Algorithm Performance (The Big-O)

CMPT 125
Mo Chen
SFU Computing Science
22/1/2020

Lecture 7

Today:

- Barometer instructions
- Manipulating Big-O expressions
- Growth rates of common functions

The Story So Far . . .

- Often consider the *worst-case* behaviour as a benchmark
- Derive total steps (T) as a function of input size (N)
  - use `time` command to measure for various N
  - OR . . . count the elementary operations
- Use Big-*O* to express the growth rate
  - compares algorithms' behaviour as N gets large
  - leading constants are removed
  - a hardware-independent analysis

Leading Constants (Review)

Leading constants are affected by:

- CPU speed
- other tasks in the system
- characteristics of memory
- program optimization

Regardless of leading constants, a $O(N \log N)$ algorithm will outperform a $O(N^2)$ algorithm as N gets large

As N Gets Large, The Algorithm is Most Important

A carefully crafted algorithm can make the difference between software that is usable and useless

- e.g., if it costs a $O(N)$ algorithm 0.5s to search 1 billion bank records, but a $O(\log N)$ algorithm 0.005s
- e.g., or, if 10^9 isn’t “big”, how about Google?
- e.g., real-time computing - where a nearly instant response is required

Optimizing Algorithms

If you find yourself trying all sorts of clever implementation tricks to speed up an algorithm, you should:

- Step back and ask if you're trying to improve a fundamentally inefficient algorithm
- Consider if there might be a better one

. . . but also realize that there might not be!

It's more important to reduce your running time by a factor of N, than by a factor of 10

- both are important, but not equally important

(Informal) Mathematical Definition

- $T(N) = O\big(f(N)\big)$ if and only if there is some positive number M and N_0 such that

$$|T(N)| \leq Mf(N) \text{ for all } N \geq N_0$$

  - $O\big(f(N)\big)$ is an estimate of the **upper bound** of running time $T(N)$

- Equivalently, $T(N) = O\big(f(N)\big)$ if

$$\lim_{N\to\infty} \left|\frac{T(N)}{f(N)}\right| < \infty$$

  - This limit can usually be computed using l'Hopital's rule

(Informal) Mathematical Definition

- Many possible choices for $f(N)$ -- we want the **best** one
  - $5N^2 = O(N^3)$ is correct, but not the most useful.
  - $5N^2 = O(N^2)$ would be the best estimate of running time

- Many possibilities for $T(N)$ -- we want the **worst** one
  - When looking for an item in an array, you may find it right away
  - However, in the worst case you have to go through all elements

Big-*O* and Barometer Instructions

Problem: Given an algorithm, how do you determine its Big-*O* growth rate?

- Rule of Thumb: the frequency of the algorithm's *barometer instructions* will be proportional to its Big-O running time

So, find the most frequent operation(s) and count them!

In General: Count

Q. What is N?

- The number of elements in the array

```
      int dup_chk(int a[], int length) {
1       int i = length;
N + 1     while (i > 0) {
N           i--;
N           int j = i - 1;
i + 1       while (j >= 0) {
i             if (a[i] == a[j]) {
                return 1;
              }
i             j--;
            }
          }
1       return 0;
      }
```

Outside of loop: 2 (steps)

Outer loop: $3N + 1$

Inner loop: $3\texttt{i} + 1$ for all possible `i` from 0 to $N - 1$.

$= 3/2\ N^2 - 1/2\ N$

Grand total $= 3/2\ N^2 + 5/2\ N + 3$

A *quadratic* function!

Function Calls ⇨ Substitute

Iterate through every element and find minimum

```
int range(int A[], int n) {
      int lo = min(A, n);
      int hi = max(A, n);
      return hi-lo;
}
```

$O(N)$

$O(N)$

$O(1)$

$T(N) = O(N) + O(N) + O(1)$

$= O(N)$

Function calls are not elementary operations

- substitute their Big-*O* running times

If / Else ⇒ Max

Iterate through every element to check order

```
int search(int A[], int n, int key) {
    if (!sorted(A, n)) {
        return lsearch(A, n, key);
    } else {
        return bsearch(A, n, key);
    }
}
```

$O(N)$

$O(N)$

Iterate through every element to look for a key (specific value)

$O(\log N)$

Binary search

$T(N) = O(N) + \max(O(N), O(\log N))$

$= O(N) + O(N)$
$= O(N)$

`if` / `else` is not an elementary operation

- pick the largest of the two running times
  - remember this is worst case analysis

Loops → Multiply

```
int max10by10(int a[N][N]) {
    int best = 0;
    for (int u_row = 0; u_row < N-9; u_row++) {
        for (int u_col = 0; u_col < N-9; u_col++) {
            int total = 0;
            for (int row = u_row; row < u_row+10; row++) {
                for (int col = u_col; col < u_col+10; col++) {
                    total += a[row][col];
                }
            }
            best = max(best, total);
        }
    }
    return best;
}
```

x(N-9)

x(N-9)

x10

x10

barometer instructions

$f(N) = 3 \times 10 \times 10 \times (N-9) \times (N-9) = O(N^2)$

Rules of the Big-*O* (Review)

Usually, take the dominant term, remove the leading constant, and put *O*(. . .) around it

- Properties:
    - constant factors don’t matter
    - low-order terms don’t matter

Rules about Polynomials

1. The powers of N are ordered according to their exponents
   - i.e., $N^a = O(N^b)$ if and only if $a \leqslant b$
   - e.g., $N^2 = O(N^3)$, but N^3 is not $O(N^2)$
2. A logarithm grows more slowly than any positive power of N greater than 0
   - e.g., $\log_2 N = O(N^{1/2})$

For most functions, can apply L'Hôpital's Rule:

- Theorem: If $\lim_{N\to\infty} \frac{f(N)}{g(N)}$ exists then $f(N) = O(g(N))$

Example: $\log N$ vs. N^a, $a > 0$

$$\lim_{N\to\infty} \frac{\log N}{N^a} = \lim_{N\to\infty} \frac{N^{-1}}{aN^{a-1}}$$

$$= \frac{1}{a} \lim_{N\to\infty} N^{-1-(a-1)}$$

$$= \frac{1}{a} \lim_{N\to\infty} N^{-a}$$

$$= \frac{1}{a} \lim_{N\to\infty} \frac{1}{N^a}$$

$$= 0 < \infty$$

More Rules

3. Transitivity: if $f(N) = O(g(N))$ and $g(N) = O(h(N))$ then $f(N) = O(h(N))$

4. Addition: $f(N) + g(N) = O(\max(f(N), g(N)))$

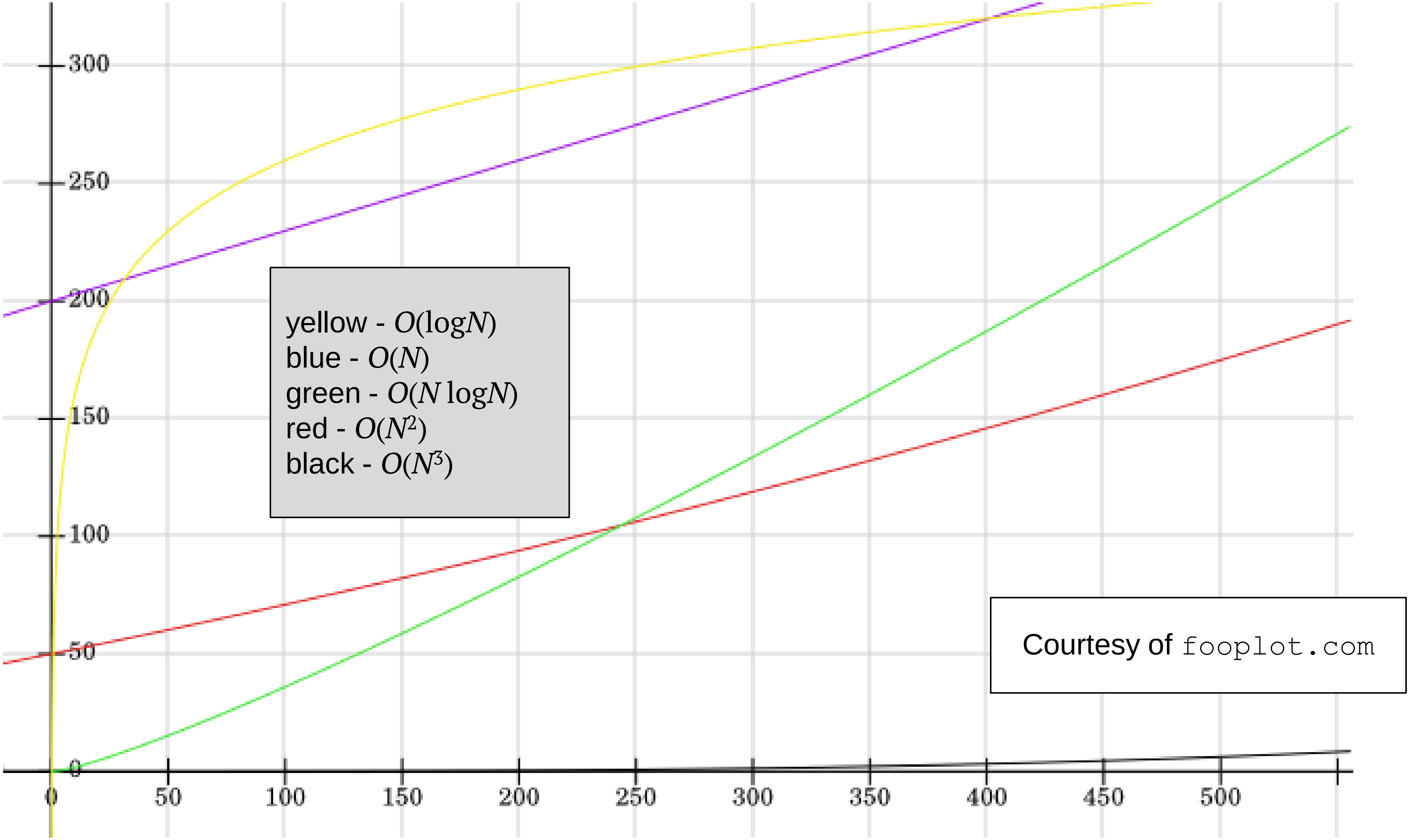
5. Multiplication: if $f_1(N) = O(g_1(N))$ and $f_2(N) = O(g_2(N))$ then $f_1(N) * f_2(N) = O(g_1(N) * g_2(N))$

e.g., $\underbrace{\underbrace{\underbrace{(10 + 5N^2)}_{O(N^2)}\underbrace{(10\log_2 N + 1)}_{O(\log N)}}_{O(N^2 \log N)} + \underbrace{\underbrace{(5N + \log_2 N)}_{O(N)}\underbrace{(10N + 2N\log_2 N)}_{O(N \log N)}}_{O(N^2 \log N)}}_{O(N^2 \log N)}$

Typical Growth Rates

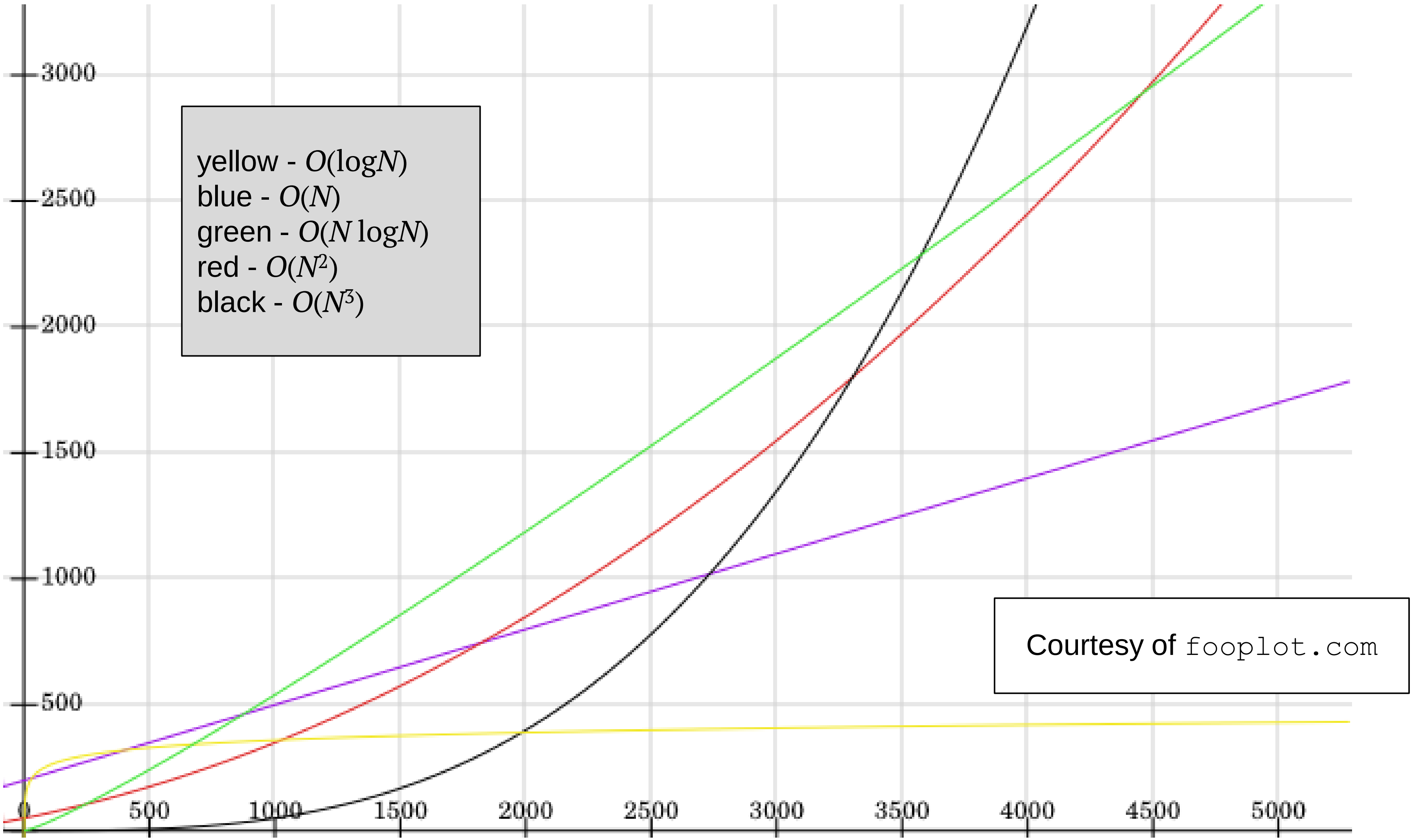
- $O(1)$ – *constant* time
  - The time is independent of N, e.g., array look-up
- $O(\log N)$ – *logarithmic* time
  - Usually the log is to the base 2, e.g., binary search
- $O(N)$ – *linear* time, e.g., linear search
- $O(N \log N)$ – e.g., quicksort, mergesort
- $O(N^2)$ – *quadratic* time, e.g., selection sort
- $O(N^k)$ – *polynomial* (where k is a constant)
- $O(2^N)$ – *exponential* time, very slow!

Some Plots



300
250
200
150
100
50
0
0
50
100
150
200
250
300
350
400
450
500
yellow - $O(\log N)$
blue - $O(N)$
green - $O(N \log N)$
red - $O(N^2)$
black - $O(N^3)$
Courtesy of fooplot.com

Some Plots


3000
2500
2000
1500
1000
500
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
yellow - $O(\log N)$
blue - $O(N)$
green - $O(N \log N)$
red - $O(N^2)$
black - $O(N^3)$
Courtesy of `fooplot.com`