CIS 575. Introduction to Algorithm Analysis
Material for January 19, 2024

Insertion Sort: Time and Space Analysis

This topic is covered in *Cormen*'s Section 2.2.

1 Analysis of Insertion Sort

Recall that we came up with an iterative implementation of *insertion sort*:

```
for i ← 2 to n
    j ← i
    while j > 1 and A[j] < A[j − 1]
        A[j] ↔ A[j − 1]
        j ← j − 1
```

Space Use The array A will occupy memory space proportional to n, but that space was allocated already before the invocation of InsertionSort which should therefore *not* be charged for that space. In general, the space use of an algorithm is defined as the *extra* memory it allocates.

InsertionSort needs to store the integer variables i and j, and thus its extra space use appears constant, though for very large n more than one computer word must be allocated for each variable as it holds an integer whose bit representation has a size that is logarithmic in n. An algorithm whose space use is "substantially less" (constant or at most logarithmic) than the size of the input, is said to be *in-place*.

Time Use If A is already "almost" sorted, the inner loop will iterate at most a few times, and hence the overall running time will be proportional to n.

But if A is (almost) sorted in *reverse* order, the inner loop could iterate close to j times. In that case, the overall running time is proportional to n^2.

It is not hard to see that if A is populated by a random generator, the expected number of iterations of the inner loop will be close to $\frac{j}{2}$, and also then the expected running time will be proportional to n^2.