Algorithm Analysis

CSCI 335

Software Design and Analysis (Chapter 2)

1

Motivation

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

9

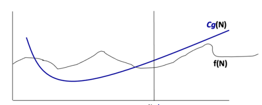
Big-Oh Notation

- We adopt a special notation to define **upper bounds** and **lower bounds** on functions.
- In Computer Science, usually, the functions we are bounding are running times and memory requirements.
- We refer to the running time as $T(N)$

10

The Growth of Functions

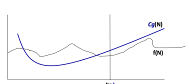
- Establish relative order of functions:
  - Compare relative rate of growth
- We accept the constant C in the requirement
  - $f(N) \leq Cg(N)$ whenever $N > k$, because C does not grow with N.
- We are only interested in large n, so it is ok if $f(N) > Cg(n)$ for $n \leq k$



11

Relative Rates of Growth

Analysis Type	Mathematical Expression	Relative Rates of Growth
Big O	$f(N) = O(g(N))$	$f(N) \leq g(N)$
Small o	$f(N) = o(g(N))$	$f(N) < g(N)$
Big Ω	$f(N) = \Omega(g(N))$	$f(N) \geq g(N)$
Big θ	$f(N) = \Theta(g(N))$	$f(N) = g(N)$



12

What does this mean?

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

13

Smallest Upper Bound

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

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

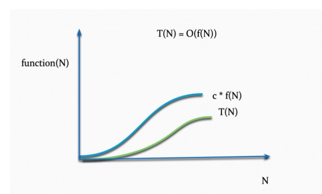
Continuation: Definitions

- 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.

16

Example

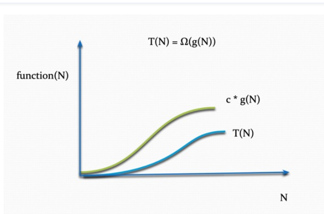
- $N = O(N^2)$
- $4N^2 + N = O(N^2)$
- $\log N = O(N)$



17

Example

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



18

Example

- $5N^3 + N = \Theta(N^3)$
- $0.2 \log N + \log\log N = \Theta(\log N)$

19

Exercise

- Order the following functions by growth rate:

N, $N^{1/2}$, $N^{1.5}$, N^2, $N \log N$, $N \log \log N$, $N \log^2 N$, $N \log(N^2)$, $2/N$, 2^N, $2^{N/2}$, 37, $N^2 \log N$, N^3

20

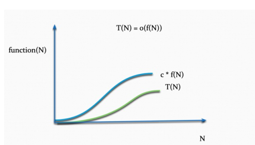
Definition: Little-Oh Notation

- $T(N) = o(p(N))$ if, for all positive constants c, there exists an n_0 such that $T(N) < cp(N)$ when $N > n_0$.
  Les formally, $T(N) = o(p(N))$ if $T(N) = O(p(N))$ and $T(N) \neq \Theta(p(N))$
- The growth rate of $T(N)$ is less (<) than the growth rate of $p(N)$
- In $O()$ we had $\leq$ and in $\Theta()$ we had both $\leq$ and $\geq$

21

Example

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



22

Exercise:

- If $T(N) = 2N$
- Then:
- $T(N) = o(N)$?
- $T(N) = O(N)$?
- $T(N) = o(N^3)$?
- $T(N) = O(N^3)$?

23

Exercise:

- If $T(N) = 4 \log \mathrm{N} + 10$
- Then:
- $T(N) = O(\log N)$?
- $T(N) = \Theta(\log N)$?
- $T(N) = o(\log N)$?
- $T(N) = o(N)$?
- $T(N) = O(N)$?

24

Important definitions!

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

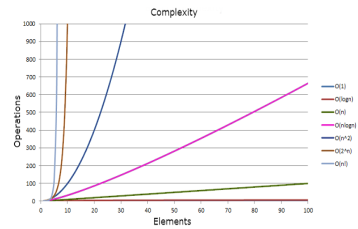
25

Growth 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, SAT, 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

26

Growth rates



27

Comparing Growth Rates

$$T_1(N) = O(f(N)) \text{ and } T_2(N) = O(g(N))$$

then **RULE 1**

$$\begin{array}{ll} (a) & T_1(N) + T_2(N) = O(f(N) + g(N)) \\ (b) & T_1(N)T_2(N) = O(f(N)g(N)) \end{array}$$

28

Comparing Growth Rates

- **Rule 2:**

If $T(N)$is a polynomial of degree k, then $T(N) = \Theta(N^k)$

29

Comparing Growth Rates

- **Rule 3:**

$log^k N = O(N)$for any constant k.
This tells us logarithms grow very slowly.

30

Using the notation

- It is considered bad style to include constants or low-order terms inside Big-Oh. For example: do not write $T(N) = O(2N^2)$ or $T(N) = O(N^2 + N)$.
  - Lower order terms and constants do not affect the growth rate.
- If it is known that $T(N) = O(N^2)$ then even though it would be true to write $T(N) = O(N^3)$, it would not be considered a good estimate for a growth rate of $T(N)$

31

Using limits

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

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

- The limit can have four possible values:
  - The limit is 0: This means that $f(N) = o(g(N))$
  - The limit is $c \neq 0$: This means that $f(N) = \Theta(g(N))$
  - The limit is ∞: This means that $g(N) = o(f(N))$
  - The limit does not exist: There is no relation

32

L'Hôpital's Rule

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

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

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

33

Example

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

34

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

35

Model of computation

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

36

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.

37

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 $O(g(x))$.

38

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.

39

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;
}
```

$O(n)$, why?

40

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?

41

Recursion

- Factorial running time:

$$T(n) = 1 + T(n-1) for\, n > 1, T(1) = 2$$

———————————————————————————— –

$$T(n) = 1 + T(n-1) = 1 + (1 + T(n-2)) = \ldots =$$
$$= 1 + (1(+\cdots(1 + T(n-k))..)\ (k\ 1's)$$
$$=> T(n) = k + T(n-k) = (n-1) + T\big(n-(n-1)\big) = (n-1) + T(1)$$
$$= (n-1) + 2 = n + 1 => T(n) = n \;=> T(n) = O(n).$$

- Linear algorithm.

42

Recursion

- Fibonacci (bad example):

```
// @n: an integer.
// @return the Fibonacci number of n.
long Fibonacci(int n) {
    if (n <= 1)
      return 1;
   else
     return Fibonacci(n - 1) + Fibonacci(n - 2);
}
```

- Running time?

43

Recursion

- Fibonacci (bad example):
- Running time: $T(n) = T(n-1) + T(n-2) + 2, T(0) = T(1) = 1$

- We can prove (by induction) that $T(n) > 1.5^n$ for $n > 2$

  => $T(n) = ?\ (1.5^n)\quad O/\Theta/\Omega/o$

Section 1.2.5 proves that $T(n) < (5/3)^n$

=> $T(n) = ?\ ((5/3)^n)\quad O/\Theta/\Omega/o$

44

Fibonacci

N	T(n)	1.5^n
0	1	1.00
1	1	1.50
2	4	2.25
3	7	3.38
4	13	5.06

- We can see than $T(n) > 1.5\hat{}n$ for small values of n
- Induction:

$$T(n) = T(n-1) + T(n-2) + 2 >$$
$$T(n-1) + T(n-2) >$$
$$(ind.Hyp.)1.5\hat{}(n-1) + 1.5\hat{}(n-2) = 2.5 * 1.5\hat{}(n-2)$$
$$> 2.25 * 1.5\hat{}(n-2) = 1.5\hat{}2 * 1.5\hat{}(n-2) = 1.5\hat{}n.$$

Exponential: Really bad result! (Use iteration instead)

45

Homework 1

Quick overview

68

Example: Part 1

- Input:

```
3 7 4 5 19 2 3
4 100 101 3 40 11 33 77
4
```

69

Output: Part 1

```
3 7 4 5 19 2 3
4 100 101 3 40 11 33 77
4
```

INPUT

```
0 0
(7, 10)
Enter a sequence of points (integer)

Output1:
(7, 4) (5, 19) (2, 3)
Enter a sequence of points (integer)

Output2:
(100, 101) (3, 40) (11, 33) (77, 4)
After copy constructor1 c{a}:
(7, 4) (5, 19) (2, 3)
(7, 4) (5, 19) (2, 3)
After assignment a = b
(100, 101) (3, 40) (11, 33) (77, 4)
After e = move(c)
(7, 4) (5, 19) (2, 3)
()
After a = move(e)
 (7, 4) (5, 19) (2, 3)
 (100, 101) (3, 40) (11, 33) (77, 4)
```

70

Example: Part 2

- Input:

```
3 2.1 20.3     12.45 13.1 34.3 54.4
2 1.1 100.0 30.2 22.3
```

71

Output: Part 2

```
3 2.1 20.3     12.45 13.1 34.3 54.4
2 1.1 100.0 30.2 22.3
INPUT
```

```
Enter a sequence of points (double)

(2.1, 20.3) (12.45, 13.1) (34.3, 54.4)
Enter a sequence of points (double)

(1.1, 100) (30.2, 22.3)

Result of a + b
(3.2, 120.3) (42.65, 35.4) (34.3,
54.4)

Result of d = a + b
(3.2, 120.3) (42.65, 35.4) (34.3,
54.4)
Second element in a:
12.45, 13.1
```

72

Overloading: extraction and insertion

Let go to the board!

73

HW1 FAQ

Let go over some questions for HW1

74

How do we add sequences of different sizes?

- Suppose A = (1, 2) (3, 4) (5, 6) and B = (7, 8) (9, 10). Then A + B = (8, 10) (12, 14) (5, 6).
- Since A has one more point than B, we can't add every point in A to a corresponding point in B. Therefore, we add nothing to A's remaining point and put it in the resultant array of points.

75

What type does `sequence_` have -- how do we store more than two points? Is the type correct?

- `sequence_` is a pointer as denoted by the asterisk. If we have multiple points we will need to store objects of type array<object, 2> somewhere. The type is correct. Recall that with dynamically allocated arrays, we can set the array length at the time of allocation i.e. base it off some information we get from the user.

76

What do we submit and how?

- You will submit a modified `points2.h` (HW 1 Spec., pg3) which implements the desired behavior. The `test_points2.cc` file contains driver code: If the implementation in the `points2.h` file is correct, the output of running the compiled program should match the expected output provided. In this case, "from scratch" (pg 1) means without the use of outside libraries. You should, however, use the provided files.

77

Any other questions?

78