Introduction to Sorting

What have we learned so far?

- The “cams” operator is:
  - compare-and-maybe-swap.
- We’ve shown that in order to comparison-sort an array of N elements, we require *at least:*
  - *N log N* cams.
- If we use reduction to simplify our problem, there are two choices to be made:
  - Equal (or quasi-equal) partitions vs. head/tail partition;
  - Work before recursion vs. work after recursion.

Now, things are going to get a bit more complicated…

- There's another way to divide the problem up—can you think how?
- There's only two helpful ways to divide a collection into 1 and N-1 elements (they are essentially the same)
- But what about dividing into *k* sub-arrays?
  - e.g. merge-sort: we put elements *iN/k…(i+1)N/k-1* into the i^{th} sub-array.
  - Any other ideas?
- How about dividing such that the i^{th}, $k+i^{th}$, $2k+i^{th}$, etc. form the i^{th} sub-array?
  - Could this make any difference?

The *five* possibilities

	Work then solve	Solve then work	Growth (~)
Equi-partition	Quick sort	Merge sort	N log N
Slice	?	Shell sort	~ $N^{1.5}$
Head-tail partition	Selection Sort	Insertion Sort*	$N^2/2$

* The number of cams for Insertion Sort is *min(N+X, N^2/2)* where *X* is # of inversions

$N \log N$ vs. $1/2\ N^2$

- Is there ever a possibility that $1/2\ N^2$ is less than $N \log N$?
  - Keep in mind that $N \log N$ is the <u>minimum</u> possible number of comparisons.
  - If we are using logs to base 2, then if $N = 4$, $\log_2 N = 2$ and $N \log_2 N$ is 8. $1/2\ N^2 = 8$.
  - So, for $2 <= N <= 4$, $1/2\ N^2 <= N \log_2 N$.
  - Yes—between $N = 2$ and $N = 4$

When to terminate the recursion

- So, we terminate the recursion (for the equi-partitioned case) when $N = 4$* and we cut over to insertion sort.
- We use insertion sort because, in the event that the sub-array is already partially sorted, it takes linear time, i.e. $N + X$ where X is the number of "inversions".

* the actual number will depend on benchmarking

Inversions

- An inversion is when two elements of a collection are out of order.

- From a collection of *N* elements, we can choose N(N-1) pairs—but half of these are just mirror images. So, the total number of pairs is $N(N-1)/2$.

- Such a pair of elements is inverted randomly true or false, so that means on average such a collection has $N(N-1)/4$ inversions.

Swap

- There are 2 forms of swap on an array *a* of length *N*:
  - swap(a, i)—
    - this form exchanges elements *i* and *i-1*.
    - It is a “stable” swap but…
    - it can only “fix” one inversion.
  - swap(a, i, j)—
    - this form allows a swap of two non-adjacent elements;
    - It is an “unstable” swap;
    - It can “fix” anything up to *2N-3* inversions (in the case of swapping the first/last of a reverse-ordered array) but the average number is of inversions per swap *N/4*.

More swapping

- Actually, there's a third form of swapping in an array *a* of length *N*:
  - *swapAndShift(a, i, j):*
    - this form allows a swap of all of the adjacent elements between *i* and *j*:
      - The value of *a[j]* ends up at index *i; a[i]* moves to *i+1; a[i+1]* moves to *i+2; etc…. a[j -1]* replaces *a[j]*.
    - It is a "stable" swap (provided that *i* is chosen appropriately);
    - It "fixes" *j - i* inversions. It is used in Insertion sort.

Number of inversions fixed per swap

- For merge sort, swaps are always with nearest neighbor so they fix exactly one inversion.

- For selection sort, up to *N-1* swaps are performed*, and since the number of inversions on average for a random list is *N(N-1)/4*, it follows that the average number of inversions per swap is *N/4*.

* actually, it can be as few as 0 swaps if the elements are already in order.

There are some more things to think about…

- Memory usage (extra to elements):
  - None: O(1) ?
  - “In-place”: O(log N)?
  - Linear: O(N)?
- Stability:
  - neighbor swaps or long-distance swaps?
- Comparison vs. Swap
  - Except for primitives, *swap* is always faster than *compare*
- Worst-case scenarios.