CS 1181
Week Eleven

Reese Hatfield

Sorting

Reese Hatfield

Sorting

- Comparable/ator
- Collections.sort()
  - Uses comparable/ator

- But how does it actually do that?

Sorting

- If I asked you to write a program that sorted a list, how would you do it?
  - Without Collections.sort()
  - How would you do this IRL?

Sorting

- First, let’s put this into more concise terms
- What are we actually trying to
- Writing the code for a process
  - Series of steps
  - General purpose for any list
    - Of any type <T>

Sorting

- This is called an *Algorithm*

- What makes something an algorithm?
  - Finite → can’t run forever
  - Correct → works for all input
  - Deterministic → not entirely* random

Sorting

- Specifically, we are writing a sorting algorithm
  - Sort of given list of type <T>
  - In reasonable time
  - Programatically
- IRL, you probably wouldn’t do use a programmatic system

Sorting

- There is not just a single correct algorithm to sort a list

- Let’s talk about a few of them
- We’ll do them all on OneNote

- Categorize them based on properties
- Photosensitivity warning

Bubble Sort

- Easiest to program
- Steps:
  - While list not sorted
    - Swap adjacent elements if out of order
- That the entire algorithm
- O(n^2)

Selection Sort

- Partition array into two parts
  - Sorted part
  - Unsorted part
- *For item in arr:*
  - *Select* the smallest piece from the unsorted part
  - Move it to the sorted part
- O(n^2)

Insertion Sort

- sorted_arr = []
- For item in unsorted_arr
  - *Insert* item into sorted_arr at correct position

- Also pretty easy to program
- O(n^2)

Sorting

- Algorithms we've seen so far
  - Same time complexity
  - O(n^2)
- These are all famously bad
- Collections.sort() does not do any of that
- Two algorithms that are better

Quick Sort

- Let a “pivot” be an element that we are trying to find the *final* position of

- Repeatedly pick pivots
- Find its canon, sorted position
- Repeat on left and right of pivot

Quick Sort

- Pick a pivot
- Ignore it for a second, put it at the end
- Find the following items in the *rest* of the list
  - itemFromLeft > the pivot
  - itemFromRight < the pivot
- Swap them

Quick Sort

- Repeat this process until:
  - Index of itemFromLeft > index of itemFromRight
- Then, swap itemFromLeft with the pivot

- With that, we can guarantee that our pivot is at the correct index
- Repeat on smaller halves of the list

Merge Sort

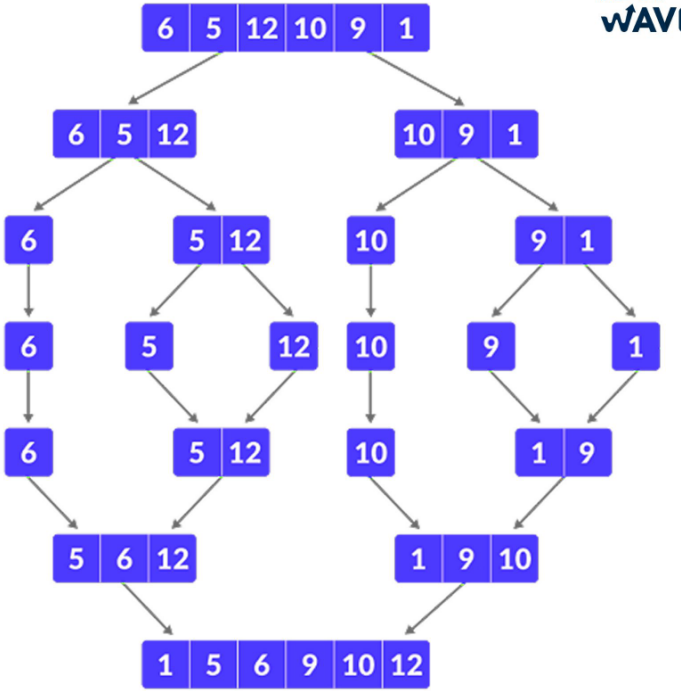
- Recursive algorithm

- Break the list into two halves
- Sort each half
- Combine them
- Return combined list

Merge Sort

- Easier to visualize
- Partition into sublists recursively

- Sometimes lists will only be size of 1



Merge Sort

- Base case is usually list of size 1 or 2
- List of size 1 is always sorted
- Size 2 would be only swap

- Combine step is the secret
  - This is much faster than it seems
- Let’s do it

Sleep Sort

- Sleep a thread for time interval based on value
- Once joined, put into a list

- That’s it
- This is silly and should not be used
- Interesting use + review of threads

Sorting

- Quick sort and merge sort are much faster
- O(n * log n)

- Collections.sort() → modified quick sort
- Actually both usually recursive
  - Merge sort often more commonly so

Sorting

- That’s it
- Exam is on the July 31st

- What now?
  - Final will be comprehensive
  - Try to lean heavier on newer topics
  - We can review whatever