Lecture 21: Introduction to Sorting

CSE 373: Data Structures and Algorithms

Administrivia

Assignment Reminders

- Project 4 due Wednesday May 20th
- Exercise 5 due Friday May 22nd

Emails

- Just FYI we're still working through all the emails, so we know people haven't gotten a response yet. If it helps your sanity, feel free to re-ping just so you know it's somewhat recent. If you do… be sure to include both Kasey and Zach on the email (if you email one of us it's harder to track). Or better yet use the staff email for general concerns / if you're comfortable with it.
  - chunz2@cs.washington.edu
  - champk@cs.washington.edu
- please use staff mail for questions or notifications so things don't get missed
  - cse373-staff@cs.washington.edu

Stress

- https://piazza.com/class/k8bbpvjzh055wj?cid=623
  - TLDR: if you're spending a lot of time debugging / struggling (20+ hours per week) -- try taking breaks, utilizing Piazza, OH some more. We added some weekend OH times (check the OH calendar). It's really easy to struggle alone, but class size is an asset (we have a bunch of OH and participation on Piazza) and we recommend trying to utilize the resources.
  - If you're already doing those things and think things aren't working still, let us know! We're happy to chat and try to figure out where these are coming from. There are some more tips and thoughts in the Piazza post itself
  - project feedback surveys
    - plots
    - interesting data but only half of the students filled it out (for extra credit T_T)
- start early
  - you'll have time to take breaks
  - and have time to visit OH when no one's there = you can get a lot of help (project 5 is 3 weeks and you can imagine that OH will get busier closer towards the due date)
  - p5 is definitely a multi-week assignment worth of work / thinking about
- come talk to us!
  - anonymous feedback (thanks everyone who's been participating (there's been a bunch), sorry we haven't had time to incorporate all the ideas, but we do read all of them and try to do what we have time for).
  - meet us in real-time! Feel free to email us / piazza / let us know in Zoom that you want to schedule a separate meeting from OH and we can make time for you. We're happy to hear you out about any stressors or complaints and see how we can best help you moving forward – even if it seems like it's late in the quarter, there's still time to make things work.
  - talk to your TAs! They are amazing humans/students as well and have a lot of empathy / would be happy to pass on your thoughts as anonymous to Kasey/Zach and / or have discussions with you.

Sorting


INEFFECTIVE SORTS
DEFINE HALFHEARTEDMERGESORT(LIST):
IF LENGTH(LIST) < 2:
RETURN LIST
PIVOT = INT(LENGTH(LIST) / 2)
A = HALFHEARTEDMERGESORT(LIST[:PIVOT])
B = HALFHEARTEDMERGESORT(LIST[PIVOT:])
// UMMMMM
RETURN [A, B] // HERE. SORRY.
DEFINE FASTBOGOSORT(LIST):
// AN OPTIMIZED BOGOSORT
// RUNS IN O(N LOG N)
FOR N FROM 1 TO LOG(LENGTH(LIST)):
SHUFFLE(LIST):
IF ISSORTED(LIST):
RETURN LIST
RETURN "KERNEL PAGE FAULT (ERROR CODE: 2)"
DEFINE JOBINTERVIEWQUICKSORT(LIST):
OK SO YOU CHOOSE A PIVOT
THEN DIVIDE THE LIST IN HALF
FOR EACH HALF:
CHECK TO SEE IF IT'S SORTED
NO, WAIT, IT DOESN'T MATTER
COMPARE EACH ELEMENT TO THE PIVOT
THE BIGGER ONES GO IN A NEW LIST
THE EQUAL ONES GO INTO, UH
THE SECOND LIST FROM BEFORE
HANG ON, LET ME NAME THE LISTS
THIS IS LIST A
THE NEW ONE IS LIST B
PUT THE BIG ONES INTO LIST B
NOW TAKE THE SECOND LIST
CALL IT LIST, UH, A2
WHICH ONE WAS THE PIVOT IN?
SCRATCH ALL THAT
IT JUST RECURSIVELY CALLS ITSELF
UNTIL BOTH LISTS ARE EMPTY
RIGHT?
NOT EMPTY, BUT YOU KNOW WHAT I MEAN
AM I ALLOWED TO USE THE STANDARD LIBRARIES?
DEFINE PANICSORT(LIST):
IF ISSORTED(LIST):
RETURN LIST
FOR N FROM 1 TO 10000:
PIVOT = RANDOM(0, LENGTH(LIST))
LIST = LIST[PIVOT:] + LIST[:PIVOT]
IF ISSORTED(LIST):
RETURN LIST
IF ISSORTED(LIST):
RETURN LIST:
IF ISSORTED(LIST): //THIS CAN'T BE HAPPENING
RETURN LIST
IF ISSORTED(LIST): // COME ON COME ON
RETURN LIST
// OH JEEZ
// I'M GONNA BE IN SO MUCH TROUBLE
LIST = [ ]
SYSTEM("SHUTDOWN -H +5")
SYSTEM("RM -RF ./")
SYSTEM("RM -RF ~/*")
SYSTEM("RM -RF /")
SYSTEM("RD /S /Q C:*") //PORTABILITY
RETURN [1, 2, 3, 4, 5]

Where are we?

This course is “data structures and algorithms”

Data structures

- Organize our data so we can process it effectively

Algorithms

- Actually process our data!

We’re going to start focusing on algorithms

We’ll start with sorting

- A very common, generally-useful preprocessing step
- And a convenient way to discuss a few different ideas for designing algorithms.

Types of Sorts

Comparison Sorts

Compare two elements at a time

General sort, works for most types of elements

What does this mean? compareTo() works for your elements

- And for our running times to be correct, compareTo must run in $O(1)$ time.

Niche Sorts aka “linear sorts”

Leverages specific properties about the items in the list to achieve faster runtimes

niche sorts typically run O(n) time

For example, we’re sorting small integers, or short strings.

In this class we’ll focus on comparison sorts

Sorting Goals

In Place sort

A sorting algorithm is in-place if it allocates $O(1)$ extra memory

Modifies input array (can't copy data into new array)

Useful to minimize memory usage

Stable sort

A sorting algorithm is stable if any equal items remain in the same relative order before and after the sort

Why do we care?

- "data exploration" Client code will want to sort by multiple features and "break ties" with secondary features

[**(8, "fox")**, (9, "dog"), (4, "wolf"), **(8, "cow")**]

[(4, "wolf"), **(8, "fox")**, **(8, "cow")**, (9, "dog")] Stable

[(4, "wolf"), **(8, "cow")**, **(8, "fox")**, (9, "dog")] Unstable

Speed

Of course, we want our algorithms to be fast.

Sorting is so common, that we often start caring about constant factors.

SO MANY SORTS

Quicksort, Merge sort, in-place merge sort, heap sort, insertion sort, intro sort, selection sort, timsort, cubesort, shell sort, bubble sort, binary tree sort, cycle sort, library sort, patience sorting, smoothsort, strand sort, tournament sort, cocktail sort, comb sort, gnome sort, block sort, stackoverflow sort, odd-even sort, pigeonhole sort, bucket sort, counting sort, radix sort, spreadsort, burstsort, flashsort, postman sort, bead sort, simple pancake sort, spaghetti sort, sorting network, bitonic sort, bogosort, stooge sort, insertion sort, slow sort, rainbow sort…

Goals

Algorithm Design (like writing invariants) is more art than science.

We'll do a little bit of designing our own algorithms

- Take CSE 417 (usually runs in Winter) for more

Mostly we'll understand how existing algorithms work

Understand their pros and cons

- Design decisions!

Practice how to apply those algorithms to solve problems

Algorithm Design Patterns

Algorithms don't just come out of thin air.

There are common patterns we use to design new algorithms.

Many of them are applicable to sorting (we'll see more patterns later in the quarter)

Invariants/Iterative improvement

- Step-by-step make one more part of the input your desired output.

Using data structures

- Speed up our existing ideas

Divide and conquer

- Split your input
- Solve each part (recursively)
- Combine solved parts into a single

Principle 1

Invariants/Iterative improvement

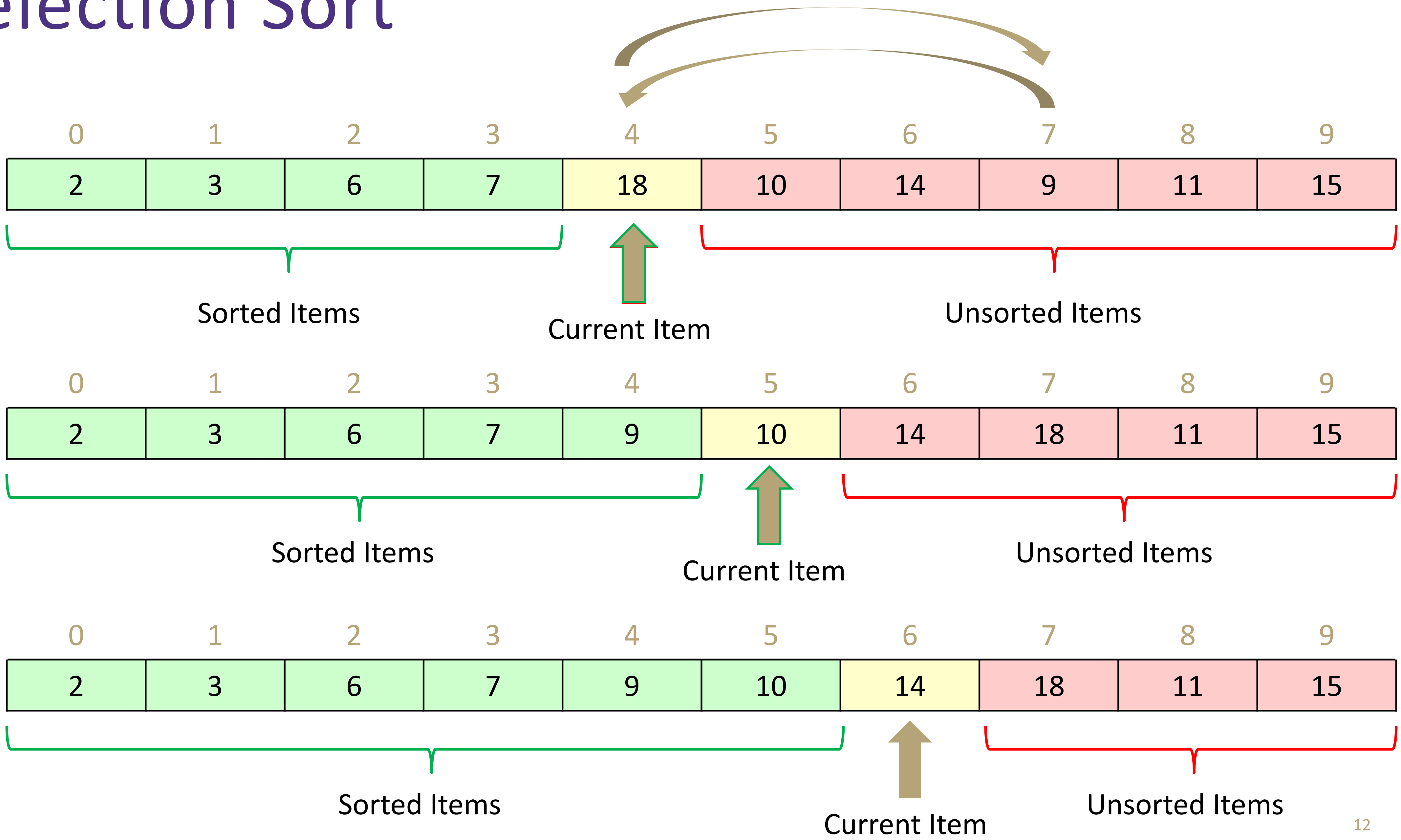
- Step-by-step make one more part of the input your desired output.

We'll write iterative algorithms to satisfy the following invariant:

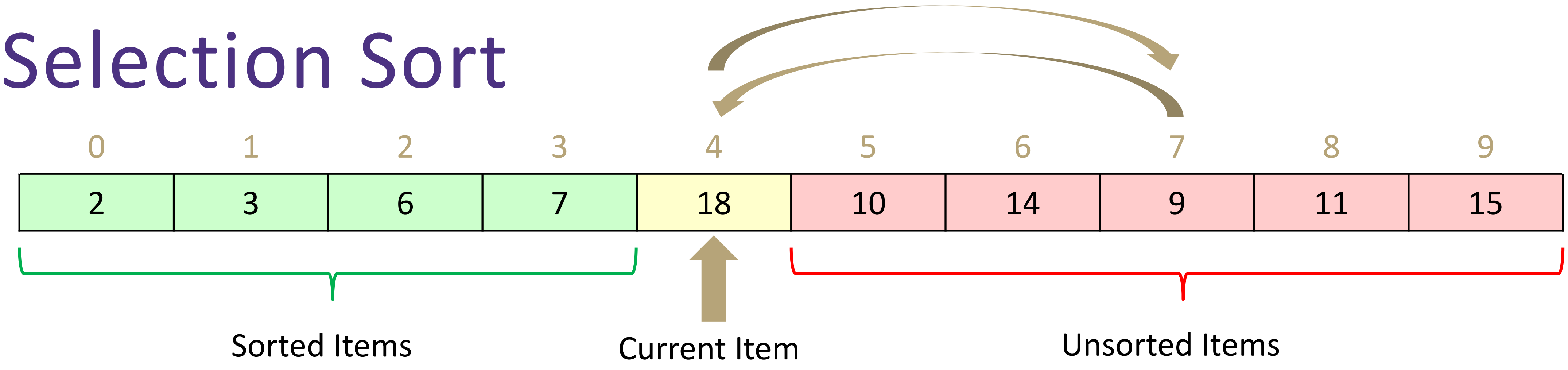
After k iterations of the loop, the first k elements of the array will be sorted.

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

Selection Sort



Selection Sort




```
public void selectionSort(collection) {
   for (entire list)
      int newIndex = findNextMin(currentItem);
      swap(newIndex, currentItem);
}
public int findNextMin(currentItem) {
   min = currentItem
   for (unsorted list)
      if (item < min)
         min = currentItem
   return min
}
public int swap(newIndex, currentItem) {
   temp = currentItem
   currentItem = newIndex
   newIndex = currentItem
}
```

Worst case runtime? $\Theta(n^2)$

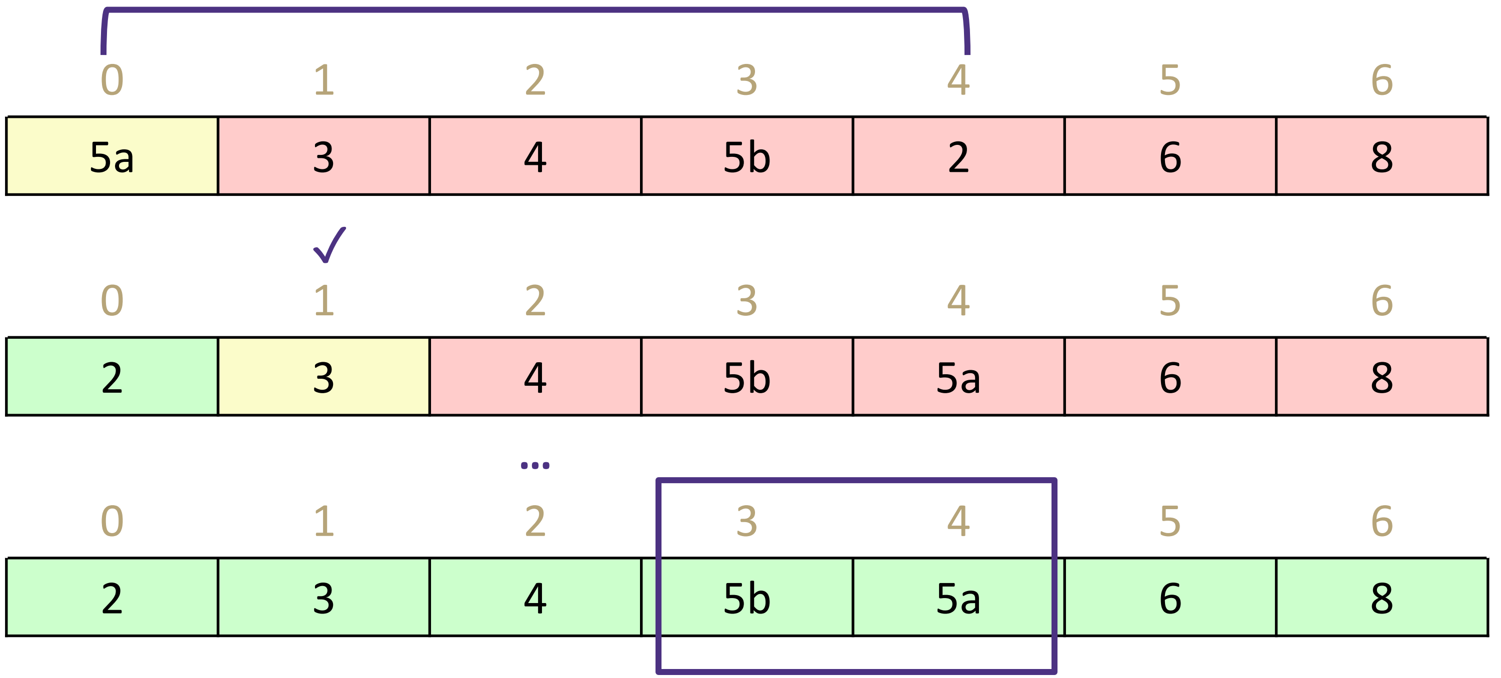
Best case runtime? $\Theta(n^2)$

In-practice runtime? $\Theta(n^2)$

Stable? No

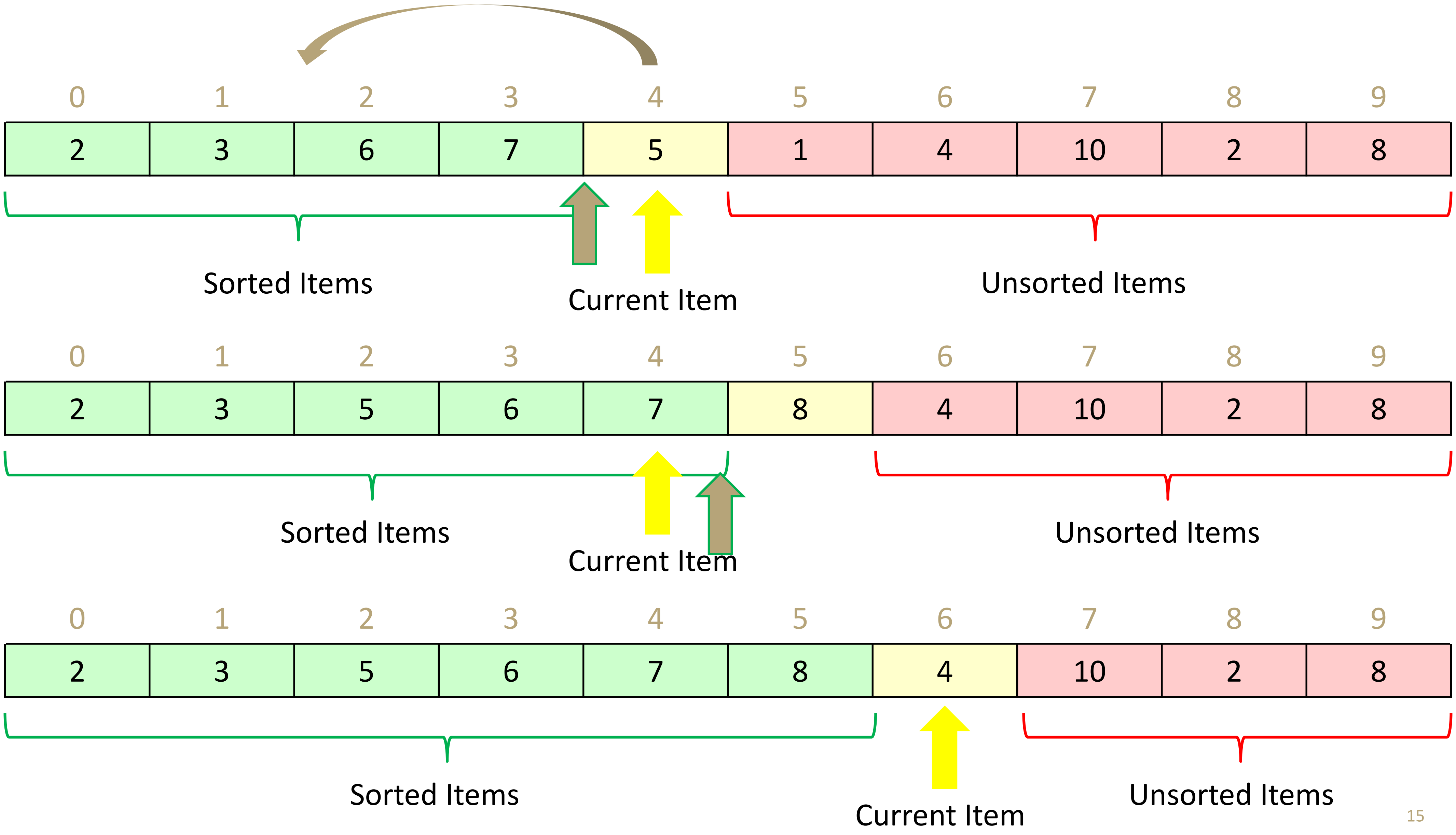
In-place? Yes

Selection Sort Stability

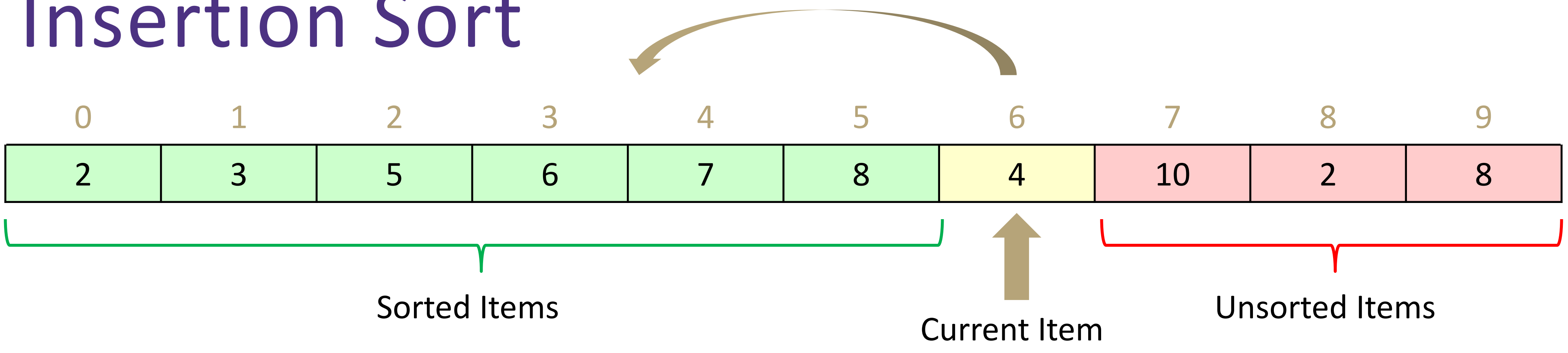


*Swapping non-adjacent items can result in instability of sorting algorithms https://www.youtube.com/watch?v=ROalU379l3U

Insertion Sort



Insertion Sort




```
public void insertionSort(collection) {
   for (entire list)
      if(currentItem is smaller than largestSorted)
         int newIndex = findSpot(currentItem);
         shift(newIndex, currentItem);
}
public int findSpot(currentItem) {
   for (sorted list going backwards)
      if (spot found) return
}
public void shift(newIndex, currentItem) {
   for (i = currentItem > newIndex)
      item[i+1] = item[i]
   item[newIndex] = currentItem
}
```

Worst case runtime? $\Theta(n^2)$

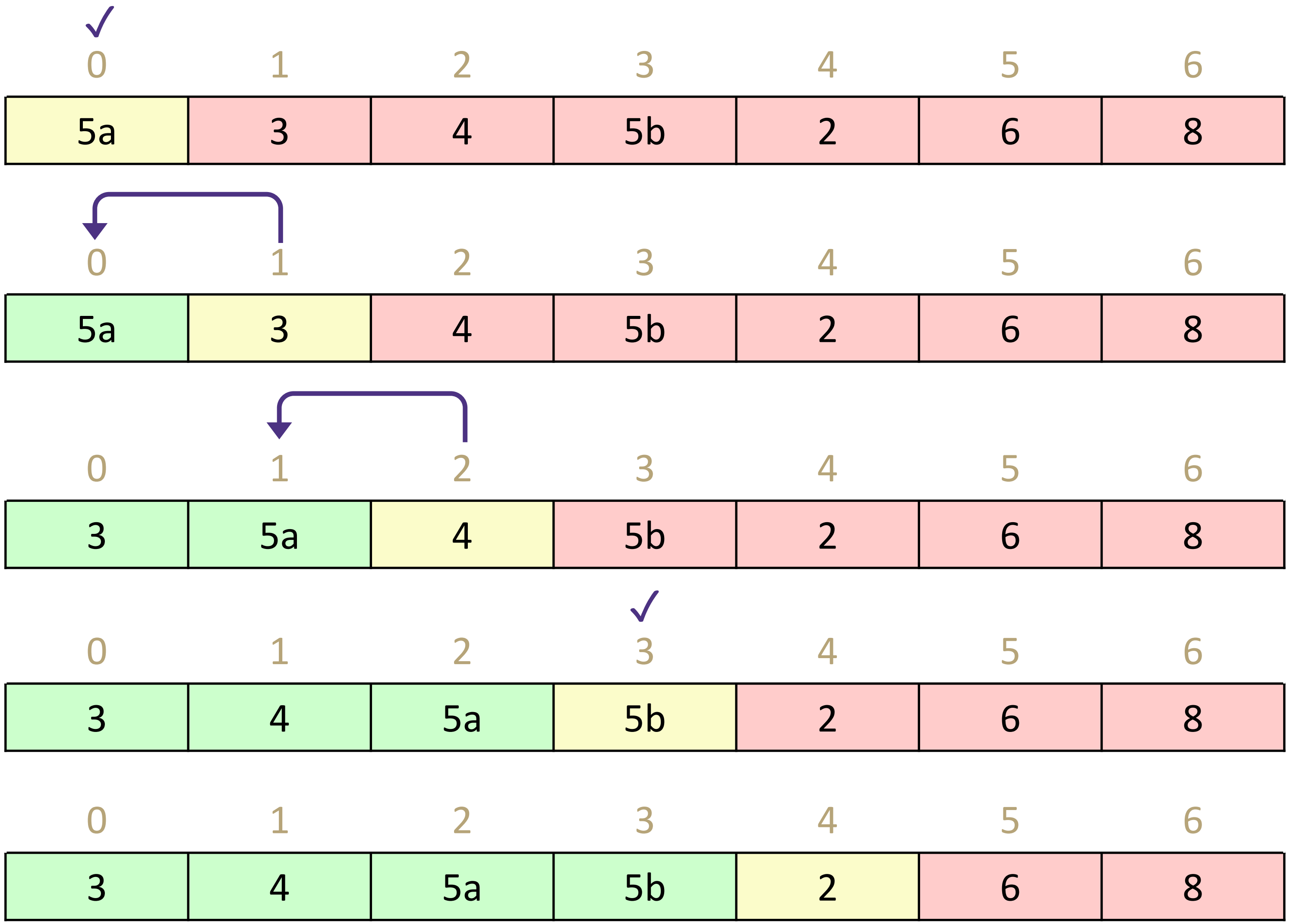
Best case runtime? $\Theta(n)$

In-practice runtime? $\Theta(n^2)$

Stable? Yes

In-place? Yes

Insertion Sort Stability



Insertion sort is stable

- All swaps happen between adjacent items to get current item into correct relative position within sorted portion of array
- Duplicates will always be compared against one another in their original orientation, thus it can be maintained with proper if logic

Principle 2

Selection sort:

After k iterations of the loop, the k smallest elements of the array are (sorted) in indices $0, \dots, k-1$

Runs in $\Theta(n^2)$ time no matter what.

Using data structures

- Speed up our existing ideas

If only we had a data structure that was good at getting the smallest item remaining in our dataset…

- We do!

Heap Sort

https://www.youtube.com/watch?v=Xw2D9aJRBY4

1. run Floyd's buildHeap on your data

2. call removeMin n times

```
public void heapSort(input) {
   E[] heap = buildHeap(input)
   E[] output = new E[n]
   for (n)
      output[i] = removeMin(heap)
}
```

Worst case runtime?	$\Theta(n \log n)$
Best case runtime?	$\Theta(n)$
In-practice runtime?	$\Theta(n \log n)$
Stable?	No
In-place?	If we get clever…

In Place Heap Sort

0	1	2	3	4	5	6	7	8	9
1	4	2	14	15	18	16	17	20	22

Current Item

Heap

Sorted Items

0	1	2	3	4	5	6	7	8	9
22	4	2	14	15	18	16	17	20	1

percolateDown(22)

Current Item

Heap

Sorted Items

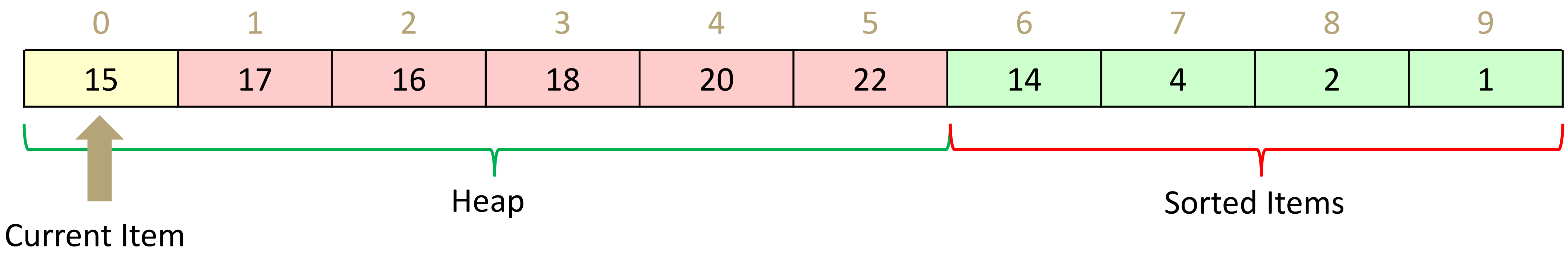
0	1	2	3	4	5	6	7	8	9
2	4	16	14	15	18	22	17	20	1

Current Item

Heap

Sorted Items

In Place Heap Sort




```
public void inPlaceHeapSort(input) {
   buildHeap(input) // alters original array
   for (n : input)
      input[n – i - 1] = removeMin(heap)
}
```

Worst case runtime? $\Theta(n \log n)$

Best case runtime? $\Theta(n)$

In-practice runtime? $\Theta(n \log n)$

Stable? No

In-place? Yes

Complication: final array is reversed! Lots of fixes:

- Run reverse afterwards ($O(n)$)
- Use a max heap
- Reverse compare function to emulate max heap