Agenda

• Last week
  - Test
• This week
  - Section 7.1 and possibly part of 7.2
• Remainder of semester
  - Chapter 7
Announcement

• Revised homework assignment for next Tuesday
  - 7.1, 7.2, 7.6, 7.9, 7.12
• Test will be returned tomorrow
Complexity of algorithms

- Among decidable problems, we still have levels of difficulty for algorithms.
- We may consider an algorithm more difficult for a variety of reasons:
  - Takes longer to execute
  - Requires more memory to execute

Time complexity: Given an algorithm and an input string, how long will the algorithm take to execute?
Example

INSERTION-SORT(A)
1. For j = 2 to length(A)
2. key := A[j]
3. i := j-1
4. While i > 0 and A[i] > key
6. i := i-1
7. A[i+1] := key
Trace

- Start by looking at 6 & compare to 4
  - $6 \geq 4$
  - Next look at 1

- 8 is okay
  - Move 5 then move 3

- 1 3 4 5 6 8
How long does insertion sort take?

- **Two loops**
- **Outer loop executed (n-1) times**
  - \( n = \text{length}(A) \)
- **Inner loop executed up to j times**
- **Total time is at most** \( \sum_{1 \leq k < n} (3 + \sum_{1 \leq i < k} 2) \)
  - \( 3(n-1) + 2 \frac{(n-1)n}{2} = (n-1)(3+n) = n^2 + 2n - 3 \)
Time complexity

Definition: Let $M$ be any deterministic Turing machine that halts on all inputs. The running time or time complexity of $M$ is the function $f: \mathbb{N} \rightarrow \mathbb{N}$, where $f(n)$ is the maximum number of steps that $M$ uses on any input of length $n$. 
Big-O notation

• In general, the time complexity will be the sum of terms that is dominated by one term
  - E.g., $n^2 + 2n - 3$ is dominated by the $n^2$ term

• Time complexity is most concerned with behavior for large $n$
  - We disregard all terms except for the dominating term
  - $n^2 + 2n - 3 = O(n^2)$
Asymptotic upper bound

**Definition:** Let $f$ and $g$ be two functions from $\mathbb{N}$ to $\mathbb{R}^+$ (the set of positive real numbers). Then $f(n) = O(g(n))$ if positive integers $c$ and $n_0$ exist such that for every $n \geq n_0$, $f(n) \leq c \times g(n)$. In this case, we say that $g(n)$ is an upper bound for $f(n)$.
Example

• $3n^4 + 5n^2 - 4 = O(n^4)$
  $3n^4 + 5n^2 - 4 \leq 4n^4$ for every $n \geq 2$ since

  $3n^4 + 5n^2 - 4 \leq 4n^4 \Rightarrow$
  $n^4 - 5n^2 + 4 \geq 0 \Rightarrow$
  $(n^2-4)(n^2-1) \geq 0$ clearly holds for all $n \geq 2$

• For polynomials, we can drop everything except $n^k$, where $k$ is the largest exponent
Big-O notation and logarithms

• Recall \( \log_b n = \frac{\log_x n}{\log_x b} \)
  - \( \log_b n = O(\log_x n) \) for every \( x > 0 \)
  - With big-o notation, the base of the logarithm is unimportant!

• \( 5n^5 \log_3 n - 3n^2 \log_2 \log_2 n = O(n^5 \log n) \)
Mathematics with big-O notation

• If \( f(n) = O(n^3) + O(n) \), then \( f(n) + O(n^3) \)
  - Can simply select the largest term

• What does \( f(n) = 3^{O(n)} \) mean?
  - \( 3^{O(n)} \geq 3^{cn} \) for some constant \( c \)

• How about \( O(1) \)?
  - \( O(1) \geq c \) for some constant \( c \)
  - Constant time
Exponential time

• What about \( f(n) = 2^{O(\log n)}? \)
  \[- n = 2^{\log_2 n} \Rightarrow n^c = 2^{O(\log n)} \Rightarrow 2^{O(\log n)} = n^{O(1)} \]

• An algorithm takes exponential time if its complexity is \( O(a^{n^k}) \), where \( k > 0 \)

• An algorithm takes polynomial time if its complexity is \( O(n^k) \) for some \( k > 0 \)
Small-o notation

Definition: Let $f$ and $g$ be two functions from $\mathbb{N}$ to $\mathbb{R}^+$ (the set of positive real numbers). Then $f(n) = o(g(n))$

$$\lim_{n \to \infty} \left(\frac{f(n)}{g(n)}\right) = 0$$

i.e., for any positive real number $c$, a number $n_0$ exists such that for every $n \geq n_0$, $f(n) < c \times g(n)$
Some identities

- \( n^i = o(n^k) \) for every \( i < k \)
- \( \log n = o(n) \)
- \( \log \log n = o(\log n) \)