Algorithms Lab

Analysis of Algorithms

Zheng-Liang Lu

Department of Computer Science & Information Engineering
National Taiwan University

Introduction

- A problem may be solved by various algorithms.
- We compare these algorithms by measuring their efficiency.
- Adopting a theoretical approach, we identify the growth rate of running time in function of input size n.
- This introduces the notion of time complexity.[1]
- Let's start with the following two examples.

[1] See https://en.wikipedia.org/wiki/Time_complexity. Similar to time complexity, we later turn to the notion of space complexity.

Example 1: SUM

```
...
        int sum = 0, i = 1; // Assign         -> 2.
        while (i <= n) {    // Compare        -> n + 1.
            sum = sum + i;  // Add and assign -> 2n.
            ++i;            // Increase by 1  -> n.
        }
...
```

- Let n be any nonnegative number.
- Then count the number of all runtime operations.
- Note that we ignore declarations in the calculation. (Why?)
- In this case, the total number of operations is $4n + 3$.

Example 2: TRIANGLE

```
*
* *
* * *
* * * *
* * * * *
```

```
...
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= i; j++)
                System.out.printf("*");
            System.out.println();
        }
...
```

- Before counting, I assume that it will be $cn^2 + \cdots$ for some constant c. (Why?)

Big O Notation[2]

- Let $f(n)$ be the time cost of your algorithm, and $g(n)$ be some simple function.
- We define

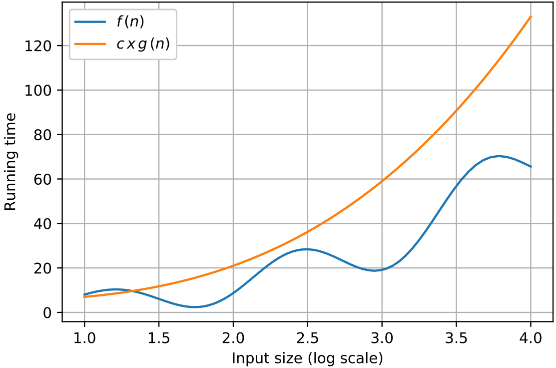
$$f(n) = O(g(n)) \text{ as } n \to \infty$$

provided that there is a constant $c > 0$ and some n_0 such that

$$f(n) \leq c \times g(n), \quad \forall n \geq n_0.$$

- Too abstract? See the illustration shown in the next page.

[2] See https://en.wikipedia.org/wiki/Big_O_notation. You can also check the other 4 symbols: o, Θ, Ω, and ω.



- Clearly, $g(n)$ is the asymptotic upper bound of $f(n)$.[3]
- In other words, big O implies the worst case of the algorithm.
- We then classify the algorithms in Big O sense.

[3] See https://en.wikipedia.org/wiki/Big_O_notation#Infinite_asymptotics.

Discussions (1/4)

- Assume that the algorithm takes $8n^2 - 3n + 4$ steps.
- When n becomes large enough, the leading term dominates the whole behavior of the polynomial.
- So we simply focus on the leading term.
- It is easy to find a constant, say $c = 9$, so that $9n^2 \geq 8n^2$ holds.
- We then conclude that

$$8n^2 - 3n + 4 = O(n^2).$$

- It could say that the algorithm runs in $O(n^2)$ time.

Discussions (2/4)

- It is clear that SUM runs in $O(n)$ time and TRIANGLE runs in $O(n^2)$ time. (Why?)
- As a thumb rule, k-level loops run in $O(n^k)$ time.
- Determine the time complexity for the loop shown below.

```
...
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= i; j++) {
                for (int k = 1; k <= 5; k++) {
                    // Loop body.
                }
            }
        }
        // This algorithm runs in O( ? ) time.
...
```

Discussions (3/4): Which Will You Choose?

Benchmark

Size	$O(n)$	$O(n^2)$	$O(n^3)$
1	c_1	c_2	c_3
10	$10c_1$	$100c_2$	$1000c_3$
100	$100c_1$	$10000c_2$	$1000000c_3$

- In theory, the smaller the order, the faster the algorithm.

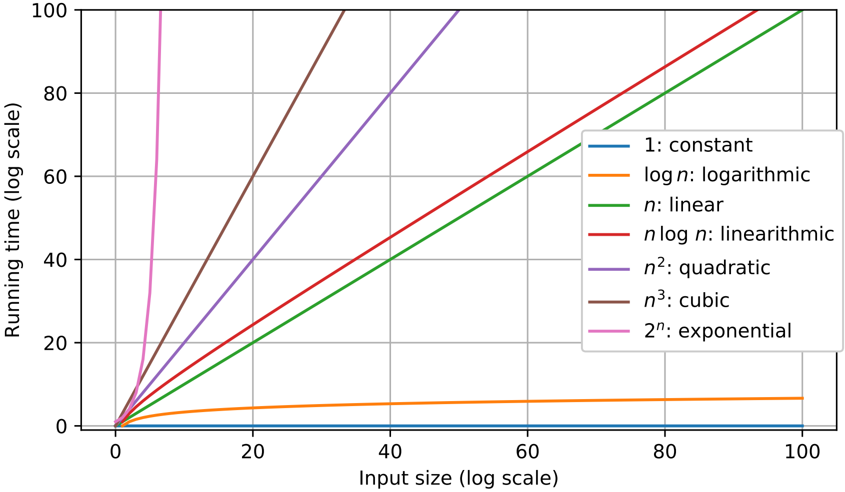
Discussions (4/4)

- It is worth to note that

$$8n^2 - 3n + 4 \neq O(n), \text{ and } 8n^2 - 3n + 4 = O(n^3). \text{ (Why?)}$$

- We would say that $8n^2 - 3n + 4 = O(n^2)$ for complexity analysis. (Why?)

Orders of Growth Rates



Big O Table

Growth order	Description	Example
$O(1)$	independent of n	x = y + z
$O(\log n)$	divide in half	binary search
$O(n)$	one loop	find maximum
$O(n \log n)$	divide and conquer	merge sort
$O(n^2)$	double loop	check all pairs
$O(n^3)$	triple loop	check all triples
$O(2^n)$	exhaustive search	check all subsets

Constant-Time Algorithms

- Basic instructions (e.g. $+$) run in $O(1)$ time. (Why?)
- Some algorithms indeed run in $O(1)$ time, for example, the arithmetic formulas. (Why?)
- However, there is no free lunch. (Why?)
- We should strike a balance by making a trade-off between generality and efficiency.
  - To reuse the program, it must be a general solution whose assumption should be little and weak.
  - To speed up the program, it could be optimized for the desire cases (so making assumptions).

- In addition, a program without writing explicit loops may not run in $O(1)$ time.
- For example, calling Arrays.sort() still takes more than $O(1)$ time to finish the sorting task.
- All in all, the time complexity is about the effort spent on the task but not how many time you sacrifice.

Exponential-Time Algorithms & Computability

- We, in fact, are overwhelmed by lots of intractable problems.
  - For example, the travelling salesman problem (TSP).[4]
  - Playing game well is hard.[5]
- Even worse, Turing (1936) proved the first undecidable (unsolvable) problem, called the halting problem.[6]
- You can find any textbook for theory of computation or computational complexity for further details.

[4] See https://en.wikipedia.org/wiki/Travelling_salesman_problem.
[5] See https://en.wikipedia.org/wiki/Game_complexity. Check out AlphaGo.
[6] See https://en.wikipedia.org/wiki/Halting_problem.

Logarithmic-Time Algorithms

- We have met one of logarithmic-time algorithms. (Which?)
- In conclusion, the log-time algorithms run much faster than the linear-time algorithms.
- However, the log-time algorithms require one assumption: ordered sequence.
- You will learn this kind of algorithms in any course about algorithms and data structures.

Outstanding Theoretical Problem[8]

$$\mathbb{P} \stackrel{?}{=} \mathbb{NP}$$

- In layman's term, $\mathbb{P}$ is the problem set of "being solved and verified in polynomial time."
- $\mathbb{NP}$ is the problem set of "being verified in polynomial time but perhaps being solved in exponential time."
  - For example, id verification is easier than hacking an account.
- One could say that $\mathbb{P}$ is easier than $\mathbb{NP}$.
- $\mathbb{P} \stackrel{?}{=} \mathbb{NP}$ asks if $\mathbb{NP}$ is solved by $\mathbb{P}$.
- It is still an open issue and also one of the Millennium Prize Problems.[7]

[7] See https://en.wikipedia.org/wiki/Millennium_Prize_Problems.

[8] See https://en.wikipedia.org/wiki/P_versus_NP_problem.