Topic 12.2: Analysis of Quicksort

Revisiting Sorting One Last Time

© Lauren Himbeault 2025

By the end of this video:

- Understand how quicksort works conceptually.
- Be able to follow and understand Java code for quicksort.
- Recognize how pivot choice affects running time.
- Connect the idea of a good vs. bad pivot to the overall efficiency of sorting.

© Lauren Himbeault 2025

A reminder on sorting and what we know

- Insertion sort
  - like shelving libraries books
- Selection sort
  - drafting players for your basketball team
- Mergesort
  - Organizing puzzle pieces by sections

© Lauren Himbeault 2025

and now...

- Insertion sort
  - like shelving libraries books
- Selection sort
  - drafting players for your basketball team
- Mergesort
  - Organizing puzzle pieces by sections
- Quick Sort
  - time for an analogy

© Lauren Himbeault 2025

QuickSort: in a nutshell Part 1

- Sorting your closet to ensure your favorite items are the easiest to access.

© Lauren Himbeault 2025

QuickSort: in a nutshell Part 1

- Sorting your closet to ensure your favorite items are the easiest to access.

Step 1. Pick an item
Step 2. Sort all your clothes into two piles
- ones you like **less** than that item
- ones your like **as much or more** than that item

© Lauren Himbeault 2025

QuickSort: in a nutshell Part 1

- Sorting your closet to ensure your favorite items are the easiest to access.

Step 1. Pick an item

Step 2. Sort all your clothes into two piles

- ones you like **less** than that item
- ones your like **as much or more** than that item

Step 3. For each pile you just created

- perform steps 1 and 2 again

Step 4... Do this for each pile you create until....

© Lauren Himbeault 2025

QuickSort: in a nutshell Part 1

- Sorting your closet to ensure your favorite items are the easiest to access.

Step 1. Pick an item
Step 2. Sort all your clothes into two piles
- ones you like **less** than that item
- ones your like **as much or more** than that item
Step 3. For each pile you just created
- perform steps 1 and 2 again
Step 4... Do this for each pile you create until....
Step 5. Hit a pile that cannot be further divided
Step 6. Recombine everything
- in the reverse order (starting with the last pile you touched), recombine your clothes
- Less-liked clothes first,
- your favorite item
- the more-liked clothes.

© Lauren Himbeault 2025

QuickSort Visually

Closet

© Lauren Himbeault 2025

QuickSort Visually

Closet

© Lauren Himbeault 2025

QuickSort Visually

Closet

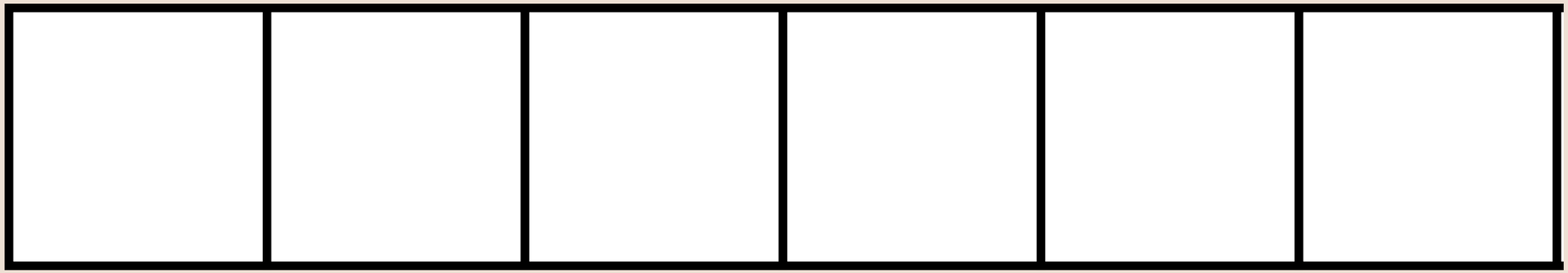
Likes less

Likes more

© Lauren Himbeault 2025

QuickSort Visually

Closet



Likes less

Likes more

© Lauren Himbeault 2025

QuickSort Visually

Closet

Likes less

Likes more

Likes less

Likes more

© Lauren Himbeault 2025

QuickSort Visually

Closet

Likes less

Likes more

Likes less

Likes more

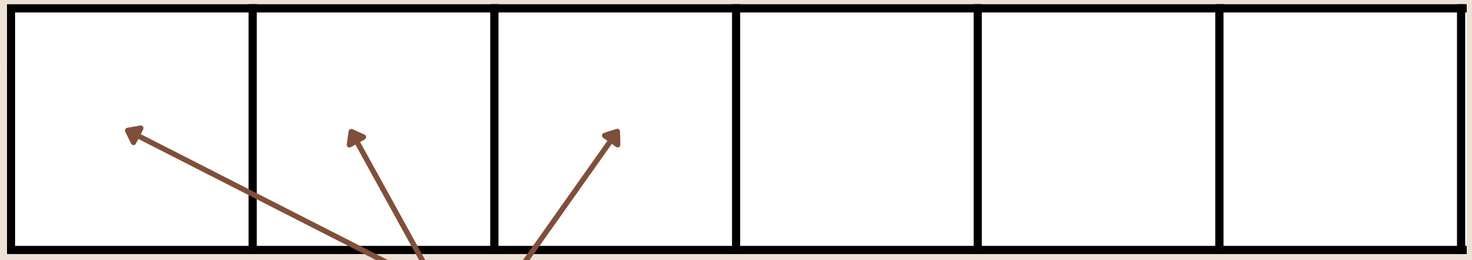
© Lauren Himbeault 2025

QuickSort Visually

Likes less

Likes more

Closet

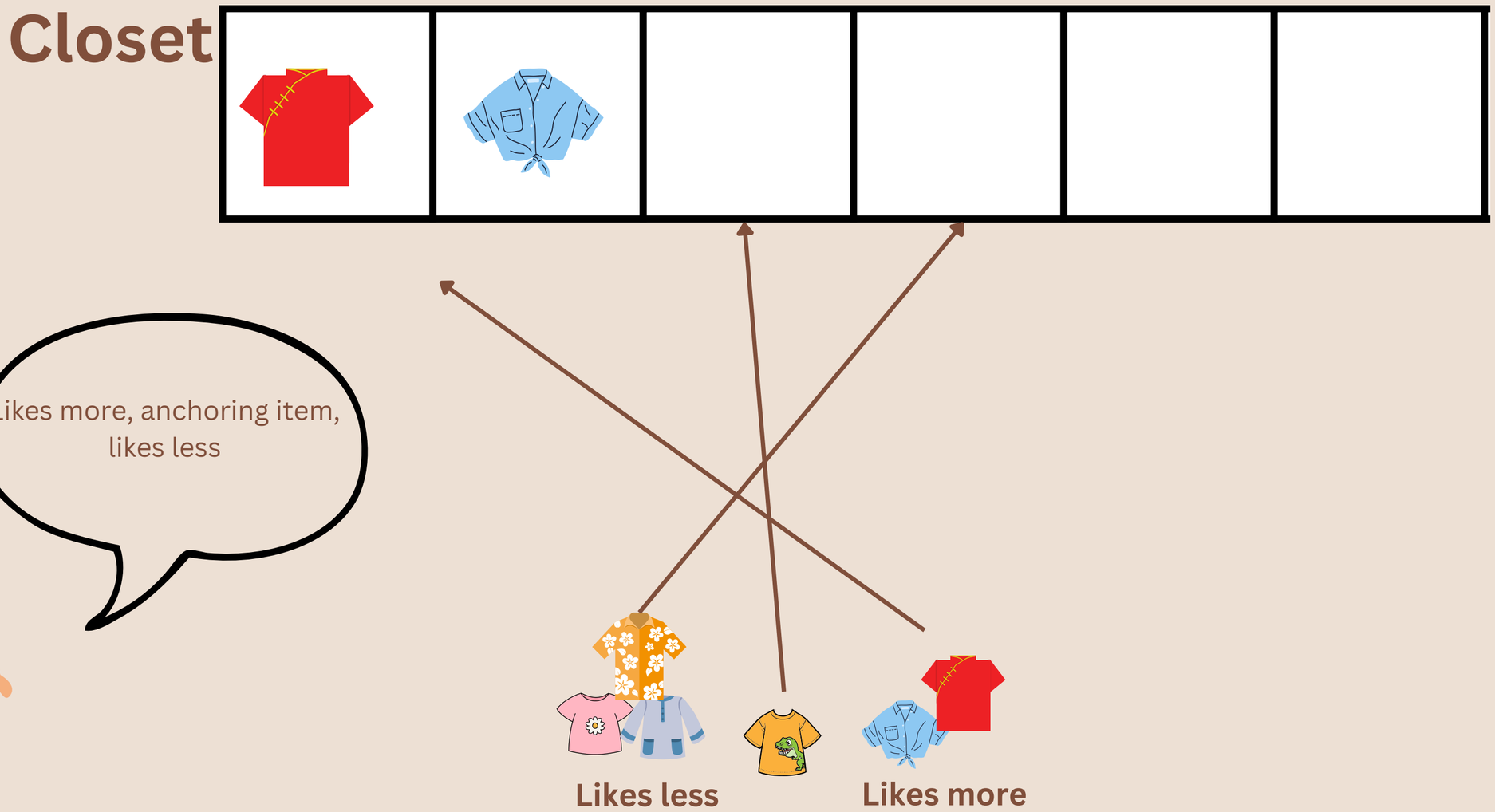


Likes less

Likes more

© Lauren Himbeault 2025

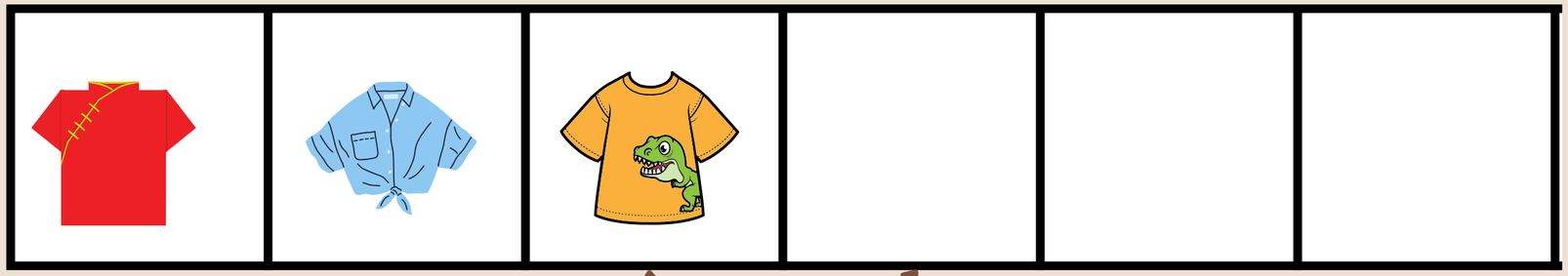
QuickSort Visually

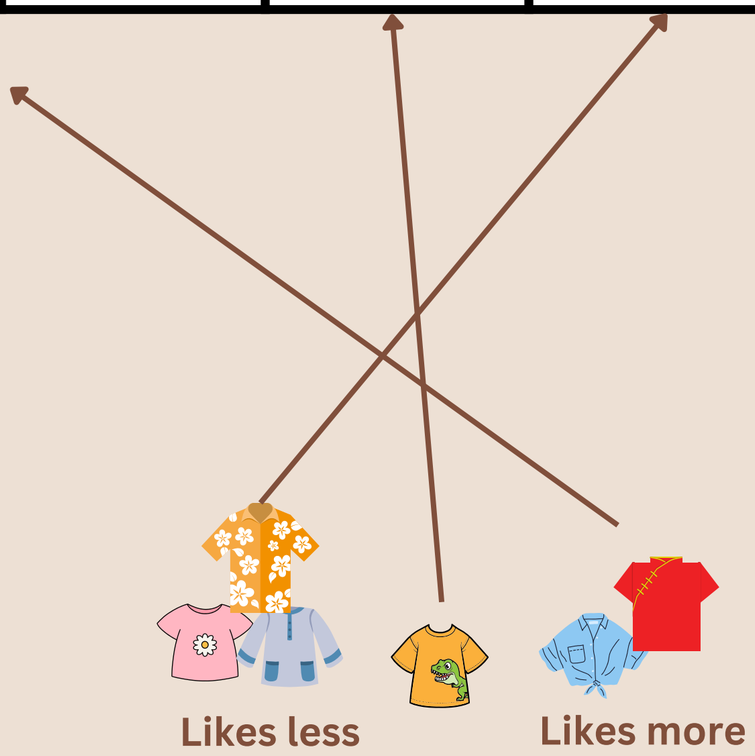


© Lauren Himbeault 2025

QuickSort Visually

Closet





Likes less

Likes more

© Lauren Himbeault 2025

QuickSort Visually

Closet

Likes less

Likes more

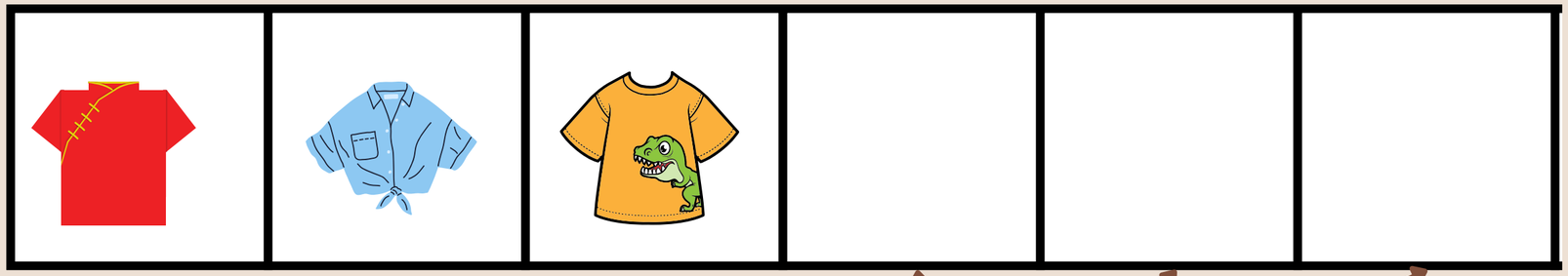
Likes less

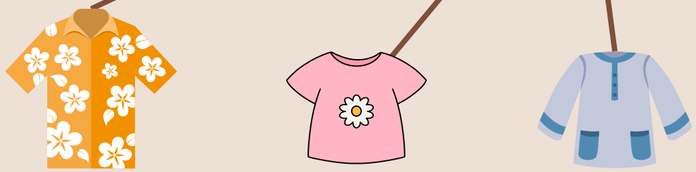
Likes more

© Lauren Himbeault 2025

QuickSort Visually

Closet



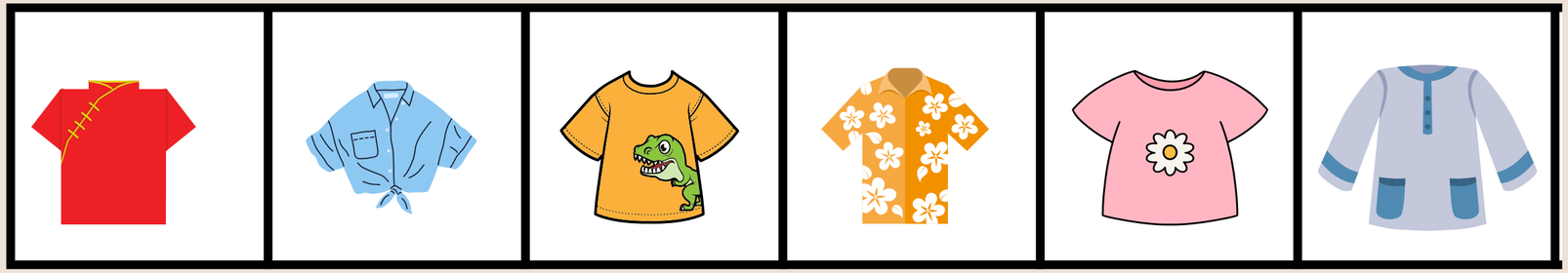


Likes less **Likes more**

© Lauren Himbeault 2025

QuickSort Visually

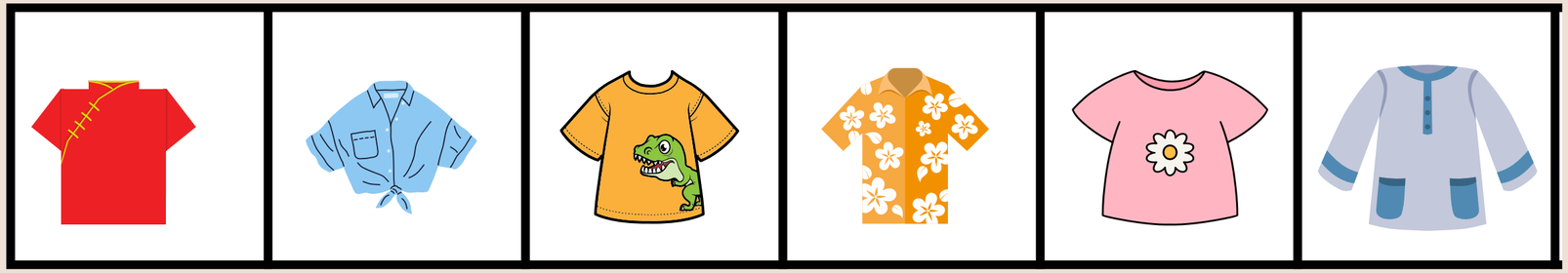
Closet



© Lauren Himbeault 2025

QuickSort Visually

Closet



© Lauren Himbeault 2025

QuickSort

What are the “big facts” for this process?

- If a pile is empty/has 1 element, it can’t be further divided
- We ALWAYS processed our shirts in the order of:
  - divide around a shirt
  - check out the favourites pile
  - move the anchor shirt to the closet
  - check out the less favourites pile

© Lauren Himbeault 2025

QuickSort

What are the “big facts” for this process?

- If a pile is empty/has 1 element, it can’t be further divided **(base case, 0 shirts or 1 anchor shirt and that’s it, no work to be done here)**
- We ALWAYS processed our shirts in the order of:
  - divide around a shirt **separate into favourites (left) and less favourites (right)**
  - check out the favourites pile **look at the left pile and process it (recursive like merge sort)**
  - move the anchor shirt to the closet **Place the anchor shirt in it’s new home**
  - check out the less favourites pile **look at the right pile and process it (recursive like merge sort)**

© Lauren Himbeault 2025

```java
public class QuickSort {
    public static void quicksort(int[] array, int low, int high) {
        if (low < high) {
            int pivotIndex = partition(array, low, high);
            quicksort(array, low, pivotIndex - 1);
            quicksort(array, pivotIndex + 1, high);
        }
    }

    private static int partition(int[] array, int low, int high) {
        int pivot = array[high]; // Choosing the last element as pivot
        int i = low - 1;
        for (int j = low; j < high; j++) {
            if (array[j] <= pivot) {
                i++;
                swap(array, i, j);
            }
        }
        swap(array, i + 1, high); // Place pivot in the correct spot
        return i + 1;
    }

    private static void swap(int[] array, int i, int j) {
        int temp = array[i];
        array[i] = array[j];
        array[j] = temp;
    }
}
```

© Lauren Himbeault 2025

Choosing your Pivot (anchor shirt)

The pivot choice makes a BIG difference

Let’s see why….

© Lauren Himbeault 2025

Choosing your Pivot (anchor shirt)

The pivot choice makes a BIG difference
Let’s see why….

7	3	1	8	0	10

I grab the one closest to me *(easiest)*

Swap Pivot to Middle
Move everything around it →

© Lauren Himbeault 2025

Choosing your Pivot (anchor shirt)

The pivot choice makes a BIG difference
Let’s see why….

7	3	1	8	0	10

I grab the one closest to me *(easiest)*

- *swap pivot to end*
- *parition steps*
- *tracking partition index*
- *swap pivot to index*

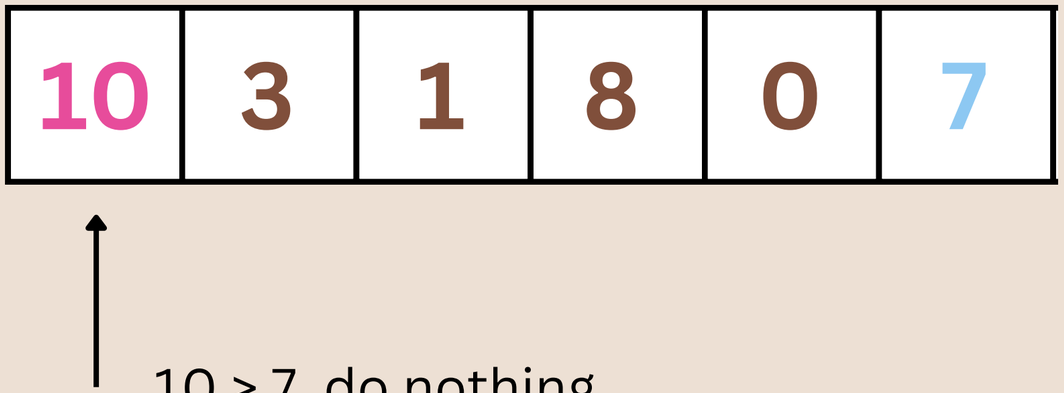
© Lauren Himbeault 2025

i = -1

- ***swap pivot to end***
- *partition steps*
- *tracking partition index*
- *swap pivot to i + 1*

© Lauren Himbeault 2025

i = -1



10 > 7, do nothing

- *swap pivot to end*
- ***partition steps***
- ***tracking partition index***
- *swap pivot to i + 1*

© Lauren Himbeault 2025

i = -1

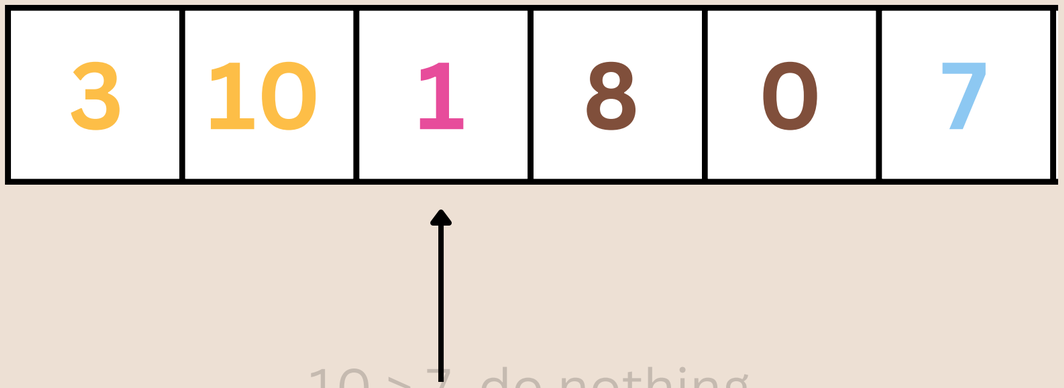
10 > 7, do nothing

3 < 7: increment i & swap with 10

- *swap pivot to end*
- ***partition steps***
- ***tracking partition index***
- *swap pivot to i + 1*

© Lauren Himbeault 2025

i = 0



- *swap pivot to end*
- ***partition steps***
- ***tracking partition index***
- *swap pivot to i + 1*

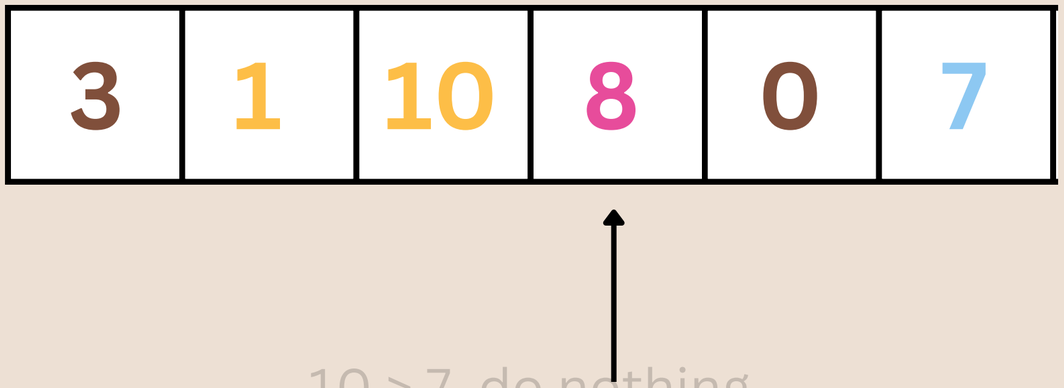
10 > 7, do nothing

3 < 7: increment i & swap with 10

1 < 7: increment i & swap with 10

© Lauren Himbeault 2025

i = 1



- *swap pivot to end*
- ***partition steps***
- ***tracking partition index***
- *swap pivot to i + 1*

10 > 7, do nothing

3 < 7: increment i & swap with 10

1 < 7: increment i & swap with 10

8 > 7, do nothing

© Lauren Himbeault 2025

i = 1

- *swap pivot to end*
- ***partition steps***
- ***tracking partition index***
- *swap pivot to i + 1*

10 > 7, do nothing

3 < 7: increment i & swap with 10

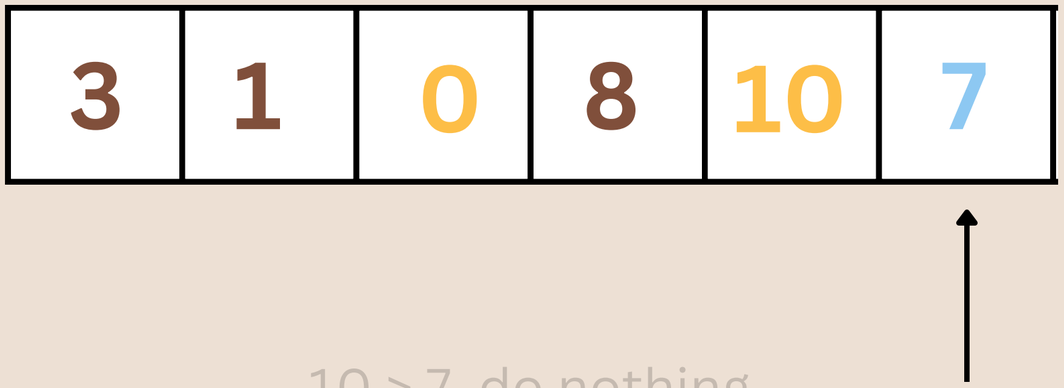
1 < 7: increment i & swap with 10

8 > 7, do nothing

0 < 7: increment i & swap with 10

© Lauren Himbeault 2025

i = 2



- *swap pivot to end*
- ***partition steps***
- ***tracking partition index***
- *swap pivot to i + 1*

10 > 7, do nothing

3 < 7: increment i & swap with 10

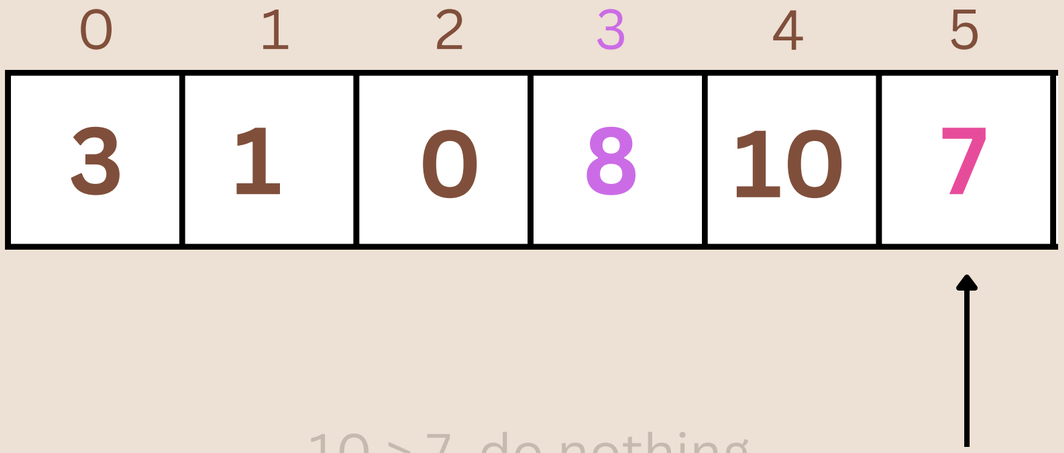
1 < 7: increment i & swap with 10

8 > 7, do nothing

0 < 7: increment i & swap with 10

Now my iterator index is pointing at my last element (pivot), so I stop

© Lauren Himbeault 2025



- *swap pivot to end*
- *partition steps*
- *tracking partition index*
- ***swap pivot to index***

i = 2

10 > 7, do nothing

3 < 7: increment i & swap with 10

1 < 7: increment i & swap with 10

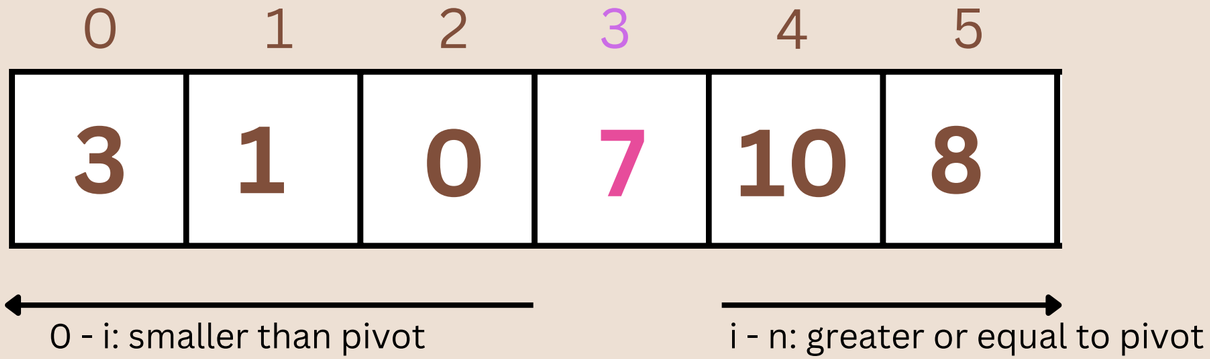
8 > 7, do nothing

0 < 7: increment i & swap with 10

Now my iterator index is pointing at my last element (pivot), so I stop

Swap my pivot into my **i**th spot

© Lauren Himbeault 2025



- *swap pivot to end*
- *partition steps*
- *tracking partition index*
- ***swap pivot to index***

10 > 7, do nothing

3 < 7: increment i & swap with 10

1 < 7: increment i & swap with 10

8 > 7, do nothing

0 < 7: increment i & swap with 10

Now my iterator index is pointing at my last element (pivot), so I stop

Swap my pivot into my **i**th spot

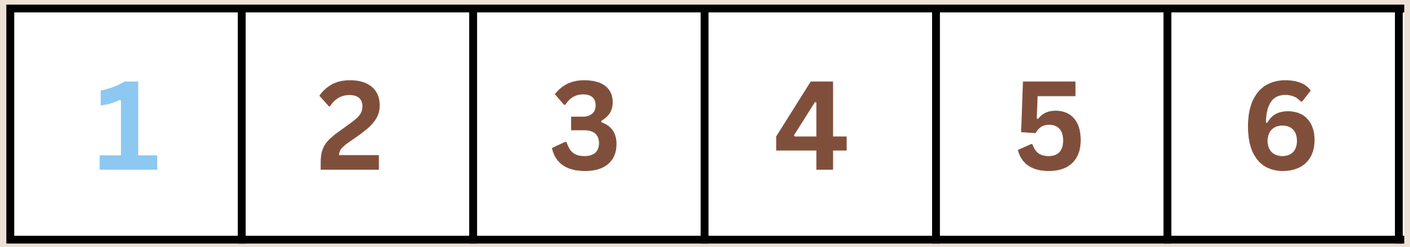
Now we do it again on each sub array

Watch a full visualization here, select QUI in the nav menu

***note: they pick the first element as the pivot and leave it in front rather than doing an initial swap**

© Lauren Himbeault 2025

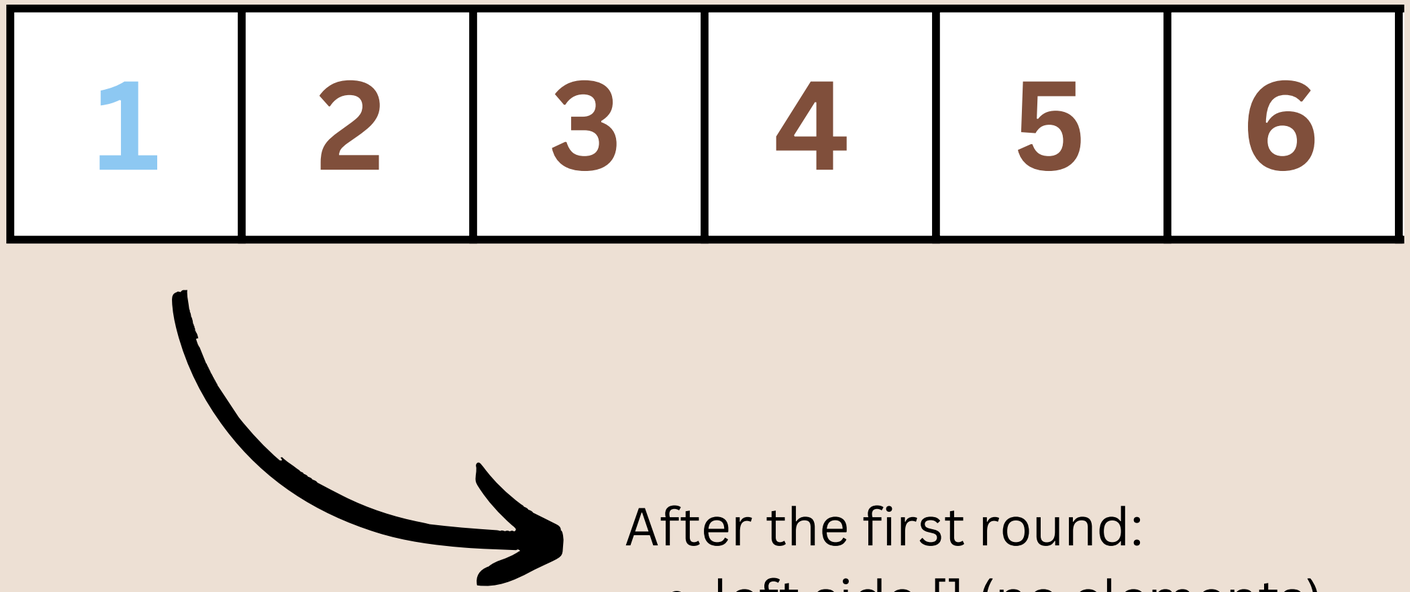
How Bad Can a Bad Pivot Be?



Following suit from before: I choose the first element as my pivot

© Lauren Himbeault 2025

How Bad Can a Bad Pivot Be?



After the first round:

- left side [] (no elements)
- right side [2,3,4,5,6] (all the elements)

© Lauren Himbeault 2025

How Bad Can a Bad Pivot Be?

After the first round:

- left side [] (no elements)
- right side [2,3,4,5,6] (all the elements)

If our pivot doesn't split the array (roughly) in half, each element still ends up with n-1 comparisons (instead of logn like it should be, like merge sort)

© Lauren Himbeault 2025

Takeaway

- A good pivot splits the list roughly in half at each round, just like merge sort giving this a runtime of O(nlogn)

- A bad pivot barely splits the list, forcing each element to still be compared to every other element, giving this a runtime of $O(n^2)$

Pivot Choice GREATLY affects the runtime for this algorithm

© Lauren Himbeault 2025

Pause and Practice

Try to run through quicksort yourself. Draw it out and draw out each iteration. Then, try to code it up and see if you're right!
(Use the visualization tools to help check your answer before you copy/paste solution code)
[5, 9, 1, 3, 7, 2, 6]
Consider the following:

- If you were the picking the pivot, which number would you pick to split this list evenly?
- Could you pick a better pivot if you knew more about the data set? What would you need to know and how would it affect your choice?

© Lauren Himbeault 2025