App Academy

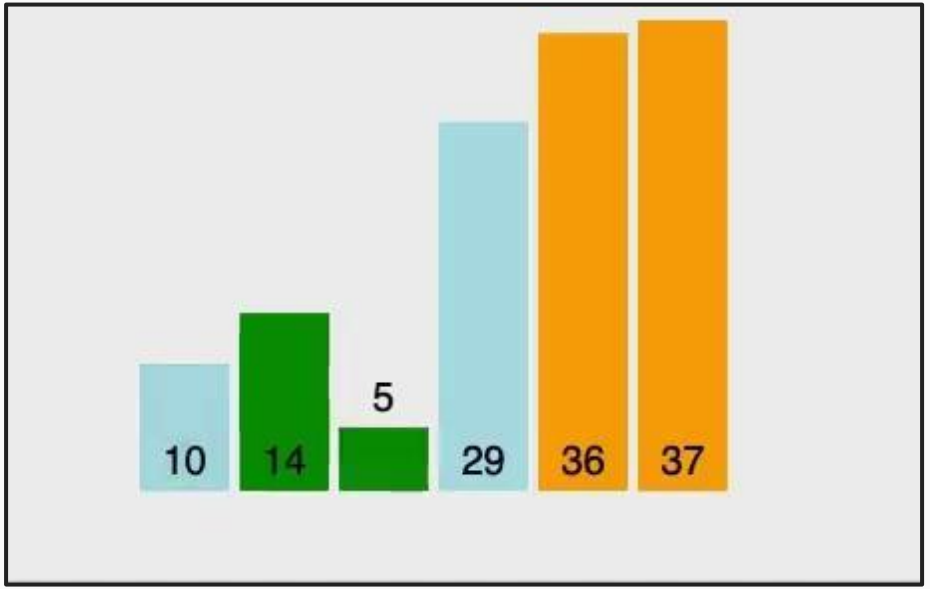
Sorting EOD

Whats the deal with all this sorting???

- Bubble Sort
- Selection Sort
- Insertions Sort
- Merge Sort
- Quick Sort
- ENOUGH WITH ALL THE SORTING ALREADY!!!
- Who's sorting all this stuff, and what are they sorting???

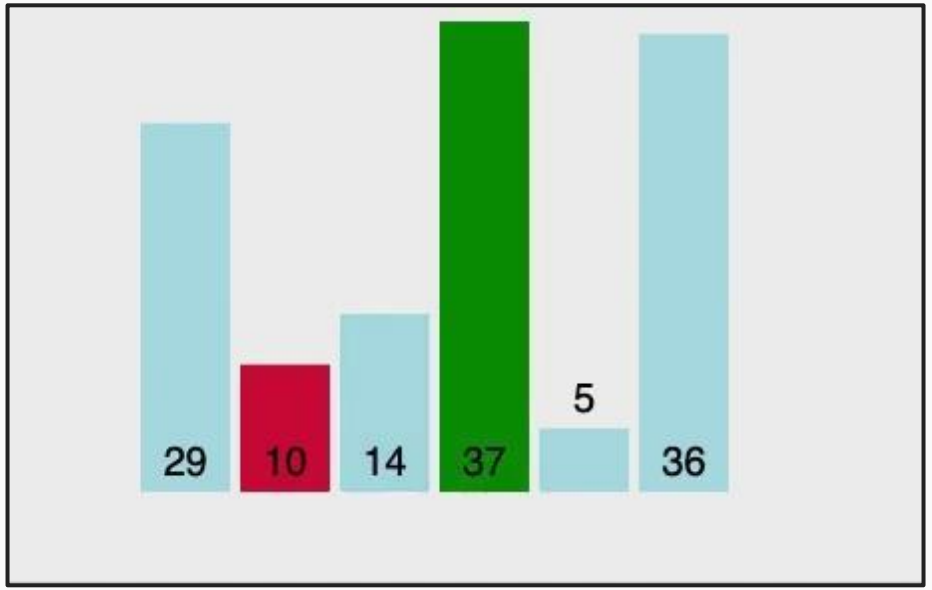
Bubble Sort

- Big 0:
  - Best Time: 0(n)
  - Average Time: $0(n^2)$
  - Worst Time: $O(n^2)$
  - Space: O(1)
- How it works: biggest values are moved to the end of the sort array until it is completely sorted
- When to use: To check if an array is sorted, or to sort if the array is nearly sorted



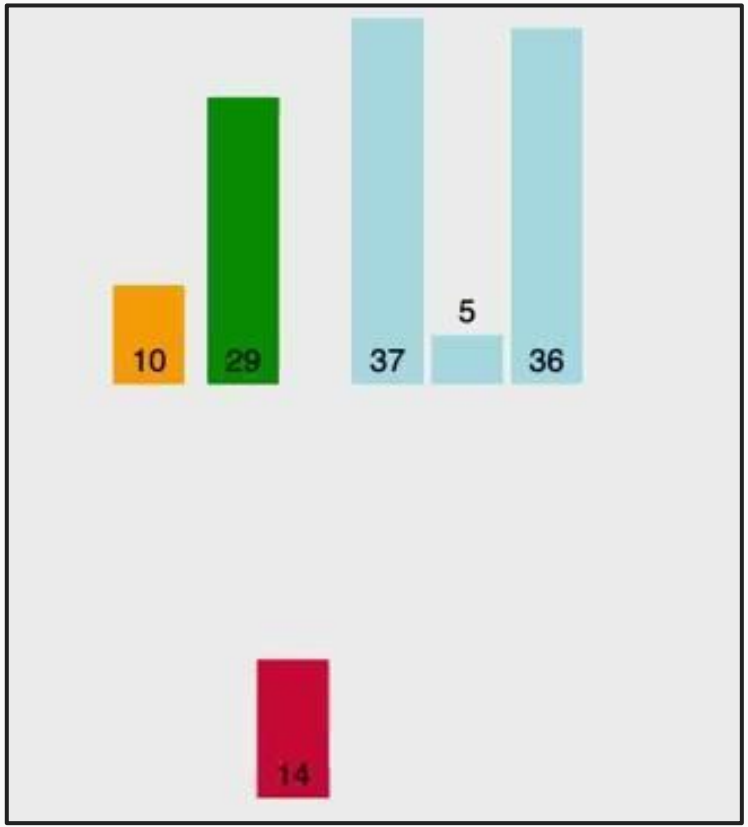
Selection Sort

- Big 0:
  - Best Time: $O(n^2)$
  - Average Time: $O(n^2)$
  - Worst Time: $O(n^2)$
  - Space: O(1)
- How it works: smallest values are moved to the beginning of the sort array until it is completely sorted
- When to use: Pretty much never



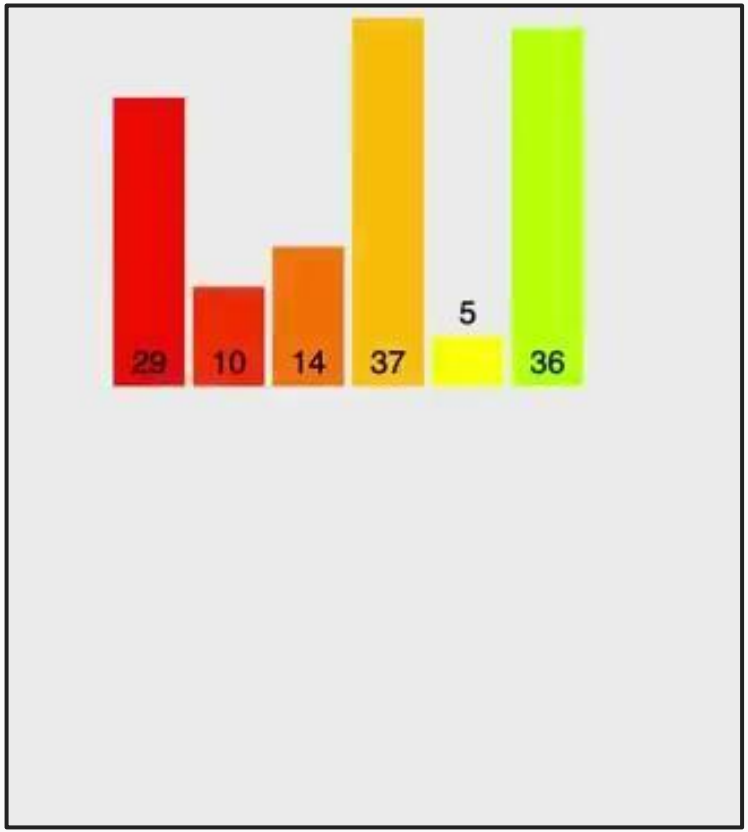
Insertion Sort

- Big 0:
  - Best Time: 0(n)
  - Average Time: $0(n^2)$
  - Worst Time: $O(n^2)$
  - Space: O(1)
- How it works: split array into sorted and not sorted sections, insert value by value into the sorted
- When to use: To check if an array is sorted, or to sort if the array is nearly sorted. Also best case for sorting a live data stream

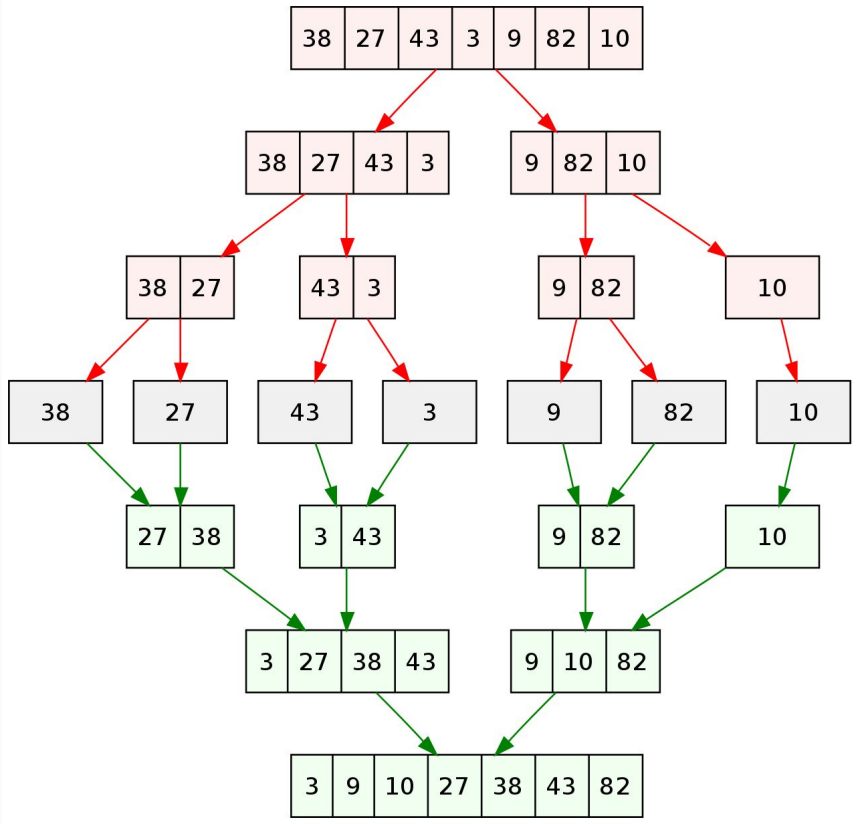


Merge Sort

- Big 0:
  - Best Time: 0(n log n)
  - Average Time: 0(n log n)
  - Worst Time: O(n log n)
  - Space: O(n log n)
- How it works: Divide and Conquer. Breaks down arrays until they are length 0 or 1, then merges them back together
- When to use: Often for its speed! Merge sort will be more more performant on larger arrays and data sets than Quick sort

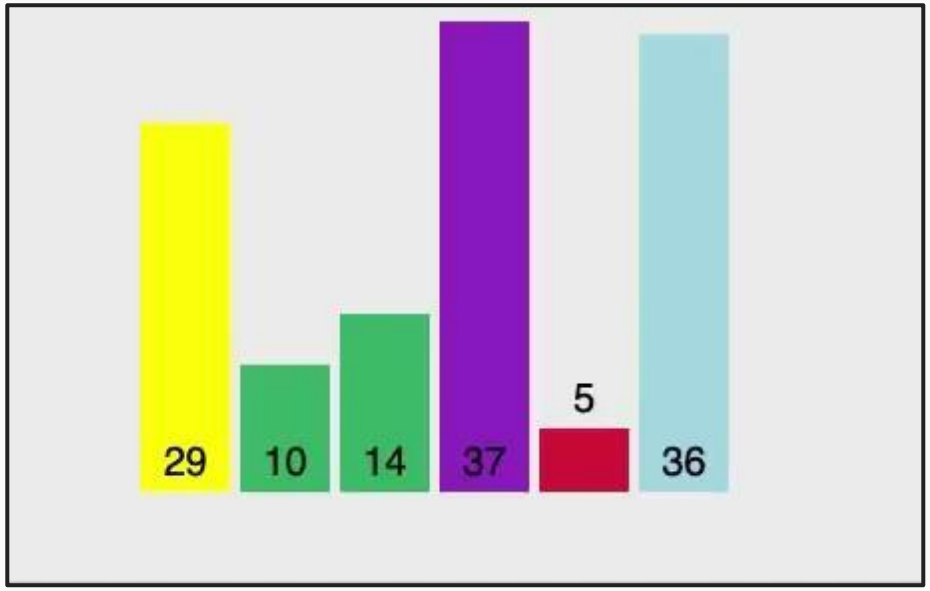


Merge Sort (cont)

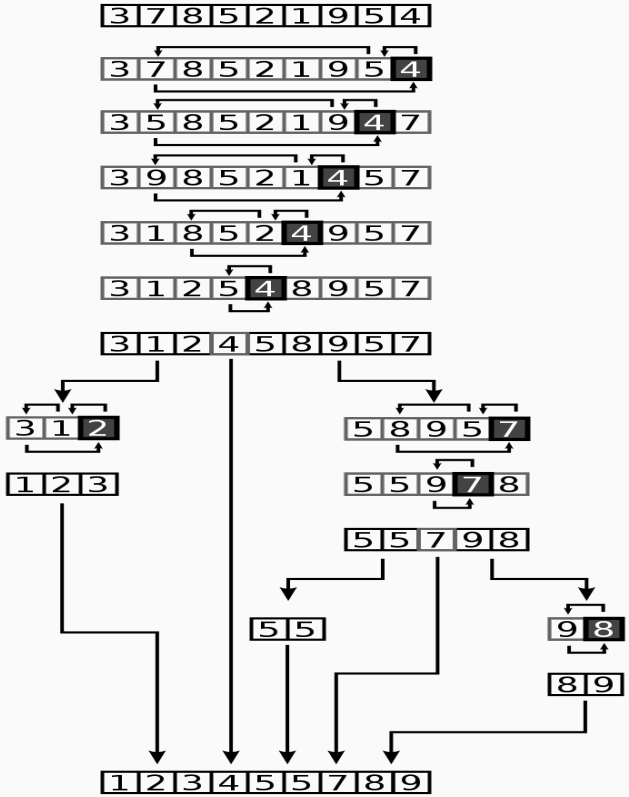


Quick Sort

- Big 0:
  - Best Time: 0(n log n)
  - Average Time: 0(n log n)
  - Worst Time: O(n^2)
  - Space: O(log n)
- How it works: Select a pivot point, and then partitions remaining values in > < sub arrays
- When to use: Quick sort is more performant than merge sort in the case of smaller array size or datasets



Quick Sort (cont)



Sorting IRL

- Array.sort( ) is based on the JS engine (per browser)
  - Chrome (V8): Quick Sort & Insertion Sort (node.js uses V8 as well)
  - IOS: Merge Sort
  - Mozilla: Merge Sort
- Python uses Timsort (mix of Merge Sort & Insertion Sort)
- C++ uses Introsort (mix of Quick Sort, Heap Sort and Insertion Sort)
- Ruby uses Quick Sort

Discussion: How do sorts work together?

Different sorts work better on different data, but we might not know what the data we are going to get will look like…

Is it sorted already, or nearly sorted?

How big a data set is it? Do we have 100's or 1,000,000,000's?

Any other special circumstances? (memory limit, or streamed data)

What if we used Bubble Sort to check if the data is already sorted, then use array.length to find out how much data, and then pick Merge Sort or Quick Sort based on the length?

Additional Resources

- When to use certain sorts: https://www.geeksforgeeks.org/when-to-use-each-sorting-algorithms/
- All sorts together with different data sets https://www.toptal.com/developers/sorting-algorithms
- Visualizing how each sort works (used for previous slides) https://visualgo.net/en