.

Sorting Algorithms

Why do I care? Here are some applications:

Sorting Algorithms

Why do I care? Here are some applications:

- Literally anything that displays user-generated content uses sorting algorithms.
  - Yelp reviews
  - Reddit posts
  - Facebook posts

Bubble Sort

Lining up by height/birthday/etc....

- Sort by swapping elements that are in the incorrect order.
- The max element “bubbles” to the end after every pass.

Bubble Sort

Lining up by height/birthday/etc....

- Sort by swapping elements that are in the incorrect order.
- The max element “bubbles” to the end after every pass.

Bubble Sort

Visualization

https://youtu.be/lyZQPjUT5B4?t=52

https://visualgo.net/en/sorting

Selection Sort

Sorting a pile of cards

- Sort by selecting the largest and putting it in the back
  - or selecting the smallest and putting it in the front.

- A very natural way to sort.

Selection Sort

Visualization

https://www.youtube.com/watch?v=Ns4TPTC8whw

https://visualgo.net/en/sorting

Insertion Sort

Sorting a hand of cards

- Sort by taking elements one at a time and finding the right place to insert it.

- A very natural way to sort when adding something to a sorted list.

Insertion Sort

Visualization

https://www.youtube.com/watch?v=ROalU379l3U

https://visualgo.net/en/sorting

Heap Sort

Getting a data structure to do it for you

- Selection sort except you use a heap to find the largest value.

- Can be implemented in-place by using the array representation of heaps.

Heap Sort

Visualization

https://www.youtube.com/watch?v=2DmK_H7IdTo

https://visualgo.net/en/sorting

________ Sort

Algorithm

Suppose we’re trying to sort an array…

1. Start with the sorted element counter at 0 and the current index at 0.
2. If the sorted element counter is one less than the size of the array, return.
3. Compare the element at the current index with the next element to the right. If they are out of order, swap them.
4. Increment the current index.
5. If the number of elements to the right of the current index is less than the sorted element counter, increment the sorted element counter and reset the current index to 0. Go back to (2).
6. Else, go back to (3).

________ Sort

Run through an example

[9, 1, 7, 5] - numSortedElements = 0

________ Sort

Run through an example

[9, 1, 7, 5] - numSortedElements = 0

[1, 9, 7, 5] - numSortedElements = 0

________ Sort

Run through an example

[9, 1, 7, 5] - numSortedElements = 0

[1, 9, 7, 5] - numSortedElements = 0

[1, 7, 9, 5] - numSortedElements = 0

________ Sort

Run through an example

[9, 1, 7, 5] - numSortedElements = 0

[1, 9, 7, 5] - numSortedElements = 0

[1, 7, 9, 5] - numSortedElements = 0

[1, 7, 5, 9] - numSortedElements = 1

_________ Sort

Run through an example

[$\underline{1, 7}$, 5, 9] - numSortedElements = 1

________ Sort

Run through an example

[$\underline{1,\ 7}$, 5, 9] - numSortedElements = 1

[1, $\underline{7,\ 5}$, 9] - numSortedElements = 1

________ Sort

Run through an example

[$\underline{1,\ 7}$, 5, 9] - numSortedElements = 1

[1, $\underline{7,\ 5}$, 9] - numSortedElements = 1

[1, 5, 7, 9] - numSortedElements = 2

________ Sort

Run through an example

[1, 7, 5, 9] - numSortedElements = 1

[1, 7, 5, 9] - numSortedElements = 1

[1, 5, 7, 9] - numSortedElements = 2

[1, 5, 7, 9] - numSortedElements = 2

________ Sort

Run through an example

[$\underline{1,\ 7}$, 5, 9] - numSortedElements = 1

[1, $\underline{7,\ 5}$, 9] - numSortedElements = 1

[1, 5, 7, 9] - numSortedElements = 2

[$\underline{1,\ 5}$, 7, 9] - numSortedElements = 2

[1, 5, 7, 9] - numSortedElements = 3

________ Sort

Run through an example

[$\underline{1,\ 7}$, 5, 9] - numSortedElements = 1

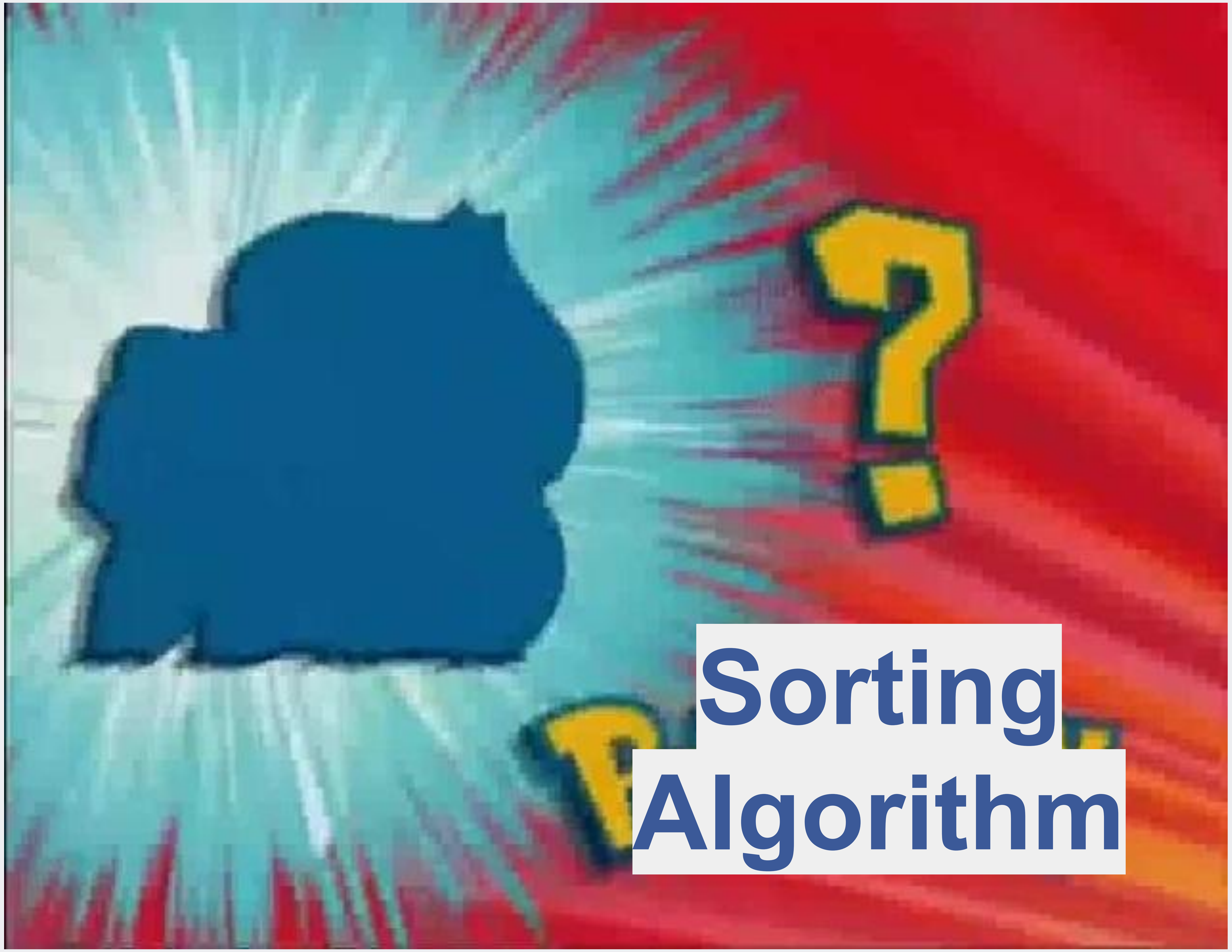
[1, $\underline{7,\ 5}$, 9] - numSortedElements = 1

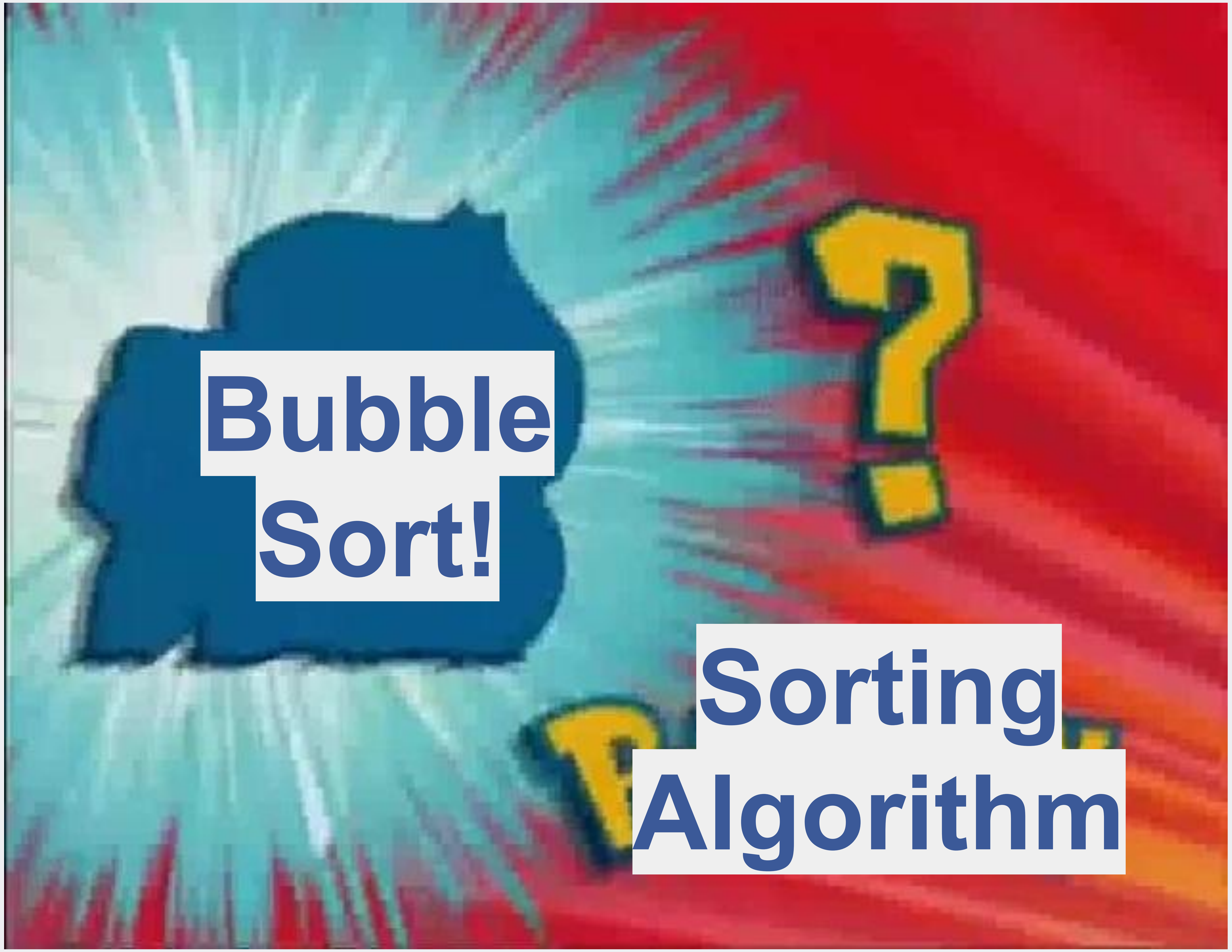
[1, 5, 7, 9] - numSortedElements = 2

[$\underline{1,\ 5}$, 7, 9] - numSortedElements = 2

[1, 5, 7, 9] - numSortedElements = 3

Done!


?
Sorting
Algorithm


Bubble Sort!
Sorting Algorithm

Implementing Bubble Sort

https://repl.it/@dsyang/InClassBubbleSort

Now it’s your turn!

Grab a worksheet per person

If you have a….

Red card: grab one from the right

Black card: grab one from the left.

________ Sort

Algorithm

Suppose we're trying to sort an array/dynamic array…

1. Start the current index at 1.
2. If the current index has reached the size of the array (is out of bounds) return. Otherwise, temporarily store the element in the current index.
3. Set the previous index to the one before the current index.
4. If the previous index is less than 0 (out of bounds), go to step (6).
5. Compare the elements in the current and previous indices. If the current index has the smaller element, copy the previous element to the current index, decrement the previous index, and return to step (4). Else, go on to step (6).
6. Put the stored temporary element into the position after the previous index.
7. Increment the current index and return to step (2).

__________ Sort

Run through an example

[9, 1, 7, 5] - ▮ current index = 1, previous index = 0

________ Sort

Run through an example

[9, 1, 7, 5] - current index = 1, previous index = 0

[9, 1, 7, 5] - 1 current index = 1, previous index = 0

________ Sort

Run through an example

[9, 1, 7, 5] - ▇ current index = 1, previous index = 0

[9, 1, 7, 5] - 1 current index = 1, previous index = 0

[9, 9, 7, 5] - 1 current index = 1, previous index = -1

________ Sort

Run through an example

[9, 1, 7, 5] - current index = 1, previous index = 0

[9, 1, 7, 5] - 1 current index = 1, previous index = 0

[9, 9, 7, 5] - 1 current index = 1, previous index = -1

[1, 9, 7, 5] - current index = 2, previous index = -1

________ Sort

Run through an example

[9, 1, 7, 5] - █ current index = 1. Start

...

[1, 9, 7, 5] - █ current index = 2.

________ Sort

Run through an example

[9, 1, 7, 5] - [redacted] current index = 1. Start

...

[1, 9, 7, 5] - [redacted] current index = 2.

...

[1, 7, 9, 5] - [redacted] current index = 3.

________ Sort

Run through an example

[9, 1, 7, 5] - █ current index = 1. Start

...

[1, 9, 7, 5] - █ current index = 2.

...

[1, 7, 9, 5] - █ current index = 3.

...

[1, 5, 7, 9] - █ current index = 4.

__________ Sort

Run through an example

[9, 1, 7, 5] - current index = 1. Start

...

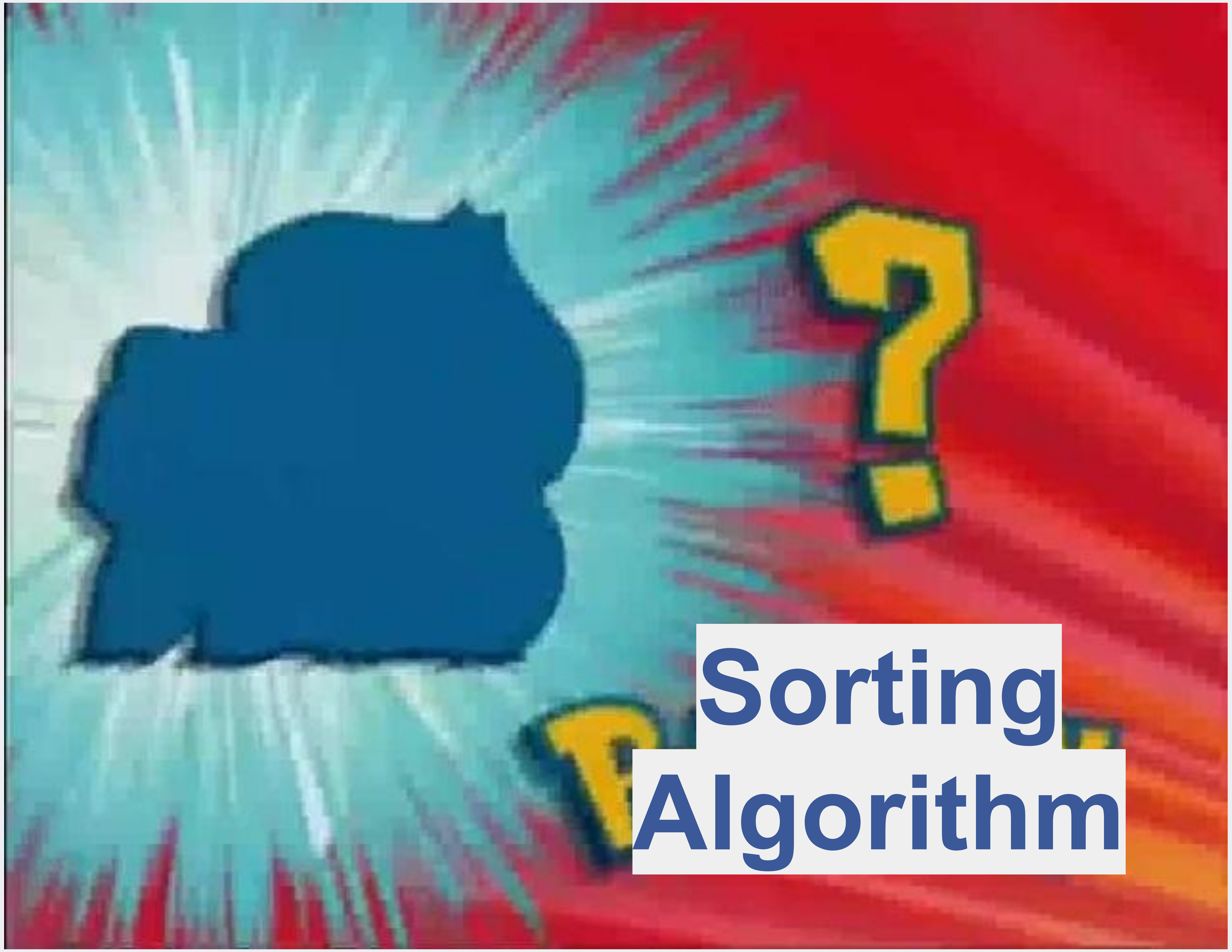
[1, 9, 7, 5] - current index = 2.

...

[1, 7, 9, 5] - current index = 3.

...

[1, 5, 7, 9] - current index = 4. Done!


?
Sorting
Algorithm

________ Sort

Algorithm

Suppose we’re trying to sort an array…

1. Start with the first index.
2. Find the smallest element in that index or to the right.
3. Swap the smallest element with the current element.
4. Increment the index.
5. If we’ve reached the last index, return. We’re done.
6. Else, go back to (2).

___________ Sort

Run through an example

[9, 1, 7, 5] - index = 0

__________ Sort

Run through an example

[9, 1, 7, 5] - index = 0

[9, $\underline{1}$, 7, 5] - index = 0

________ Sort

Run through an example

[9, 1, 7, 5] - index = 0

[9, $\underline{1}$, 7, 5] - index = 0

[1, $\underline{9}$, 7, 5] - index = 1

________ Sort

Run through an example

[9, 1, 7, 5] - index = 0

[9, 1, 7, 5] - index = 0

[1, 9, 7, 5] - index = 1

[1, 9, 7, 5] - index = 1

[1, 5, 7, 9] - index = 2

________ Sort

Run through an example

[9, 1, 7, 5] - index = 0

[9, 1, 7, 5] - index = 0

[1, 9, 7, 5] - index = 1

[1, 9, 7, 5] - index = 1

[1, 5, 7, 9] - index = 2

[1, 5, 7, 9] - index = 2

[1, 5, 7, 9] - index = 3

__________ Sort

Run through an example

[9, 1, 7, 5] - index = 0

[9, 1, 7, 5] - index = 0

[1, 9, 7, 5] - index = 1

[1, 9, 7, 5] - index = 1

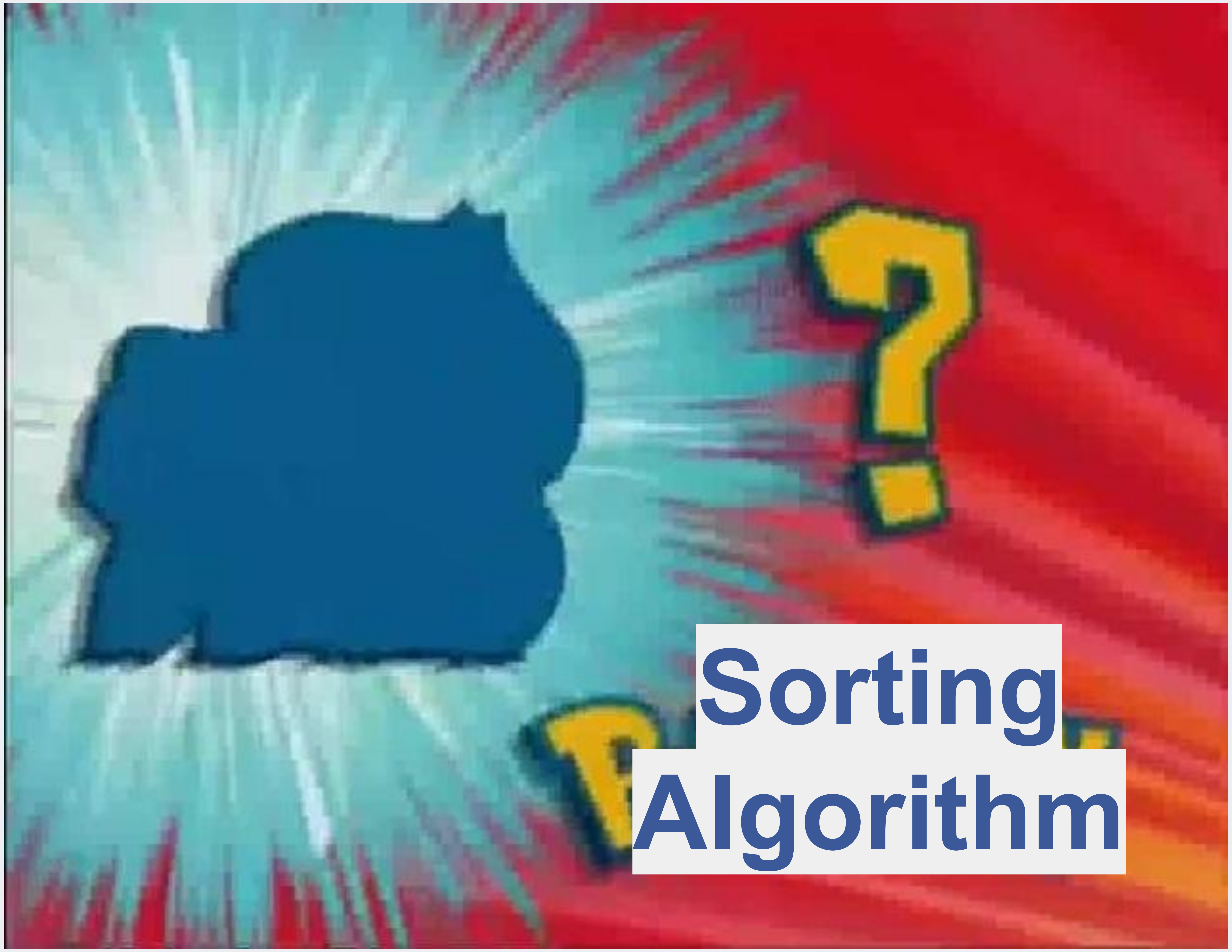
[1, 5, 7, 9] - index = 2

[1, 5, 7, 9] - index = 2

[1, 5, 7, 9] - index = 2

[1, 5, 7, 9] - index = 3

Done!


Sorting
Algorithm

Heap Sort

Algorithm

Suppose we're trying to sort an array…

1. Start with heapEnd = size - 1.
2. Rearrange the array to be a max binary heap.
3. Swap index 0 with heapEnd. Decrement heapEnd.
4. If heapEnd == 0, return.
5. Heapify the max binary heap from index 0 - heapEnd.
6. Go back to (3).

Heap Sort

Run through an example

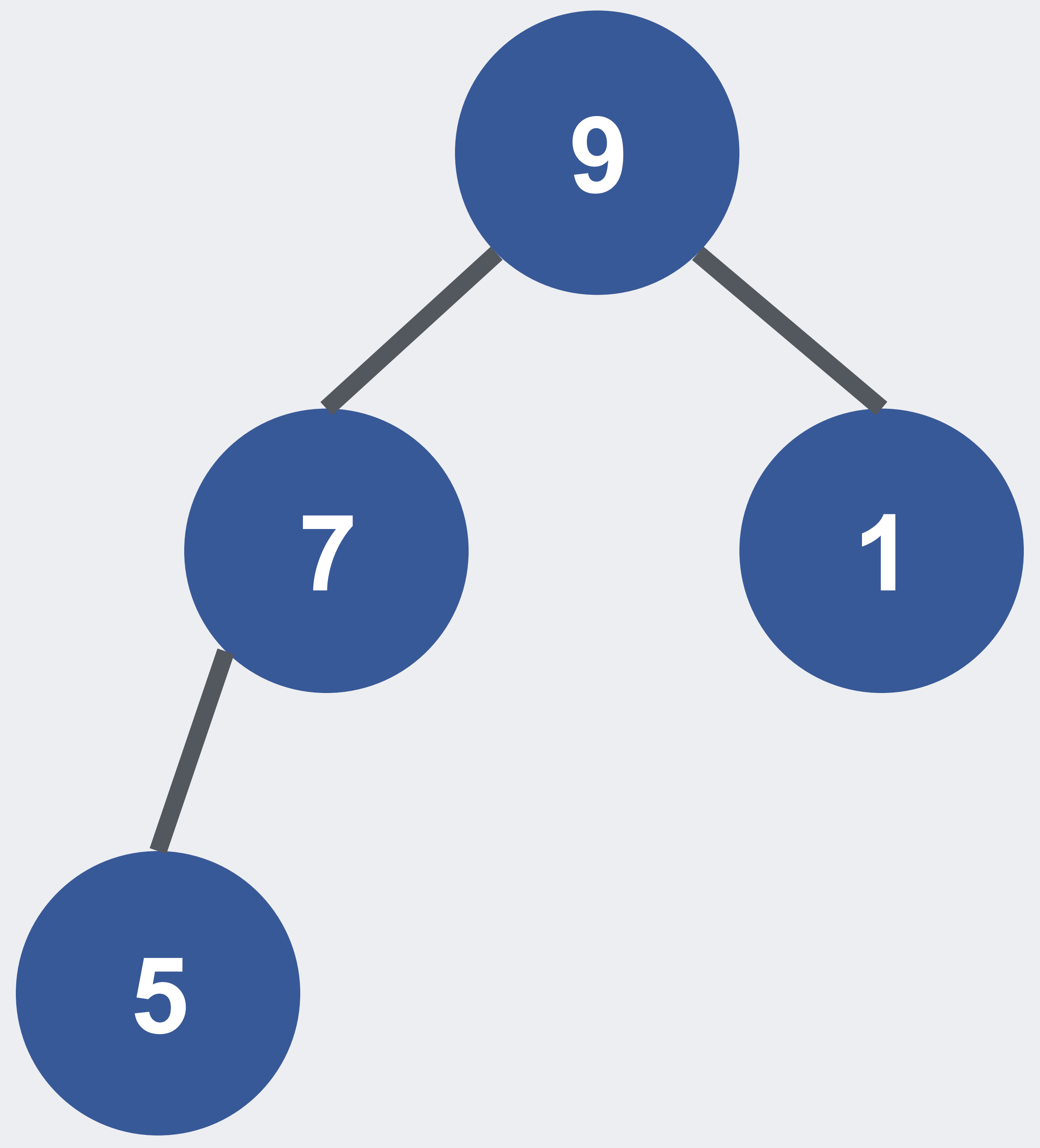
[9, 1, 7, 5] - heapEnd = 3

Heap Sort

Run through an example

[9, 1, 7, 5] - heapEnd = 3

[9, 7, 1, 5] - heapify



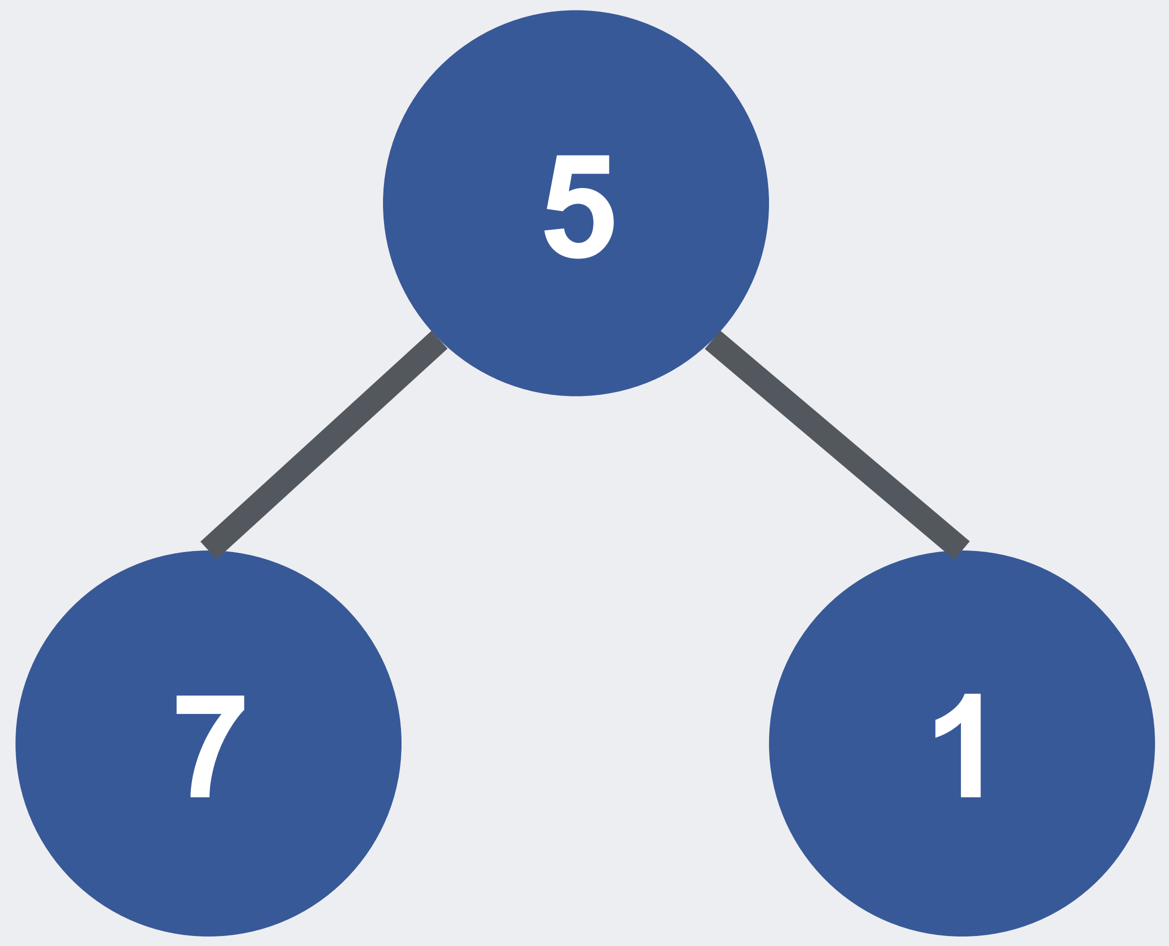
Heap Sort

Run through an example

[9, 1, 7, 5] - heapEnd = 3

[9, 7, 1, 5] - heapify

[5, 7, 1, 9] - heapEnd = 2



Heap Sort

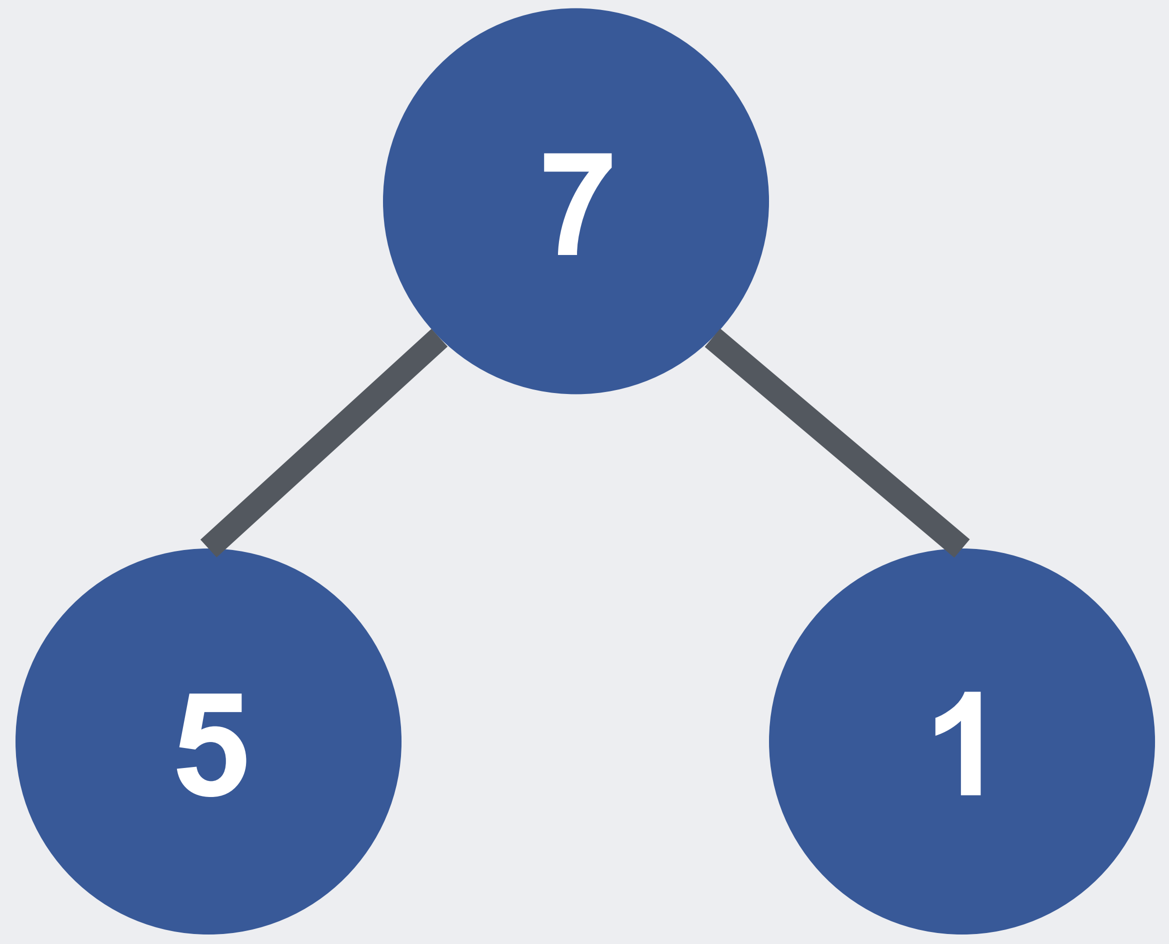
Run through an example

[9, 1, 7, 5] - heapEnd = 3

[9, 7, 1, 5] - heapify!

[5, 7, 1, 9] - heapEnd = 2

[7, 5, 1, 9] - heapify



Heap Sort

Run through an example

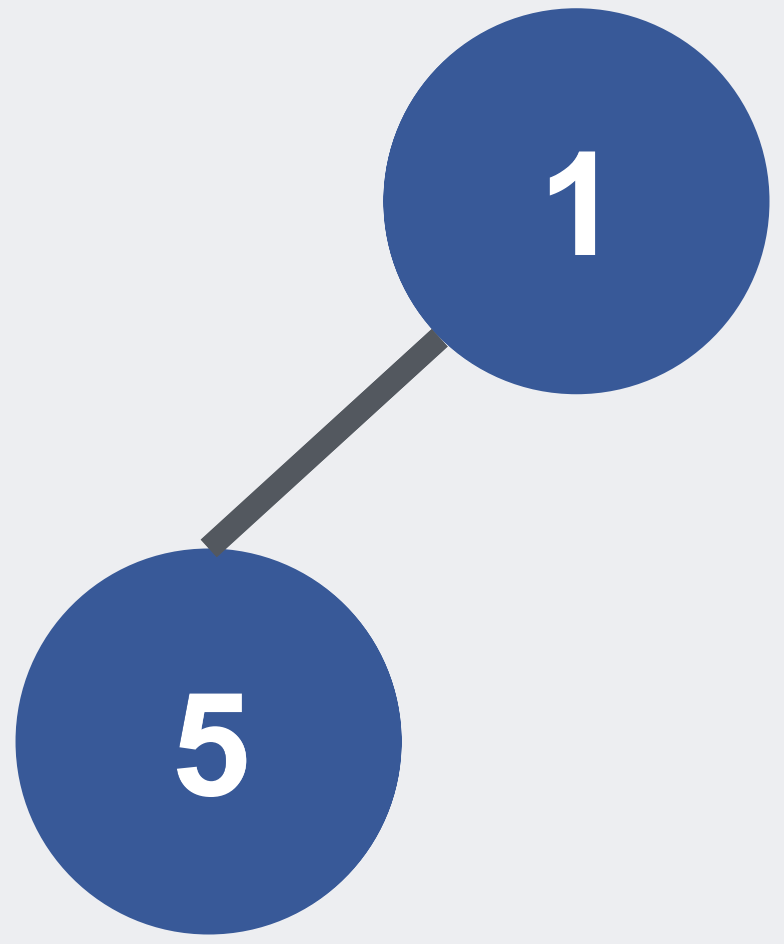
[9, 1, 7, 5] - heapEnd = 3

[9, 7, 1, 5] - heapify!

[5, 7, 1, 9] - heapEnd = 2

[7, 5, 1, 9] - heapify

[1, 5, 7, 9] - heapEnd = 1

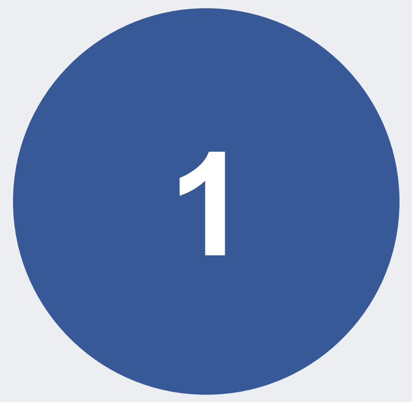


Heap Sort

Run through an example

[5, 1, 7, 9] - heapify

[1, 5, 7, 9] - heapEnd = 0



Heap Sort

Run through an example

[5, 1, 7, 9] - heapify

[1, 5, 7, 9] - heapEnd = 0

[1, 5, 7, 9] - Done!

Implementing Heap Sort

https://repl.it/@dsyang/HeapSort