INFORMATION & COMPUTER SCIENCES
UNIVERSITY of HAWAIʻI at MĀNOA

Ch 3.3: Complexity of Algorithms

ICS 141: Discrete Mathematics for Computer Science I

KYLE BERNEY
DEPARTMENT OF ICS, UNIVERSITY OF HAWAII AT MANOA

Analyzing Algorithms

- Analyzing an algorithm means predicting the resources that the algorithm requires
  - Memory
  - Communication bandwidth
  - Power consumption
  - Computation
- A model for the resources of a particular technology is needed

Random-Access Machine (RAM) Model

- Instructions are executed sequentially
- Includes common instructions for modern computers:
  - Arithmetic operations (add, subtract, multiply, divide, remainder, floor, ceiling)
  - Data movement (load, store, copy)
  - Control statements (conditional and unconditional branch, subroutine call, return)
- Each instruction takes a constant amount of time
- Primitive data types
  - Integers
  - Floating point (i.e., real numbers)

Time Complexity

- Definition: The running time (or time complexity) of an algorithm on a particular input is the number of primitive operations or "steps" executed
- Analyze the runtime of various scenarios:
  - Best-case: smallest number of operations performed
  - Worst-case: largest number of operations performed
  - Average-case: average number of operations performed (typically found using probabilistic analysis)
  - Expected: expected number of operations performed (randomized algorithms)

Time Complexity

- Generally concerned with finding the worst-case runtime

1. Provides an upper bound on the runtime of the algorithm for arbitrary input
   - Guarantees that the algorithm will never take longer
2. Worst-case may occur fairly often
   - Ex: Searching for an element that is not present
3. Average-case is often roughly as bad as the worst-case

Insertion Sort

```
InsertionSort(A[1 ... n])
    for j = 2 to n
        key = A[j]
        // Insert A[j] into the sorted sequence A[1 ... j − 1]
        i = j − 1
        while i > 0 and A[i] > key
            A[i + 1] = A[i]
            i = i − 1
        A[i + 1] = key
```

- Recall: each instruction takes a constant amount of time
    - Define constants for each line
    - Count the number of times each line executes

Insertion Sort

```
1: INSERTIONSORT(A[1 . . . n])
2:     for j = 2 to n
3:         key = A[j]
4:         // Insert A[j] into the sorted sequence A[1 . . . j − 1]
5:         i = j − 1
6:         while i > 0 and A[i] > key
7:             A[i + 1] = A[i]
8:             i = i − 1
9:         A[i + 1] = key
```

- Line 2 executes n times

$$\Rightarrow c_1 n$$

Insertion Sort

```
1: InsertionSort(A[1 . . . n])
2:     for j = 2 to n
3:         key = A[j]
4:         // Insert A[j] into the sorted sequence A[1 . . . j − 1]
5:         i = j − 1
6:         while i > 0 and A[i] > key
7:             A[i + 1] = A[i]
8:             i = i − 1
9:         A[i + 1] = key
```

- Line 3 executes $(n - 1)$ times

$$\Rightarrow c_2(n-1)$$

Insertion Sort

```
1: INSERTIONSORT(A[1 . . . n])
2:     for j = 2 to n
3:         key = A[j]
4:         // Insert A[j] into the sorted sequence A[1 . . . j − 1]
5:         i = j − 1
6:         while i > 0 and A[i] > key
7:             A[i + 1] = A[i]
8:             i = i − 1
9:         A[i + 1] = key
```

- Line 5 executes $(n - 1)$ times

$$\Rightarrow c_3(n-1)$$

Insertion Sort

```
1: INSERTIONSORT(A[1 . . . n])
2:     for j = 2 to n
3:         key = A[j]
4:         // Insert A[j] into the sorted sequence A[1 . . . j − 1]
5:         i = j − 1
6:         while i > 0 and A[i] > key
7:             A[i + 1] = A[i]
8:             i = i − 1
9:         A[i + 1] = key
```

- Let t_j be the number of times that Line 6 executes for a given value of j

$$\Rightarrow c_4 \sum_{j=2}^{n} t_j$$

Insertion Sort

```
1: INSERTIONSORT(A[1 ... n])
2:     for j = 2 to n
3:         key = A[j]
4:         // Insert A[j] into the sorted sequence A[1 ... j − 1]
5:         i = j − 1
6:         while i > 0 and A[i] > key
7:             A[i + 1] = A[i]
8:             i = i − 1
9:         A[i + 1] = key
```

- Line 7 executes $(t_j - 1)$ times for a given value of j

$$\Rightarrow c_5 \sum_{j=2}^{n} (t_j - 1)$$

Insertion Sort

```
1: INSERTIONSORT(A[1 . . . n])
2:     for j = 2 to n
3:         key = A[j]
4:         // Insert A[j] into the sorted sequence A[1 . . . j − 1]
5:         i = j − 1
6:         while i > 0 and A[i] > key
7:             A[i + 1] = A[i]
8:             i = i − 1
9:         A[i + 1] = key
```

- Line 8 executes $(t_j - 1)$ times for a given value of j

$$\Rightarrow c_6 \sum_{j=2}^{n} (t_j - 1)$$

Insertion Sort

```
1: InsertionSort(A[1 . . . n])
2:     for j = 2 to n
3:         key = A[j]
4:         // Insert A[j] into the sorted sequence A[1 . . . j − 1]
5:         i = j − 1
6:         while i > 0 and A[i] > key
7:             A[i + 1] = A[i]
8:             i = i − 1
9:         A[i + 1] = key
```

- Line 9 executes $(n-1)$ times

$$\Rightarrow c_7(n-1)$$

Insertion Sort

- Let $T(n)$ be the runtime of insertion sort
- Sum up the runtime of each line:

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} t_j$$

$$+ c_5 \sum_{j=2}^{n} (t_j - 1) + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7(n-1) \; .$$

- The number of times Line 5 executes, t_j, depends on the input sequence

Insertion Sort

- Best-case:
  - When the array is already sorted, we always find that

    $$A[i] \not> key$$

    the first time the **while** loop is executed
  - Therefore, $t_j = 1$

$$
\begin{aligned}
T(n) &= c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} 1 + c_7(n-1) \\
&= c_1 n + c_2(n-1) + c_3(n-1) + c_4(n-1) + c_7(n-1) \\
&= (c_1 + c_2 + c_3 + c_4 + c_7)n - (c_2 + c_3 + c_4 + c_7) \\
&= \Theta(n)\ .
\end{aligned}
$$

Insertion Sort

- Worst-case:
  - When the array is in reverse sorted order, we always find that
    $$A[i] > key$$
    until $i \not> 0$ and the **while** loop terminates
  - Therefore, $t_j = j$

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} j$$

$$+ c_5 \sum_{j=2}^{n} (j-1) + c_6 \sum_{j=2}^{n} (j-1) + c_7(n-1)$$

Insertion Sort

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} j$$

$$+ c_5 \sum_{j=2}^{n} (j-1) + c_6 \sum_{j=2}^{n} (j-1) + c_7(n-1)$$

$$= c_1 n + c_2(n-1) + c_3(n-1) + c_4 \left(\frac{n(n+1)}{2} - 1 \right)$$

$$+ c_5 \left(\frac{n(n-1)}{2} \right) + c_6 \left(\frac{n(n-1)}{2} \right) + c_7(n-1)$$

$$= \left(\frac{c_4}{2} + \frac{c_5}{2} + \frac{c_6}{2} \right) n^2 + \left(c_1 + c_2 + c_3 + \frac{c_4}{2} - \frac{c_5}{2} - \frac{c_6}{2} + c_7\right) n$$

$$- (c_2 + c_3 + c_4 + c_7) = \Theta(n^2) \; .$$