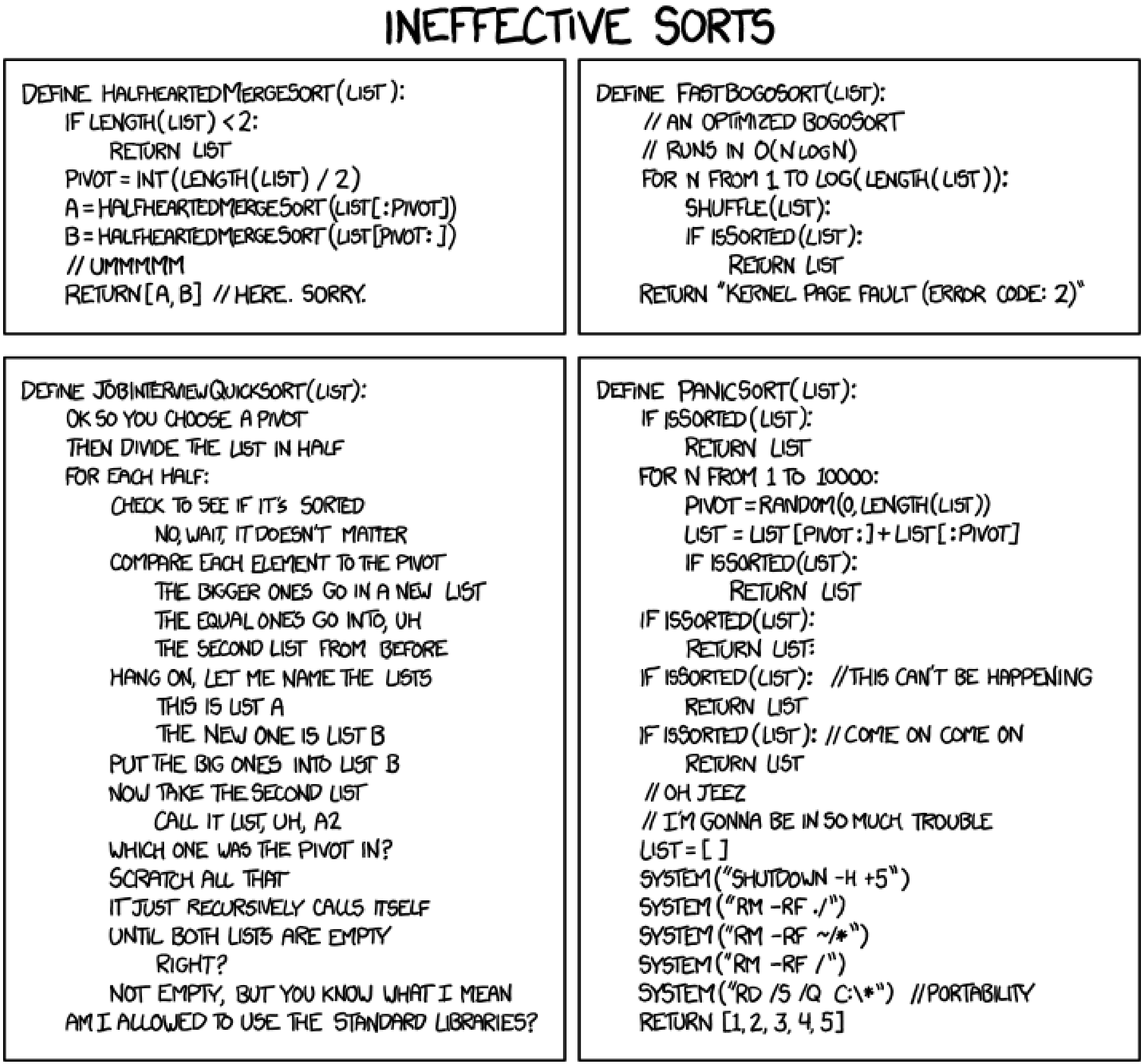
CS240
Fall 2014

Mike Lam, Professor

Misc. Sorting



http://xkcd.com/1185/

Retrospective

- Which sorting algorithm is best?

Retrospective

- Which sorting algorithm is best?

(it's a trick question!)

Simple Sorts

- Selection sort
  - Predictable: $n(n+1)/2$ comparisons and n copies
  - Few memory writes
  - In-place but **not** stable
- Insertion sort
  - Fast on nearly-sorted lists: $\sim n$ operations
  - In-place and stable
- Bubble sort
  - In-place and stable

Binary Insertion Sort

- Minor variant of insertion sort
- Binary search for insert location
  - Instead of linear scan
- Fewer comparisons: $O(n \log n)$
- Average time is still $O(n^2)$
  - Still requires $O(n)$ swaps per insertion

Skip List Sort

- Add every item to a skip list: $O(n \log n)$
- Then iterate over the list: $O(n)$
- Stable
- Requires lots of extra space: $O(n \log n)$
  - Similar to space requirements for merge sort
  - But cannot be optimized

Heap Sort

- **Heaps** are data structures that allow for $log(n)$ access to the minimum item in a list
- Heapsort algorithm

```
for elem in items:
    heap.insert(elem)
for i in range(len(items)):
    items[i] = heap.extract_min()
```

- $O(n \log n)$ worst case
- In-place, but not stable
- We'll examine this more later in the semester

Divide-and-Conquer Sorts

- Merge sort
  - $O(n \log n)$ worst-case time
  - Not in-place
  - Stable
  - Variants (e.g., Timsort) are widely used
- Quick sort
  - $O(n \log n)$ average/expected time
  - In-place
  - Not stable
  - Variants (e.g., introsort) are widely used

Minimum Worst Case

- Question: "Can we sort faster than $O(n \log n)$?"

Minimum Worst Case

- Question: "Can we sort faster than $O(n \log n)$?"
  - Not if we're using comparisons!
- Lower bound on worst-case comparison-based sorting: $\Omega(n \log n)$
- Justification involves a binary decision tree
  - Each node represents the result of a comparison
  - Each leaf node (or path through the tree) represents a possible permutation of the original list
  - Height of the list is at least $\log(n!) \geq (n/2)\log(n/2)$

Minimum Worst Case

- Question: "Can we sort faster than $O(n \log n)$?"
  - Possibly, if we're not using comparisons
- How do you sort without comparing items directly?
  - Multiple cycles of splitting items into bins
    - Preserve ordering within bins
  - Need to make restrictions on item domains
  - Examples:
    - Integer numbers < 10,000
    - Five-letter character strings

Non-Comparative Sorting

- Bucket sort
  - Create N buckets
  - Separate all elements into buckets: $O(n)$
  - Concatenate all buckets: $O(N)$
  - Requires some knowledge about the domain to be efficient
    - Goal: items are evenly distributed across all buckets
  - Stable when implemented carefully
  - Running time: $O(n + N)$

Non-Comparative Sorting

- Radix sort
  - Represent elements as ordered tuples
    - With O(1) access to elements by index
  - Perform repeated bucket sorts
    - One sort per index
    - Start with least-significant index
  - Running time is $O(d(n+N))$
    - *d* is the dimensionality of the input domain

Sorting Visualizations

- http://www.sorting-algorithms.com/
- http://panthema.net/2013/sound-of-sorting/
  - https://www.youtube.com/watch?v=kPRA0W1kECg
  - https://www.youtube.com/watch?v=ZZuD6iUe3Pc

Sort Algorithm Comparison

	Worst Case Comparisons	Worst Case Assignments	Worst Case Time	Best Case Time	Average Time	In Place?	Stable?
Selection Sort							
Insertion Sort							
Binary Insertion Sort							
Merge Sort							
Quicksort							
Bucket Sort (n elements, largest element is N)							
Radix Sort (n d-tuples largest element is N)							

Next Class: Review Session

- Midterm 2 is on Friday
  - Scope: all topics covered since Midterm 1: stacks, queues, linked lists, skip lists, recursion, recurrences, tail recursion elimination, basic sorting, divide-and-conquer sorting
  - Most important sorting algorithms: selection, insertion, merge, and quick sorts
- Review session on Wednesday
  - Canvas survey to collect topics
  - Email me if you have problems you'd like me to solve in-class (no guarantees!)