CS125 : Introduction to Computer Science

Lecture Notes #34
Insertion Sort Analysis

Lecture 34 : Insertion Sort Analysis

Analyzing `InsertionSort`

The worst case here is much like `SelectionSort`. The primary work here is InsertInOrder – the rest is just starting and returning from InsertionSort recursive calls, and as we saw above for SelectionSort, that is constant time per step and thus linear overall. ¡p¿ InsertInOrder's worst case is when A[hi] needs to be shifted all the way to the front of the array – then *every* value needs to be moved to a different cell, i.e. n values need to be moved. But of course, the size of the subarray that InsertInOrder works on is different each step, so we have a table much like for FindMinimum. Remember that the first call InsertionSort has, is the *last* time InsertInORder runs, since we don't run InsertInOrder in the first InsertionSort call, until we make the second InsertionSort call and return from it:

InsertionSort call	# of cells InsertInOrder moves in worst case in this call
1	n
2	n-1
3	n-2
4	n-3
5	n-4
6	n-5
.	.
.	.
.	.
k	n-(k-1) == n-k+1
.	.
.	.
.	.
n-4	5
n-3	4
n-2	3
n-1	2
n (i.e. base case)	0

So again, worst case is about (n squared)/2. On average, we might expect to have to shift A[hi] down half the array, rather than the entire array, so the average case would involve multiplying each of the second-column values above by one-half. So, the average case would be about (n squared)/4.

Note that for SelectionSort, the worst and average cases were the same. For InsertionSort, though they are both quadratic, if you were to time them, the average case would take only about half as long as the worst case.

This trend continues – if you had an already sorted array, then in InsertInOrder, A[hi] would need to be compared to A[hi-1], but no shifting would be needed and the method would end there. That's just constant time...so over the life of the algorithm – n steps – we spend constant time on each step so it's linear total. Or in other words, on an already-sorted array, all InsertionSort basically does is to compare A[2] to A[1], then compares A[3] to A[2], then compares A[4] to A[3], and so on, up to comparing A[hi] to A[hi-1]. That's n-1 comparisons (there's no comparison in the base case) so linear time.