CS330 Activities for Lecture 7 [small change p.2]

3.1 Algorithms - Key Concepts
- **Definition.** An algorithm is a step-by-step process for solving a problem.
- What are the key characteristics of a “good” algorithm? Why are they hard to develop?
  - Algorithms are well-ordered, have unambiguous operations, have effectively computable operations (right level of detail), produce a result (hopefully correct), halt in a finite amount of time, and they must “scale”

3.2 Growth of Functions - Key Concepts

**Computational Complexity**
- Computational complexity is the amount of a particular resource used by an algorithm as the problem size grows.
  - Resources include time, memory, electrical power, ….
- The **worst-case time complexity** of an algorithm is a function that gives the maximum number of atomic operations performed by the algorithm across all inputs of size $n$.
  - **Input size** measures the size of the input in units that correspond to the operations we’re doing on it. E.g., for an array lookup, the number of items in the array; probably not the number of bits that comprise the array value. For computational algorithms, probably numbers of some fixed maximum size (double precision, e.g.).
  - **Atomic operations** are basic units of work. Can vary but typically include comparisons, additions, multiplications, assignments to variables, etc. Preferable to wall-clock time, which can vary too much.

**Comparing Running Times**
- Typically investigate **worst-case running time** (gives most pessimistic estimate of running time).
- Can use rough comparisons of running times to weed out inherently-slower algorithms that solve the same problem.
  - E.g., selection sort vs “for all $n!$ permutations of the input, if the permutation is sorted, then stop.”
- Also, exact comparisons are much harder to do in most cases.

**Asymptotic Measures**
- Asymptotic measures are a mathematical framework for rough comparisons of functions for large enough inputs.
  - Textbook discusses Big-Oh (rough $\leq$), Big-Omega (rough $\geq$), and Big-Theta (rough $=$).
- Rough, not tight, measures because:
  - The relationship holds to **within a multiplicative constant**.
  - The relationship can have a finite number of counterexamples: For all $n$ past some initial point, the comparison always holds. (A.k.a. "eventually always true" or "has finite number of counterexamples.")

**Big-Oh and Big-Omega**
- $f \in O(g)$ means there’s a positive constant $c$ such that for those large $n$, we have $f(n) \leq c \cdot g(n)$
  - Time complexity functions are positive for $n > n_0$, so we can also say $f(n) / g(n) \leq c$.
- Technical definitions (note for $O(…)$ we check for $\leq$; for $\Omega$, we check for $\geq$).
  - $f \in O(g) \equiv (\exists c \in \mathbb{R}) (c > 0 \land (\exists n_0 \in \mathbb{N}) \forall n \in \mathbb{N} (n > n_0 \rightarrow f(n) \leq c \cdot g(n)))$
  - $f \in \Omega(g) \equiv (\exists c \in \mathbb{R}) (c > 0 \land (\exists n_0 \in \mathbb{N}) \forall n \in \mathbb{N} (n > n_0 \rightarrow f(n) \geq c \cdot g(n)))$
- Some common functions (in strictly increasing asymptotic order):
  - $\log \log n, \log n, n, n \log n, n^2, n^3, \ldots$ all polynomials…, $2^n, 3^n, \ldots, n!, \ldots$
  - For polynomials, only the degrees are salient: $f(n) \in O(g(n))$ if the degree of $f \leq$ degree of $g$.
    - Similarly, $f(n) \in \Omega(g(n))$ if the degrees are $\geq$, and $f(n) \in \Theta(g(n))$ if the degrees are equal.
  - For logarithms, the base doesn’t matter ($\log_a x = \log_b x$ within a multiplicative constant).

---

*Some material from MATH230 Dr. Robert Ellis IIT-AMAT, zyBooks Disc. Math., and Disc. Math. and its Appl., Rosen (7th Ed.) is included under the academic/nonprofit Fair Use exception of US Copyright Law. This document must NOT be distributed beyond participants in IIT’s CS330.*
1. Show that $30n + 8$ is $O(n)$:
   For all $n > \_\_\_\_, \text{ we have } (30n + 8)/n = 30 + 8/n \leq \_\_\_, \text{ so } 30n + 8 \leq \_\_\_\_*n$.

2. Show that $n^2 + 1$ is $O(n^2)$

3. Show that $10n^3 - 5n^2 \in O(n^3)$ \[\text{[Hint: What is the smallest } n \text{ that makes } 10n^3 \geq 5n^2 \text{?]}\]
   \[\text{[Added after class: Also try } n^3 - 5n^3. \text{ You’ll find that } n_0 = 0 \text{ doesn’t work.]}\]

3.3 Analysis of Algorithms - Key Concepts

- **Analysis of Algorithms** refers to analyzing the complexity of algorithms by counting or estimating the number of operations the algorithm executes for an input of a given size.
- A basic operation takes $\Theta(1)$ time: It’s $\leq$ some constant. (Fixed constant: Can’t vary the constant as the program runs.)
- A finite number of $\Theta(1)$ sums to $\Theta(1)$. E.g., for $\Theta(1) + \Theta(1)$ we add two constants and get a larger one.
- A varying number $n$ of $\Theta(1)$ sums to $\Theta(n)$.

**Complexity analysis**
- Consider function that returns $\text{max}$ of an array of $n$ integers:
  ```c
  int max(array of integers $a_1, \ldots, a_n$) {
    (assign) $v := a_1$ \hspace{1cm} \text{line 1}
    \text{for } i := 2 \text{ to } n \hspace{1cm} \text{line 2}
    \quad \text{if } a_i > v \text{ then } v := a_i \hspace{1cm} \text{line 3}
    \text{return } v \hspace{1cm} \text{line 4}
  }
  ```
- Total running time $t(n)$ for $\text{max}$ depends on the length $n$ of the array and on the length of time for one execution of each line. Assume each line takes 1 unit of time (e.g., $\leq 5$ ns). Total running time $t(n) = 1 + (1 + 1)n + 1 = 2n + 2$ units of time. Since $2n + 2$ roughly equals $n$, we have $t(n) \in \Theta(n)$.

**Describing (Worst-Case) Behavior**
- "$f(n) = \text{Worst-case running time}" means "$f(n) = \text{maximum number of operations across all inputs of size } n.$"
- Recall $f(n) \in O(g(n))$ means that $g(n)$ is a rough upper bound for $f(n)$. 
• Note \( f(n) \in O(\text{any function larger than } g(n)) \), so \( n^2 \) is in \( O(...) \) of \( n^2, n^3, \ldots, 2^n \), etc. I.e., if \( f(n) \in O(g(n)) \) and \( g(n) \in O(h(n)) \), then \( f(n) \in O(h(n)) \). But \( n^2 \) is a **tighter upper bound** than \( n^3 \), since \( n^2 \in O(n^3) \) but not vice versa.

• Similarly \( f(n) \in \Omega(g(n)) \) means that \( g(n) \) is a rough lower bound for \( f(n) \).

  • **Important**: It’s possible for the algorithm to take less than \( g(n) \) time for some input because \( f(n) \) describes worst-case behavior: It’s the largest amount of time taken across all inputs of size \( n \).

• **Example**: Take linear search of an array of length \( n \).
  - Worst case: Search takes \( \Theta(n) \) time because it tests all \( n \) array values.
  - So \( f(n) \in \Omega(n) \) as a tight lower bound; \( f(n) \in \Omega(1) \) as a very loose lower bound.
  - Best case: Search takes \( \Theta(1) \) time because it finds the value in the first slot of the array.
  - So running time of best case on input size \( n \in \Omega(1) \), but best case \( \not\in \Omega(n) \).

1. Recall selection sort of array \( A \) of length \( n \):
   
   ```
   for i = 0 to n-1 {
       find smallest value in \( A[i], A[i+1], \ldots, A[n-1] \) (takes \( \Theta(n-i) \) time)
       swap \( A[i] \) and that smallest value
   }
   ```

   What is the asymptotic worst-case running time for selection sort? What is the asymptotic best-case running time?

2. Recall insertion sort of array \( A \) of length \( n \):
   
   ```
   for i = 1 to n-2 {
       Move \( A[i] \) leftward through \( A[i-1], A[i-2], \ldots, A[0] \) until its left neighbor (if any) \( \leq \) it \( \leq \) its right neighbor.
       (Requires between 1 and \( i-1 \) steps, \( \Theta(1) \) time each step)
   }
   ```

   What is the asymptotic worst-case running time for selection sort? What is the asymptotic best-case running time?