CSCI 335
Software Design and Analysis III
Lecture 3: Part 2 Analysis of Algorithms

Professor Anita Raja

1

1

Agenda

- Big Five
- C++11: Templates/Matrices
- **Asymptotic Analysis of Algorithms**
- Assignment 1 Discussion

2

2

Mathematical Background: Exponents

- Law of exponents
- Derived from basics definitions of multiplication, division and exponents.

$$a^x \cdot a^y = a^{x+y}$$
$$a^x \div a^y = a^{x-y}$$
$$(a^x)^y = a^{xy}$$

3

3

Mathematical Background: Logarithms

- Rules of logarithms
- Derived from rules of exponents

$$\log xy = \log x + \log y$$
$$\log x/y = \log x - \log y$$
$$\log x^y = y \log x$$
$$\log_a x = \frac{\log_b x}{\log_b a}$$

4

4

Mathematical Background: Logarithms

$$log x^y = y \log x$$

Proof:

$$\text{Let } x = e^z$$
$$x^y = (e^z)^y = e^{zy}$$
$$\log x^y = log\ e^{zy} = zy$$
$$log\ x^y = y\ log\ x$$

5

5

Mathematical Background: Logarithms

$$log_a x = \frac{log_b x}{log_b a}$$

Proof:

$$\text{Let } x = a^z$$
$$log_b\, x = log_b\ a^z = z\, log_b\, a$$
$$\text{But } z = log_a x\ = \frac{log_b x}{log_b a}$$

6

6

Mathematical Background: Series

$$\sum_{i=0}^{N} 2^i = 2^{N+1} - 1$$
$$\sum_{i=0}^{N} A^i = \frac{A^{N+1} - 1}{A - 1}$$
$$\sum_{i=1}^{N} i = \frac{N(N+1)}{2} \approx \frac{N^2}{2}$$
$$\sum_{i=1}^{N} i^2 = \frac{N(N+1)(2N+1)}{6} \approx \frac{N^3}{3}$$

7

7

Mathematical Background: Modulo

- Euclidean division is defined by the equation:

$$a = qb + r$$

$$a \text{ div } b = q$$
$$a \bmod b = r = a\ - qb$$

- Note: $a \bmod b = a$ for $1 \leq a < b$, not 0!
- For example: : $3 \bmod 11 = 3\ \neq 0$

8

8

Motivation

- **Algorithm:**
- clearly specified set of simple instructions to be followed to solve a problem.
  - Important step: determine how much in the way of resources, such as time and space, the algorithm will require.

9

9

Big-Oh Notation

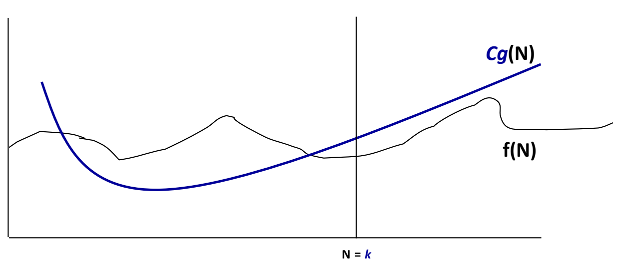
- Special notation to define upper bounds and lower bounds of functions.
- In CS, usually the functions we are bounding are running times and memory requirements.
  - Also refer to the running time as T(n).

10

10

The Growth of Functions

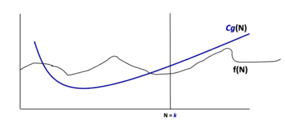
- Establish relative order of functions:
  - Compare relative rate of growth.
- We accept the constant C in the requirement
  - $f(N) \le C \cdot g(N)$ whenever $N > k$, because **C does not grow with N.**
- We are only interested in large n, so it is OK if $f(N) > C \cdot g(n)$ for $n \le k$.



11

11

Relative Rates of Growth

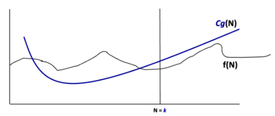


Analysis Type	Mathematical Expression	Relative Rates of Growth
Big O	f(N) = O(g(N))	f(N) $\le$ g(N)
Small o	f(N) = o(g(N))	f(N) < g(N)
Big Ω	f(N) = Ω(g(N))	f(N) $\ge$ g(N)
Big θ	f(N) = θ(g(N))	f(N) = g(N)

12

12

What it All Means



- f(N) is the actual growth rate of the algorithm.
- g(N) is the function that bounds the growth rate:
  - may be upper or lower bound
- f(N) may not equal g(N):
  - constants and lesser terms ignored because it is a *bounding function*

13

13

Smallest upper bound

- Question: If f(x) is $O(x^2)$, is it also $O(x^3)$?

14

14

Definitions

- For *N* greater than some constant, we have the following definitions:

$$T(N) = O(f(N)) \leftarrow T(N) \leq cf(N)$$

Upper bound on T(N)

$$T(N) = \Omega(g(N)) \leftarrow T(N) \geq cg(N)$$

Lower bound on T(N)

$$T(N) = \Theta(h(N)) \leftarrow \begin{matrix} T(N) = O(h(N)), \\ T(N) = \Omega(h(N)) \end{matrix}$$

Tight bound on T(N)

- There exists some constant c such that cf(N) bounds T(N)

15

15

Definitions Continued

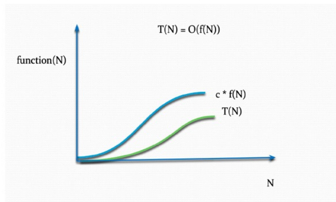
- $T(N) = O\big(f(N)\big)$ if there are positive constants c and n_0 such that
  $T(N) \leq cf(N)$ when $N \geq n_0$
- $T(N) = \Omega\big(g(N)\big)$ if there are positive constants c and n_0 such that
  $T(N) \geq cg(N)$ when $N \geq n_0$
- $T(N) = \Theta\big(h(N)\big)$ if and only if $T(N) = O\big(h(N)\big)$ *and* $T(N) = \Omega(h(N))$
- $T(N) = O\big(p(N)\big)$ if, for all positive constants c, there exists an n_0 such that
  $T(N) < cp(N)$ when $N > n_0$.
  Les formally, $T(N) = o\big(p(N)\big)$ if $T(N) = O\big(p(N)\big)$ and $T(N) \neq \Theta(p(N))$

16

16

Example

- $N = O(N^2)$
- $4N^2 + N = O(N^2)$
- $\log N = O(N)$

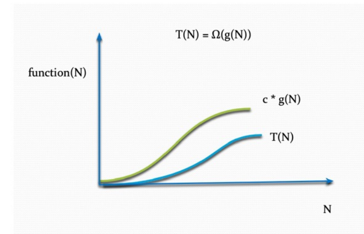


17

17

Example

- $N^2 = \Omega(N^2)$
- $N = \Omega(\log N)$

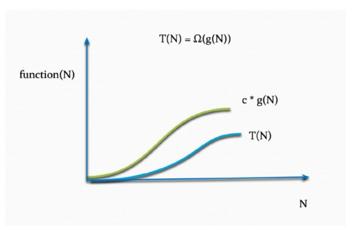


18

18

Example

- $5N^3 + \mathrm{N} = \Theta\,(N^3)$
- $0.2 \log \mathrm{N} + \log\log \mathrm{N} = \Theta\,(\log \mathrm{N})$



19

19

Definition

- Alternatively, $O(f(N))$ can be thought of as meaning
$$T(N) = O(f(N)) \leftarrow \lim_{N\to\infty} f(N) \geq \lim_{n\to\infty} T(N)$$
- The idea of these definitions is to establish a relative order among functions. Thus, we compare their **relative rates of growth.**
- **Note:** Big-Oh notation is also referred to as asymptotic analysis for this reason.

20

20

The Growth of Functions

- A problem that can be solved with polynomial worst-case complexity is called **tractable**:
  - Searching an ordered list, Sorting a list, Integer multiplication.
- Problems of higher complexity are called **intractable:**
  - Factoring a number into primes. (this is why RSA and public key encryption is successful)
  - SAT, the satisfiability problem to test whether a given Boolean formula is satisfiable
  - Graph coloring, bin packing.
- Problems that no algorithm can solve are called **unsolvable**:
  - P?NP.
  - Integer factorization in polynomial time.
  - Fastest algorithm for matrix multiplication, multiplication of 2 n-digit numbers.

21

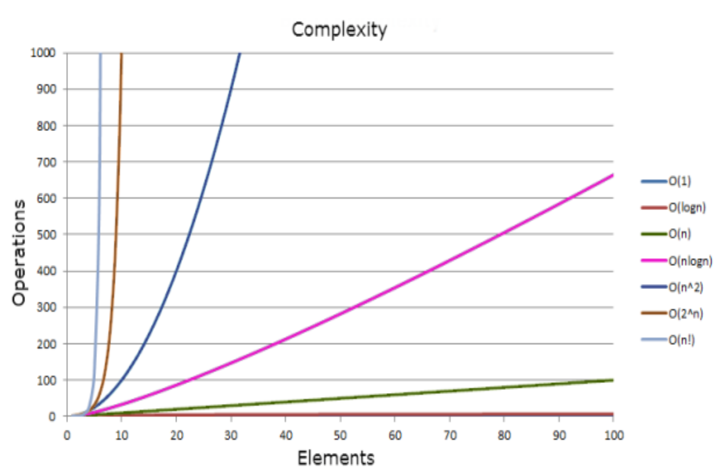
21

Typical growth rates

Function	Name
c	Constant
$\log N$	Logarithmic
$\log^2 N$	Log-squared
N	Linear
$N \log N$	
N^2	Quadratic
N^3	Cubic
2^N	Exponential

22

22



23

23

Model of computation

- Model needed for algorithm analysis
  - Note algorithm not C++/code analysis
- One unit of time corresponds to simple instructions:
  - Addition, multiplication, comparison, assignment.
- Fixed size (32-bit) integers.
- Infinite memory
  - Interest in algorithm itself, not performance in any particular machine.

24

24

What to analyze

- Running time (Time complexity)
- Memory usage (Space complexity)
- What kind of input?
  - Worst-case running time
  - Best-case running time
  - Average-case running time
    - Sometimes it is hard to define what is the average input.
- Size of the input problem, N.
  - If input is an array, N is the size of the array.
  - If input is a number, N is the number of bits used to represent it.

25

25

Useful Rules for Big-O: Summing Execution Times

- If f(N) is a polynomial of degree k, then $f(N) = \theta(N^k)$
- If $T_1(N)=O(f(N))$ and $T_2(N) = O(g(N))$, then
  - $T_1(N)) + T_2(N) = O(f(N)+g(N))$
  - $T_1(N)) .T_2(N) = O(f(N).g(N))$
- $\log^k (N) = O(N)$ for any constant k.
  - logarithms grow very slowly
- If an algorithm's execution time is $N^2 + N$ then it is said to have $O(N^2)$ execution time, not $O(N^2 + N)$.
- Formally, a function f(N) dominates a function g(N) if there exists a constant value n_0 such that for all values $N > N_0$ it is the case that $g(N) < f(N)$.

26

26

Useful Rules for Big-O

- When adding algorithmic complexities the larger value dominates.

For any **polynomial** $f(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_0$, where $a_0, a_1, \ldots, a_n$ are real numbers, f(x) is $O(x^n)$.

- If $f_1(x)$ is $O(g_1(x))$ and $f_2(x)$ is $O(g_2(x))$, then $(f_1 + f_2)(x)$ is $O(\max(g_1(x), g_2(x)))$
- If $f_1(x)$ is $O(g(x))$ and $f_2(x)$ is $O(g(x))$, then $(f_1 + f_2)(x)$ is $O(g(x))$.

27

27

Useful Rules for Big-O

- Loops:
  - Running time of a for loop is at most the running time of the statements inside the for loop times the number of iterations.
- Nested loops:
  - Analyze inside out. Running time of statement multiplied by the product of the sizes of the loops.
- Consecutive statements:
  - Just add.
- If/Else:
  - Never more than the running time of the test plus the larger of the running times of S1 and S2.

28

28

Simple example

- Simple example. Running time?

```
// @n: a positive integer
// @returns 1^3 + 2^3 + … + n^3
//     Will return 0, if n is smaller than 1.
int SumOfCubes(int n) {
  int sum_of_cubes = 0;
  for (int i = 1; i <= n; ++i)
    sum_of_cubes += i * i * i;
  return sum_of_cubes;
}
```

29

Simple example

- Simple example. Running time?

```
// @n: a positive integer
// @returns 1^3 + 2^3 + … + n^3
//     Will return 0, if n is smaller than 1.
int SumOfCubes(int n) {
  int sum_of_cubes = 0;                  O(1)
  for (int i = 1; i <= n; ++i)         O(n times)
    sum_of_cubes += i * i * i;         O(4)
  return sum_of_cubes;                   O(1)
}
                     F(n) = 1 + 4 * n + 1 = O(n)
```

30

Recursion

- Factorial:

```
// @n: an integer
// @return n * (n – 1) * … * 1, if n >= 2
//         1, otherwise.
long Factorial(int n) {
  if (n <= 1)
     return 1;
  else
     return n * Factorial(n – 1);
}
```

- Running time?

31

Summary

- Algorithmic Complexity review
- No class Monday
- Thursday 9/8 class:
  - Factorial, Fibonacci, Maximum subsequence problem
  - Read 3.1-3.3 for next class.
  - Verification of Enrollment Survey will go online.

32

33

Using limits

In order to determine the relative growth rate of two functions f(N) and g(N) we can compute the limit:

$$\lim_{N \to \infty} f(N)/g(N)$$

If the limit is

Zero: f(N) = o(g(N))
C ≠ 0 : f(N) = Ɵ (g(N)
∞ : g(N) = o(f(N))
Oscillation: no relation; limit does not exist.

34

L'Hôpital's Rule

If $\lim_{N\to\infty} f(N) = \infty$ and $\lim_{N\to\infty} g(N) = \infty$

$$\lim_{N\to\infty} \frac{f(N)}{g(N)} = \lim_{N\to\infty} \frac{f'(N)}{g'(N)}$$

Where $f'(N)$ and $g'(N)$ are derivatives of $f(n)$ and $g(n)$ respectively.

35

L'Hopital's Rule Examples

- Example 1: $f(N) = N^2$ and $g(N) = N$

$$\lim_{N\to\infty} N^2 = \infty? \text{ and } \lim_{N\to\infty} N = \infty?$$

$$\lim_{N\to\infty} \frac{N^2}{N} = \lim_{N\to\infty} \frac{f'(N) = 2N}{g'(N) = 1} = \lim_{N\to\infty} 2N = \infty$$

Therefore $N = o(N^2)$

- Example 2: $f(N) = \log N$ and $g(N) = N$

$$\lim_{N\to\infty} \log N = \infty? \text{ and } \lim_{N\to\infty} N = \infty?$$

$$\lim_{N\to\infty} \frac{\log N}{N} = \lim_{N\to\infty} \frac{f'(N) = 1/N}{g'(N) = 1} = \lim_{N\to\infty} \frac{1}{N} = 0$$

Therefore, $\log \mathrm{N} = o(N)$

36