Data Structures and Algorithms
CMPSC 465

Lecture 2
- Insertion Sort
- Correctness via loop invariants
- Measuring efficiency

Adam Smith

S. Raskhodnikova and A. Smith; based on slides by E. Demaine and C. Leiserson
Insertion Sort

```
INSERTION-SORT (A, n)    ▷ A[1 . . n]
for j ← 2 to n
    do key ← A[j]
       i ← j - 1
    while i > 0 and A[i] > key
    do  A[i+1] ← A[i]
        i ← i - 1
    A[i+1] ← key
```

“pseudocode”
Insertion Sort

```
INSERTION-SORT (A, n)  ▷ A[1 . . n]
  for j ← 2 to n
    do key ← A[j]
       i ← j – 1
      while i > 0 and A[i] > key
      do     A[i+1] ← A[i]
       i ← i – 1
       A[i+1] ← key
```

A: 1 i j n

“pseudocode”

sorted

key
Example of Insertion Sort

8 2 4 9 3 6
Example of Insertion Sort

8 2 4 9 3 6
Example of Insertion Sort

8 2 4 9 3 6

2 8 4 9 3 6
Example of Insertion Sort

8 2 4 9 3 6
2 8 4 9 3 6
Example of Insertion Sort

8 2 4 9 3 6

2 8 4 9 3 6

2 4 8 9 3 6
Example of Insertion Sort

8 2 4 9 3 6
2 8 4 9 3 6
2 4 8 9 3 6
Example of Insertion Sort

8 2 4 9 3 6

2 8 4 9 3 6

2 4 8 9 3 6

2 4 8 9 3 6
Example of Insertion Sort

8  2  4  9  3  6
2  8  4  9  3  6
2  4  8  9  3  6
2  4  8  9  3  6
Example of Insertion Sort

8 2 4 9 3 6
2 8 4 9 3 6
2 4 8 9 3 6
2 4 8 9 3 6
2 3 4 8 9 6

S. Raskhdonikova and A. Smith. Based on slides by E. Demaine and C. Leiserson
Example of Insertion Sort

8 2 4 9 3 6
2 8 4 9 3 6
2 4 8 9 3 6
2 4 8 9 3 6
2 3 4 8 9 6
Example of Insertion Sort

8 2 4 9 3 6
2 8 4 9 3 6
2 4 8 9 3 6
2 4 8 9 3 6
2 3 4 8 9 6
2 3 4 6 8 9
done
Correctness of Insertion Sort

Loop Invariant:
After execution of execution j of for loop

1. $A[1 .. n]$ is a permutation of the input array, and

2. $A[1 .. j]$ is sorted
Loop Invariants

A tool for analyzing iterative algorithms
(example of inductive reasoning)

Usually, we prove 3 statements

- **Initialization**: invariant holds on first execution
- **Maintenance**: if invariant held on all previous passes through the loop, it holds on current pass
- **Termination**: if invariant holds at the end, then some desired property holds (e.g. algorithm is correct).
Correctness of Insertion Sort

- **Initialization**: A[1] is sorted.

- **Maintenance**: If A[1..j-1] is sorted before pass j through the for loop, then A[1..j] is sorted after the pass. This holds because A[j] is inserted in the correct place in A[1..j-1].
  
  – Proving this formally requires looking carefully at while loop.

- **Termination**: If loop invariant holds at termination (j = n), Insertion Sort is correct. Loop invariant states that A[1..n] is sorted when the for loop exits. Since the array elements were never changed (only permuted), A now contains the sorted version of the input.
How to measure running time?

• Parameterize the running time by the size of the input, denoted by $n$, since short sequences are easier to sort than long ones.

• Issue: the running time depends on the input: an already sorted sequence is easier to sort.

• Generally, we seek upper bounds on the running time
Kinds of analyses

**Worst-case:** (usually)
- $T(n) =$ maximum time of algorithm on any input of size $n$.

**Average-case:** (sometimes)
- $T(n) =$ expected time of algorithm over all inputs of size $n$.
- Requires assumption about distribution of inputs.

**Best-case:** (bogus!)
- Cheat with a slow algorithm that works fast on *some* input.
Machine-independent time

What is Insertion Sort’s worst-case time?
• It depends on the speed of our computer:
  • relative speed (on the same machine),
  • absolute speed (on different machines).

**BIG IDEA:**
• Ignore machine-dependent constants.
• Look at *growth* of \( T(n) \) as \( n \to \infty \).

“Asymptotic Analysis”
Insertion sort analysis

**Worst case:** Input reverse sorted.

\[ T(n) = \sum_{j=2}^{n} c \cdot (j - 1) = cn(n - 1)/2 = \Theta(n^2) \]

[arithmetic series]

The “\( \Theta \)” notation ignores constants and “low-order” terms. Defined next lecture.

**Is insertion sort a fast sorting algorithm?**

- Moderately so, for small \( n \).
- Not at all, for large \( n \).