Big O Notation

Anson Ho

26 January 2019

Motivation

- Analyse program runtime in a better way

Motivation

- Analyse program runtime in a better way
- Learn a common language for discussion in OI

Time Complexity

- Description of runtime (or the number of operations)

Time Complexity

- Description of runtime (or the number of operations)
- Usually not a exact number
- But a function

Time Complexity

- Worst case

Time Complexity

- Worst case
- Average case (less common)
  - Random algorithm
  - Random test data

Space Complexity

- Description of memory usage

Big O Notation

- An usual representation for time/space complexity

Big O Notation

- An usual representation for time/space complexity
- An upper bound of a function

Big O Notation

- An usual representation for time/space complexity
- An upper bound of a function
- Constants and ”smaller” terms are usually ignored
  - Therefore, it is equivalent to measure runtime or to count the number of operations

Big O Notation

- An usual representation for time/space complexity
- An upper bound of a function
- Constants and ”smaller” terms are usually ignored
  - Therefore, it is equivalent to measure runtime or to count the number of operations
- e.g.: $6n^2 + 2n = \mathrm{O}(n^2)$

Definition

$f(n) = \mathrm{O}(g(n))$ if there exists $n_0, c > 0$ such that

$$f(n) \leq cg(n) \text{ for all } n \geq n_0$$

$f(n) = \Omega(g(n))$ if there exists $n_0, c > 0$ such that

$$f(n) \geq cg(n) \text{ for all } n \geq n_0$$

$f(n) = \Theta(g(n))$ if there exists $n_0, c_1, c_2 > 0$ such that

$$c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0$$

Convention

- Since Big O notation is about upper bound, $7(n+1)^2$ can be represented by O(n^n)

Convention

- Since Big O notation is about upper bound, $7(n+1)^2$ can be represented by O(n^n)
- We usually use the best (”smallest”) known upper bound

Convention

- Since Big O notation is about upper bound, $7(n+1)^2$ can be represented by $O(n^n)$
- We usually use the best (”smallest”) known upper bound
- Therefore, the first line is a bad example
- $O(n^2)$ is a better representation for that

Examples

- Linear search
  $O(n)$

Examples

- Linear search
  $O(n)$
- Bubble sort
  $O(n^2)$

Examples

- Linear search
  $O(n)$
- Bubble sort
  $O(n^2)$
- Binary search
  $O(\log n)$

Examples

- Linear search
  $O(n)$
- Bubble sort
  $O(n^2)$
- Binary search
  $O(\log n)$
- Merge sort
  $O(n \log n)$

Properties

- Addition

$$\begin{aligned} \mathrm{O}(f(n)) + \mathrm{O}(g(n)) &= \mathrm{O}(f(n) + g(n)) \\ &= \mathrm{O}(f(n)) \text{ if } f(n) \text{ is "greater" then } g(n) \end{aligned}$$

Properties

- Addition

$$\begin{aligned} \mathrm{O}(f(n)) + \mathrm{O}(g(n)) &= \mathrm{O}(f(n) + g(n)) \\ &= \mathrm{O}(f(n)) \text{ if } f(n) \text{ is "greater" then } g(n) \end{aligned}$$

- Scalar multiplication

$$\mathrm{O}(cf(n)) = \mathrm{O}(f(n)) \text{ if } c \text{ is a constant}$$

Multivariable

- $O(nmk)$
- $O(n^2m + nm^2)$

Constant

- Is $O(n)$ always better than $O(n \log n)$?

Constant

- Is $O(n)$ always better than $O(n \log n)$?
- No, since the constants are hidden

Constant

- Is $O(n)$ always better than $O(n \log n)$?
- No, since the constants are hidden
- Constant factor
  - Nature of the algorithm
  - Implementation
  - Machine performance

Constant

- Is $O(n)$ always better than $O(n \log n)$?
- No, since the constants are hidden
- Constant factor
  - Nature of the algorithm
  - Implementation
  - Machine performance
- The constant is usually dropped but not always fully dropped
  - `std::bitset`
  - $O(\frac{n}{32})$

Amortized Complexity

- Sometimes, the time complexity of a specific function is analysed instead of the whole program

Amortized Complexity

- Sometimes, the time complexity of a specific function is analysed instead of the whole program
- e.g.: precomputation, query, ...

Amortized Complexity

- Sometimes, the time complexity of a specific function is analysed instead of the whole program
- e.g.: precomputation, query, ...
- Amortized complexity: average performance of a function within a single run (cf. average case)

Amortized Complexity

- Sometimes, the time complexity of a specific function is analysed instead of the whole program
- e.g.: precomputation, query, ...
- Amortized complexity: average performance of a function within a single run (cf. average case)
- e.g.: `std::vector`, implementing queue by 2 stacks

Exercises