Asymptotic Notation

COMP215: Design & Analysis of Algorithms

Today

- The Gist
- Big-O Notation
- Examples

Asymptotic Notation

- Asymptotic notation provides the basic vocabulary for discussing the design and analysis of algorithms.

- Asymptotic notation is coarse enough to suppress all the details you want to ignore, details that depend on
  - The choice of architecture,
  - The choice of programming language,
  - The choice of compiler.

- It is useful to make comparisons between different highlevel algorithmic approaches to solving a problem, especially on larger input

Asymptotic Notation

Asymptotic Notation in Seven Word:
suppress constant factors and lower-order terms

system-dependent

irrelevant for large inputs

Asymptotic Notation - Examples

- Consider first the problem of searching an array for a given integer t. The code just checks each array entry in turn. If it ever finds the integer t it returns true, and if it falls off the end of the array without finding t it returns false.

Searching One Array

Input: array A of n integers, and an integer t.
Output: Whether or not A contains t.

```
for i := 1 to n do
    if A[i] = t then
        return TRUE
return FALSE
```

- What is the asymptotic running time of the code for searching one array, as a function of the array length n?

Asymptotic Notation - Examples

- Suppose we're now given two integer arrays A and B, both of length n, and we want to know whether a target integer t is in either one. Let's again consider the straightforward algorithm, where we just search through A, and if we fail to find t in A, we then search through B. If we don't find t in B either, we return false.

- What is the asymptotic running time of the code for searching Two arrays, as a function of the arrays length n?

Searching Two Arrays

Input: arrays A and B of n integers each, and an integer t.
Output: Whether or not A or B contains t.

---

for $i := 1$ to n **do**
    if $A[i] = t$ **then**
        return TRUE
for $i := 1$ to n **do**
    if $B[i] = t$ **then**
        return TRUE
return FALSE

Asymptotic Notation - Examples

- Suppose we want to check whether or not two given arrays of length n have a number in common. The simplest solution is to check all possibilities. That is, for each index i into the array A and each index j into the array B, we check if A[i] is the same number as B[j]. If it is, we return true. If we exhaust all the possibilities without ever finding equal elements, we can safely return false.

- What is the asymptotic running time of the code for checking for a common element, as a function of the arrays length n?

Checking for a Common Element

Input: arrays A and B of n integers each.
Output: Whether or not there is an integer t contained in both A and B.

```
for i := 1 to n do
    for j := 1 to n do
        if A[i] = B[j] then
            return TRUE
return FALSE
```

Asymptotic Notation - Examples

- Suppose we're looking for duplicate entries in a single array A, rather than in two different arrays.

Checking for Duplicates

Input: array A of n integers.
Output: Whether or not A contains an integer more than once.

```
for i := 1 to n do
    for j := i + 1 to n do
        if A[i] = A[j] then
            return TRUE
return FALSE
```

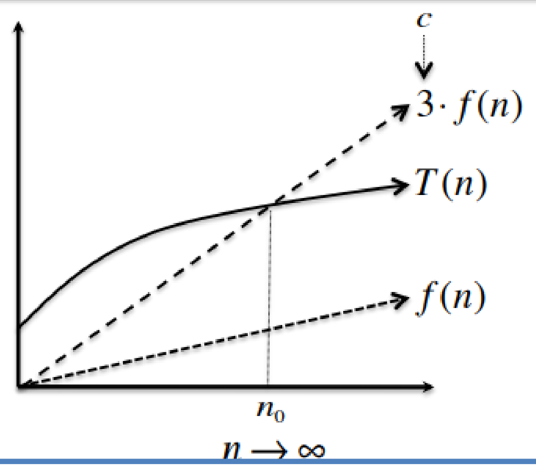
- What is the asymptotic running time of the code for checking for duplicates in one array, as a function of the array length n?

Big-O Notation

- The formal definition of big-O notation:

Big-O Notation (English Version)

T(n) = O(f(n)) if and only if T(n) is eventually bounded above by a constant multiple of f(n)



Big-O Notation (Mathematical Version)

T(n) = O(f(n)) if and only if there exist positive constants c and n_0 such that T(n) <= c · f(n) (2.1)

for all n>= n_0.

Big-O Notation

- If you want to prove that T(n) = O(f(n)), then your task is to choose the constants **c** and **n_0** so that (2.1) holds whenever n>= $n_{0.}$

- **OR:**

- If T(n) = O(f(n)), then there are constants c and n_0 such that (2.1) holds for all n >= n_0

Big-O Notation (Examples)

- Degree-k Polynomials are $O(n^k)$
- Degree-k Polynomials Are Not $O(n^{k-1})$

Big-O Notation (Examples)

1. Degree-k Polynomials are $O(n^k)$

- If T(n) is a polynomial with some degree k, then $T(n) = O(n^k)$. How?

$$T(n) = a_k n^k + \cdots a_1 n + a_0,$$

where $k \geq 0$ is a nonnegative integer and the a_i's are real numbers (positive or negative). Then $T(n) = O(n^k)$.

- Proposition 2.1 says that with a polynomial, in big-O notation, all you need to worry about is the highest degree that appears in the polynomial. Thus, big-O notation really is suppressing constant factors and lower-order terms. Prove?

Big-O Notation (Examples)

- Find c and n_0.
  1. Try n_0 =1 and c equal to the sum of absolute values of the coefficients: c = $|a_k| + \cdots + |a_1| + |a_0|$.
  2. We now need to show that these choices of constants satisfy the definition, meaning that **T(n) <= c*n^k for all n >=n_0 = 1.**
  3. To verify this inequality, fix an arbitrary positive integer n n0 = 1. We need a sequence of upper bounds on T(n) (for coefficients and power of n), culminating in an upper bound of c · nk. First let's apply the definition of T(n):

$$T(n) = a_k n^k + \cdots a_1 n + a_0,$$

Big-O Notation (Examples)

4. For coefficients, if we take the absolute value of each coefficient ai on the right-hand side, the expression only becomes larger

$$T(n) \leq |a_k|n^k + \cdots + |a_1|n + |a_0|.$$

5. For power of n, n^k is only bigger than n^i for every i in {0, 1, 2,...,k}
6. Since $|a_i|$ is nonnegative, $|a_i|n^k$ is only bigger than $|a_i|n^i$. This means that

$$T(n) \leq |a_k|n^k + \cdots + |a_1|n^k + |a_0|n^k = \underbrace{(|a_k| + \cdots + |a_1| + |a_0|)}_{=c} \cdot n^k.$$

Big-O Notation (Examples)

- **Degree-k Polynomials Are Not $O(n^{k-1})$**
- **Proposition 2.2** Let k 1 be a positive integer and define $T(n) = n^k$. Then $T(n)$ is not $O(n^{k-1})$.
- Proof by contradiction:
  - Assume that n^k is in fact $O(n^{k-1})$, for all $n > n_0$
  - That is, there are positive constants c and n_0 such that $\mathbf{n^k <= c \cdot n^{k-1}}$
  - Cancel n^{k-1} from both sides of this inequality to derive $\mathbf{n <= c}$**,** for all $n > n_0$

False statement

Big-O Notation

- Practice:
  - Arrange the following functions in order of increasing growth rate, with g(n) following f(n) in your list if and only if f(n) = O(g(n)).

a) $2^{\log_2 n}$

b) $2^{2^{\log_2 n}}$

c) $n^{5/2}$

d) 2^{n^2}

e) $n^2 \log_2 n$

Big-Omega and Big-Theta Notation

- Big-O is analogous to “less than or equal to (≤),”
- Big-omega is analogous to “greater than or equal(≥)
- Big-theta is “equal to (=),”

O	Ω	Θ
Big Oh	Omega	Theta
Upper Bound	Lower Bound	Upper & Lower Bound
Worst Case	Best Case	Average ‘ish’

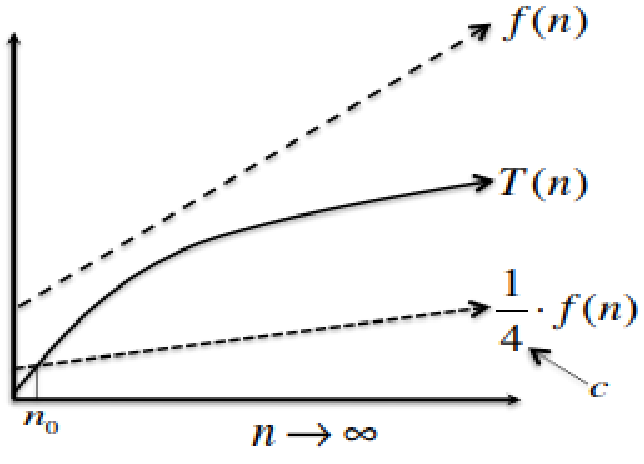
Big-Omega

Big-Omega Notation (Mathematical Version)

$T(n) = \Omega(f(n))$ if and only if there exist positive constants c and n_0 such that

$$T(n) \geq c \cdot f(n)$$

for all $n \geq n_0$.



KALAMAZOO COLLEGE

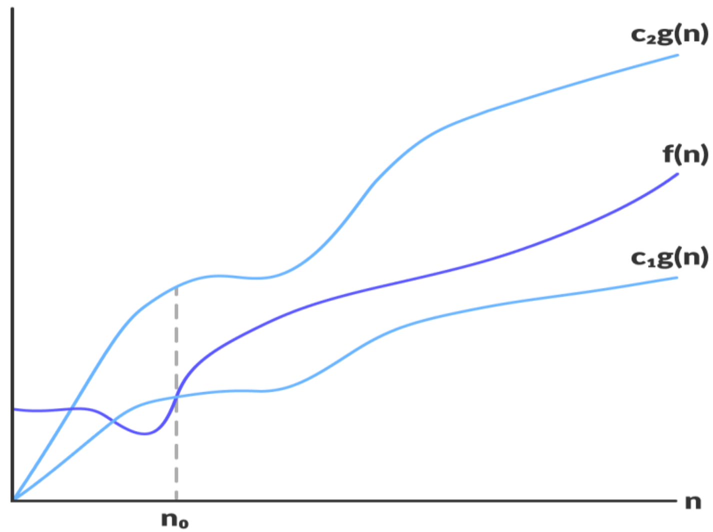
Big-Theta

Big-Theta Notation (Mathematical Version)

$T(n) = \Theta(f(n))$ if and only if there exist positive constants c_1, c_2, and n_0 such that

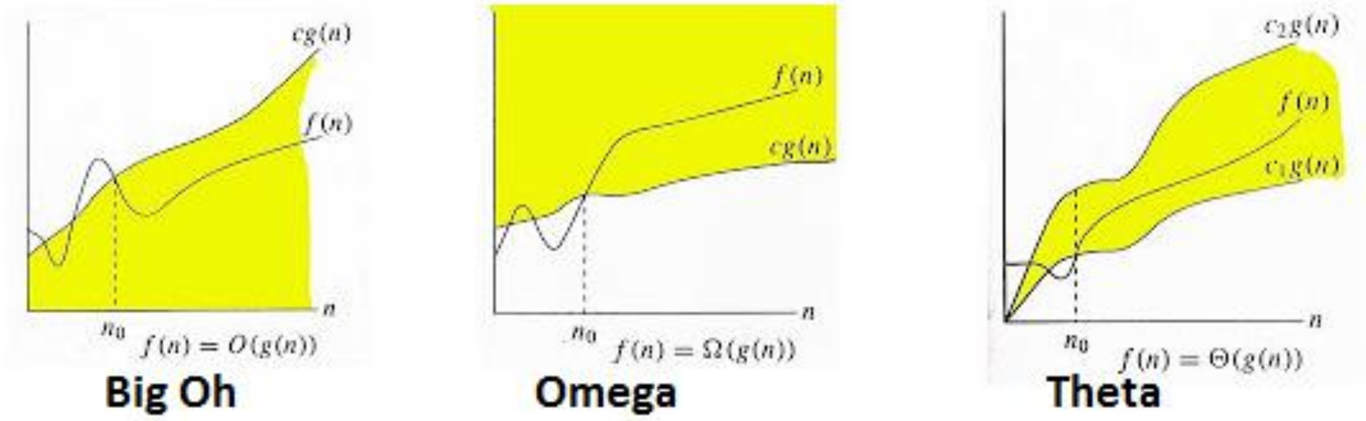
$$c_1 \cdot f(n) \leq T(n) \leq c_2 \cdot f(n)$$

for all $n \geq n_0$.



$f(n) = \Theta(g(n))$

Asymptotic Notation



Asymptotic Notation

Quiz 2.5

Let $T(n) = \frac{1}{2}n^2 + 3n$. Which of the following statements are true? (There might be more than one correct answer.)

a) $T(n) = O(n)$

b) $T(n) = \Omega(n)$

c) $T(n) = \Theta(n^2)$

d) $T(n) = O(n^3)$

Asymptotic Notation

- True or False?

If $f(n) = O(g(n))$ and $g(n) = O(h(n))$, then $h(n) = \Omega(f(n))$

If $f(n) = O(g(n))$ and $g(n) = O(f(n))$ then $f(n) = g(n)$