CSE 431/531: Analysis of Algorithms (Summer 2023)
Asymptotic Analysis

Chen Xu

June 1, 2023

Feedback from the mock quiz

- Most of you did the quiz! This is great! No worries this quiz is just a survey and it is not counted.
- Some of you have chosen the option of there is an algorithm that you don't know how to prove it is correct. Later you will have a chance to share it with the class! I have got one too! If any of you can give a counter example of my problem, I can treat you a meal.
- Some of the notations will appear in today's lecture.
- I am surprised to see that many people can do leetcode medium. Some part of the first project you will meet with tasks with similar difficulty.
- I am worried about the students who missed the first lecture. Still trying to get contact.

Tips

- Understand the problem statement and give the correct **instances**.
- Try the trivial solution first.
- Find the correlation between instances and fit the paradigms learned in class.

What is an instance?

The instance of a problem is basically the input.
For example, the instance of integer sorting problem can be a list of integers in any order.

Integer sorting

Instance: An array A of integers
Problem: What is the sorted array of A?

While the same instance can be different input for other problems:

Is-sorted problem

Instance: An array A of integers
Problem: Is A sorted?

Can you tell the difference between these two problems?

Instances come in many shapes and forms

Polynomial identity testing

Instance: A polynomial $p(x)$ of maximum degree d
Problem: Does $p(x) = 0$ for all x?

What is the instance? Is x the instance? No! Is d the instance? No!
x is a variable. d is determined by $p(x)$.
The instance is the $p(x)$ the polynomial itself. In one of the many ways, you can represent it using d coefficients since the maximum degree is d. By this representation the size of the instance is d.

Instances can be multi-dimensional

All-four-corner-one problem

Instance: A boolean matrix $A \in \{0,1\}^{m \times n}$
Problem: Does there exist a 2×2 submatrix in A such that all four corner elements are 1?

Example:

1 1 0 0 1
0 1 0 1 0
0 0 1 1 0
1 1 0 1 0

The input size is $m \times n$. It is two-dimensional.

What is Asymptotic Analysis?

The problems we mentioned above are all solvable. You can write algorithms to solve them.

Asymptotic Analysis is a method of describing the behavior of algorithms as the **input size** approaches infinity. It provides a high-level understanding of algorithm complexity, helping us compare algorithms in terms of their efficiency. There are mainly two aspects:

- Time asymptotic analysis – we would like to know how fast our algorithm executes on different inputs of size n.
  We usually denote this as $\mathbf{T(n)}$.
- Space asymptotic analysis – we would like to know how large the storage is in order to run the algorithm as the input size n grows.
  We usually denote this as $\mathbf{S(n)}$.

Five important notations

To describe the **growth rate** of the $T(n), S(n)$ of an algorithm. We have five notations:

- O-notation: Asymptotic upper bound.
- Ω-notation: Asymptotic lower bound.
- Θ-notation: Asymptotic tight bound.
- o-notation: Asymptotic upper bound that cannot be tight.
- ω-notation: Asymptotic lower bound that cannot be tight.

They are pronounced Big O, Big Omega, Theta, Little o, Little omega respectively.

O-notation definition

O-Notation For a function $g(n)$

$O(g(n)) = \{\text{function } f : \exists c > 0, n_0 > 0 \text{ such that } f(n) \leq cg(n), \forall n \geq n_0\}$

How do we read this definition:

- It is a set of functions with regard to $g(n)$ where $g(n)$ is also a function. Strictly, We write function $f(n) \in O(g(n))$. Conventionally, we may write $f(n) = O(g(n))$.
- The functions in this set all satisfy a condition, that is, you can always find a constant c (that is a constant no matter how large n is), and a n_0 threshold (we only study n bigger than this threshold), that for all n bigger than n_0, the $f(n)$ is smaller than $cg(n)$.
- c and n_0 come before n. Before you talk about n and the comparison of $f(n), cg(n)$, the c and n_0 are fixed already.

Example of O-notation proof

Let us study an example:

Does $f(n) = 3n^2 + 2n$ belong to $O(g(n))$,$g(n) = n^2 - 10n$?

Proof: Let $c = 4$ and $n_0 = 50$, for every $n > n_0 = 50$, we have,

$$\begin{aligned} f(n) - cg(n) &= 3n^2 + 2n - c(n^2 - 10n) \\ &= 3n^2 + 2n - 4(n^2 - 10n) \\ &= -n^2 + 42n \\ &\leq 0. \text{ when } n \geq 50 \\ f(n) &\leq 4g(n) \text{ when } n \geq 50 \end{aligned}$$

Therefore, $3n^2 + 2n \in O(n^2 - 10)$

Exercises

Exercise 1: Show that $3n^2 + 2n \in O(n^3 - 10)$
Exercise 2: Show that $n^{100} \in O(2^n)$

Wrong way of writing the notation

- As mentioned, although we can write $f(n) = O(g(n))$, writing $O(g(n)) = f(n)$ is wrong. The equal symbol is just a substitution of $\in$ symbol.
- We say a function belongs to a set of functions, but we don't say a set of function belongs to a function.

Ω-notation

Ω-notation is similar to O-notation except we flip the $\leq$ to $\geq$.

Ω-Notation For a function $g(n)$

$\Omega(g(n)) = \{\text{function } f : \exists c > 0, n_0 > 0 \text{ such that } f(n) \geq cg(n), \forall n \geq n_0\}$

Again, this is a set of functions.
But does this flip cause the set to flip?
In other words, does $f(n) \in O(g(n))$ imply $f(n) \notin \Omega(g(n))$? No!
Counter example: $3n^2 + 2n \in O(n^2 - 10n)$ and $3n^2 + 2n \in \Omega(n^2 - 10n)$.
They both hold. (Exercise!)

A true statement

Here is what is actually true:

Theorem

$f(n) \in O(g(n)) \Leftrightarrow g(n) \in \Omega(f(n))$

- Think about $a \leq b$ and $b \geq a$. They are equivalent.

Θ-notation

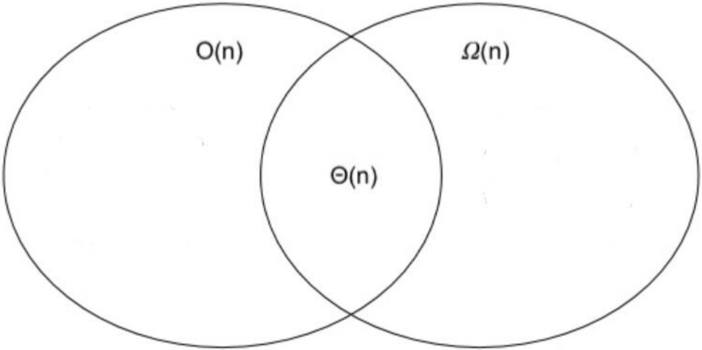
Θ-Notation For a function $g(n)$

$\Theta(g(n)) = \{\text{function } f : \exists c_1, c_2 > 0, n_0 > 0 \text{ such that } c_1 g(n) \leq f(n) \leq c_2 g(n), \forall n \geq n_0\}$

One way to show $f(n) \in \Theta(g(n))$ is to show

$f(n) \in O(g(n))$ **and** $f(n) \in \Omega(g(n))$

Relations



The usual way of using O,Ω,Θ notations

- The most common cases are when $f(n)$ are polynomials or exponential functions.
- For polynomials We usually keep the highest degree term and omit the leading coefficient.
- Indeed, for example, we can show that $an^2 + bn + c \in \Theta(n^2)$.
- We don't write $\Theta(an^2 + bn + c)$, although it is correct.

Tightness

- $3n^2 \in O(n^3)$ is true. $3n^2 \in O(n^2)$ is also true.
- But $3n^2 \notin O(n^{1.9999999999})$.
- We call the $O(n^2)$ a tight upper bound for $3n^2$.
- We can show that $\Theta(g(n))$ is included both in $O(g(n))$ and $\Omega(g(n))$

o-notation and ω-notation

- Recall that $O(g(n))$,$\Omega(g(n))$,$\Theta(g(n))$ are all sets of functions.
- $o(g(n))$ is simply the set which $O(g(n))$ excludes $\Theta(g(n))$.
- $\omega(g(n))$ is simply the set which $\Omega(g(n))$ excludes $\Theta(g(n))$.
- So $3n^2 \in O(n^2)$ but $3n^2 \notin o(n^2)$ because $3n^2 \in \Theta(n^2)$
- But $3n \in o(n^2)$.

Definition of o-notation and ω-notation

- The formal definition of o and ω notation are not so intuitive. It changes the $\exists c$ into $\forall c$, and it changes $\leq$ to $<$.

o-Notation For a function $g(n)$

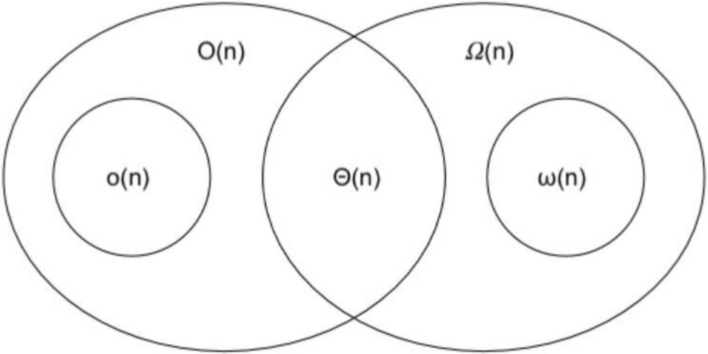
$o(g(n)) = \{\text{function } f : \forall c > 0, \exists n_0 > 0 \text{ such that } f(n) < cg(n), \forall n \geq n_0\}$

ω-Notation For a function $g(n)$

$\omega(g(n)) = \{\text{function } f : \forall c > 0, \exists n_0 > 0 \text{ such that } f(n) > cg(n), \forall n \geq n_0\}$

- $\exists$ means we only need to find one valid c to show the existence where $\forall$ means we need to enumerate all possibilities.
- Our previous proof of O notations does not work for o notations.

Relations including o and ω



Running time of common algorithms

- When talking about $T(n)$, we need to distinguish between $T_{best}(n)$, $T_{worst}(n)$, $T_{average}(n)$.
- Sometimes it matters a lot! The worst and the average are the two interesting ones.

Time complexity of some common algorithms

Algorithm	T_{avg}	T_{worst}
Binary Search	$\Theta(\log n)$	$O(\log n)$
Quick Sort	$\Theta(n \log n)$	$O(n^2)$
Insertion Sort	$\Theta(n^2)$	$O(n^2)$
Merge Sort	$\Theta(n \log n)$	$O(n \log n)$
Dijkstra	$\Theta(E + V \log V)$	$O(E + V \log V)$
Floyd Warshall	$\Theta(V^3)$	$O(V^3)$
Standard Matrix Multiplication	$\Theta(n^3)$	$O(n^3)$
Strassen's Mat. Mult.	$\Theta(n^{log_2 7})$	$O(n^{log_2 7})$
01-Knapsack	Unknown	$O(nW)$
Travelling Salesman	Unknown	$O(n!)$
Subset Sum	Unknown	$O(n2^n)$

Table: Time Complexity of Some Algorithms