DS 320 Algorithms for Data Science — **Lecture #2 Worksheet**
Fall 2024 — Profs. Krzysztof Onak and Aldo Pacchiano

Insertion Sort and Induction

We will analyze the runtime of the following algorithm.

Algorithm 1 InsertionSort(A).

```
Input: $A$ is an array of integers. It is indexed 1 to $n$.
for $i = 2$ to $A.length$ do
    $current = A[i]$
    $j = i - 1$
    while $j > 0$ and $A[j] > current$ do
        $A[j+1] = A[j]$
        $j = j - 1$
    end while
    $A[j+1] = current$
end for
return $A$
```

First, we analyze the algorithm's runtime. This time, we'll *formally* argue its runtime.

Theorem 1. *Insertion Sort runs in time $O(n^2)$.*

Proof.

Now, we argue the algorithm's *correctness*—that is, that on *every possible input*, it correctly outputs a sorted version.

Correctness via Induction

Theorem 2. *For any input instance A, Insertion Sort returns an array sorted in ascending order.*

We want to use induction on a claim that, when true for all integers, proves our claim. What claim might that be?

Proof. We show the following by induction on i: (*premise*)

Base Case:

Inductive Hypothesis:

Inductive Step:

We need another component to prove the inductive step. Separately, we want to prove the following:

Lemma 1. (Loop Invariant) *If the algorithm enters into the "while" loop it will shift* $A[j+1, i-1]$ *to* $A[j+2, i]$ *in the same order for some* j*.*

$\square$

Loop Invariants

We'll use a different technique now to prove Lemma 1.

Definition 1. A *loop invariant* is something that is true before we start and after every iteration of a loop.

We prove that a loop invariant is true by showing the following three things about it:

- **Initialization:** It is true prior to the first iteration of the loop.
- **Maintenance:** If it is true before an iteration of the loop, it remains true before the next iteration.
- **Termination:** When the loop terminates, the invariant gives us a useful property that helps show that the algorithm is correct.

See CLRS Section 2.1 for details, p.18 in the Third Edition. We use this to prove our Lemma.

Proof of Lemma 1. We will prove this formally as a loop invariant.

Initialization: Before the first iteration of the "while" loop,

Maintenance: If our statement holds before an iteration of the loop—that $A[j+1, i-1]$ is shifted to $A[j+2, i]$ in order—then

Termination: When the loop terminates,

□

Comparison-Based Lower Bound via A Counting Argument

Note: We will not cover this in class. So Insertion Sort provably always correctly sorts any input array in $O(n^2)$ time! But can we do better? Perhaps we can improve on the $O(n^2)$ running time to get an algorithm that runs in time $O(n)$? To answer this question we need to be more precise about what a "solution" can do. Selection sort inspects the input data using only a single operation: a comparison (i.e. its branching condition is of the form "If $A[i] \leq A[j]$ then…"). It is the result of these comparisons (and nothing else) that determines which swaps are performed, which comparisons are performed next, and ultimately which permutation π of the input array A is finally output. That is to say, Insertion Sort operates in the comparison based model of computation:

Definition 2. An algorithm operates in the *Comparison Model* if it can be written as a binary decision tree in which:

1. Each vertex is labelled with a fixed comparison (i.e. $A[i] < A[j]$ for particular i, j)
2. Computation proceeds as a root-leaf path down the tree, branching left if the comparison evaluates to TRUE and right otherwise, and
3. The leaves are labelled with the output of the algorithm (in this case, permutations)

In this model, the running time of the algorithm corresponds to the depth of the tree.

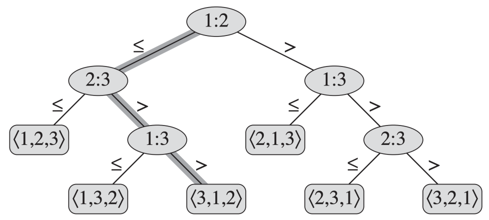

Figure 1: Example decision tree from CLRS.

As it turns out, we can prove an easy lower bound for sorting algorithms in the comparison model. Lower bounds of this sort serve as a guide: either we should not waste effort trying to derive algorithms that improve on the lower bound, or, we should find techniques that step outside of the model in which the lower bound is proven.

Theorem 3. *Any algorithm that solves the sorting problem in the comparison model must have run time at least* $\Omega(n \log n)$.

How many permutations of A of length n are there? What does that imply in terms of leaves?

What does the number of leaves imply about the depth of the tree?

Proof. The proof is via a nice counting argument. Consider any sorted array A of length n.

Based on material developed by Prof. Kira Goldner