Reminders

- Exams will be returned next Tuesday
- Solutions for Exam will be posted
- Homework will be released today
  - Notification will be sent via email
- Quiz next Thursday on Algorithms and Complexity

CMSC 203: Lecture 12

Algorithms and Complexity
(Continued)

Reminder of Big-O

- **Big-O notation** is used for asymptotic analysis
  - Interested as input goes to infinite
  - Excludes coefficients and low order terms
- Provides an “upper-bound” for the algorithm
- Given two functions f(x) and g(x), we say that f(x) is O(g(x)) if: $\exists c, k$ such that $|f(x)| \leq c|g(x)|$ for all $x > k$
  - g(x) is an upper bound for f(x) for all $x > k$

Practice

Show that $7x^2$ is $O(x^3)$

Practice

Show that $7x^2$ is $O(x^3)$

We need some *k* and *c* such that $7x^2 < x^3$ when $x > k$

c = 1, k = 7

$7x^2 < (1)x^3$ when $x > 7$

Practice

Show that x^2 is not $O(x)$

Practice

Show that x^2 is not $O(x)$

Assume some *c* and *k* exist

$n^2 \leq cn$ for all $n > k$

Divide by *n* for: $n \leq c$

No *c* exists that can be greater than all integers

n will eventually grow larger than *c*

Therefore, this contradicts our assumption a *c* exists

x^2 is not $O(x)$

Growth Functions

- We want to compare the runtime of algorithms
- We can compute the “growth function” with Big-O
- There are several common growth functions we will see

Constant Growth

- **Constant Time** - O(1)
  - Size of input has no bearing on time complexity
  - Examples:
    - Accessing single element in an array
    - Checking if a number is even or oddt

Logarithmic Growth

- **Logarithmic Time** – O(log n)
  - Frequently seen in binary search and binary trees
  - Operations per instance decrease to complete decreases per instance
  - Examples:
    - Binary search (cuts size of list in half each search)
    - Finding a word in dictionary (with binary search)
  - Algorithms are considered efficient with this

Linear Growth

- **Linear Time** – O(n)
  - Run time increases linearly with size of input
  - Frequently seen when algorithm accesses every item in list once
  - Examples:
    - Search / min / max in unsorted list
    - Reading a book (each page)
  - Still pretty good run-time

Linearithmic Growth

- **Linearithmic Time** – O(n log n)
  - Perform a logarithmic operation for every element in the input
  - Frequently seen with sorting and binary trees
  - Examples:
    - Sorting a deck of cards (with mergesort)
  - Not as great as linear, but better than polynomial

Polynomial Growth

- **Polynomial Time** – $O(n^k)$ for some constant k
  - Bounded by any polynomial expression
  - Examples:
    - Bubble sort
    - Most nested “for” loops
    - Checking if every item on a shopping list is in the cart
  - Considered **Tractable**
  - Complexity class **P** (more on this later)

Exponential Growth

- **Exponential Time** – $O(c^n)$ for some constant c
  - **Intractable** – very poor run time
    - ie: Very slow with even moderately sized input
  - Examples:
    - Bruteforcing a password
    - Satisfiability of a logical proposition

Factorial Growth

- **Factorial Time** – O(n!)
  - Extremely long time, very intractable
  - Example:
    - Finding all orders for a traveling salesman to visit n cities

Comparison of Growth Functions

- In order of smallest to largest
  - **Constant** $O(1)$
  - **Logarithmic** $O(log(n))$
  - **Linear** $O(n)$
  - **Linearithmic** $O(nlog(n))$
  - **Polynomial** $O(n^2), O(n^3), ...$
  - **Exponential** $O(2^n)$
  - **Factorial** $O(n!)$

Finding Big-O

- Consider time complexity of an algorithm
  - Steps in algorithm relative to input size
- Find upper bound
  - See last class / beginning of this class
  - Try to use growth functions (or combination thereof)
  - Use simplifying assumptions

Simplifying Assumptions

- Constants **DON'T** matter
  - $f(n) = O(cf(n)) \rightarrow f(n) = O(f(n))$
- Lower order terms **DON'T** matter
  - $f(n) = O(f(n)) \wedge g(n) = O(g(n)) \rightarrow$
    $f(n) + g(n) = O(max(f(n), g(n)))$
- Product of functions **DO** matter
  - $f(n) = O(f(n)) \wedge g(n) = O(g(n)) \rightarrow$
    $f(n) * g(n) = O(f(n) * g(n))$

Example of Simplifying Assumptions

- $3x^4 + 18x^3 + 32x^2 + 4x + 20 = O(?)$
- $(3n + 8) * (4n + 12) = O(?)$
- $3n^2 * 10n^2 = O(?)$
- $20 + 15 + 12 = O(?)$

Example of Simplifying Assumptions

- $3x^4 + 18x^3 + 32x^2 + 4x + 20 = O(x^4)$
- $(3n + 8) * (4n + 12) = O(n^2)$
- $3n^2 * 10n^2 = O(n^4)$
- $20 + 15 + 12 = O(1)$

- **Note**: Even though $2x = O(n^4)$, we wish to form a **tight-bound** that is the smallest Big-O growth
  - Thus, we would prefer to use O(x)

Practice

- Find the runtime and Big-O of these programs
- ```
  sum := 0
  for j := 1 to n
      for i := 1 to n
      sum := sum + 1
  for k := 0 to n
      a[k] := k
  ```
- ```
  i := 1
  k := 0
  while i ≤ k
      k := k + 1
      i := i*2
  ```

P vs. NP

- All **tractable** problems are in **P**
- All **intractable** problems are in **NP**
  - Problems in NP are “hard” problems where approximations are generally accepted
  - Tractable if some magic computer (“oracle”) gave us an answer we only have to verify
- It has not been proven that P ≠ NP, but it is widely believed to be true
  - Proving P = NP makes problems like AI easier, but breaks many fundamentals in cryptography