Computer Science 136
Data Structures
Lecture #12 (October 8, 2021)

1. Announcements:
   (a) Lab: please make the fix suggested in email.
   (b) Sample exam posted on the website. Answers discussed on Tuesday.
   (c) Questions?
2. Motivation: Binary Search
   (a) Suppose that data in an array are already in order. We can *divide-and-conquer* to find the location where a particular item might be found.
   (b) Keep bounds on possible locations: `low` and `high`.
   (c) Use recursion and logic to divide the range in half.
   (d) Search takes $O(\log_2 n)$ to find (or not).
3. Getting it done: Sorting.
   (a) Bubble sort.
      i. A pass: swap all values that are out of order, from left to right.
      ii. Repeat, as long as something swapped in previous pass.
      iii. Each pass moves the largest value to the right. Now we can think of the problem as having been reduced in size.
      iv. Recursion? You bet.
      v. Best case: $O(n)$ Worst case: $O(n^2)$ Runs quickly on sorted data, slowly on unsorted data, and less slowly on disordered data.
   (b) Selection sort. Halloween sorting technique.
      i. Same as bubble sort, but we realize that we only need to swap the maximum into place. A pass: find position of maximum value, swap it to the end.
      ii. Problem size is now reduced. However, you *must* repeat n-1 times.
      iii. All cases: $O(n^2)$ compares, $O(n) swaps$.
      iv. There's a recursive solution.
   (c) Insertion sort. Poker hand sorting technique.
      i. Think of low-indexed value as being sorted.
      ii. Pass: expand by adding one untested value to i sorted values. This takes $O(n)$ compares and $O(n)$ *movements* in all but the best case (sorted).
      iii. Repeat passes until sorted portion of array includes all values.
      iv. Recursive solution is pretty.
      v. Best case: $O(n)$ on sorted data, $O(n^2)$ on all others.
   (d) Quicksort.
      i. Small arrays of values are sorted.
      ii. Partition the data using a *pivot*. Attack the problem recursively, on left and right.
      iii. Performance on best case:
      iv. Performance on worst case:
      v. Improving worst case performance: Random pivots. Shuffling.

Notes: