CSC3100 Data Structures Tutorial 3

Jingjing Qian (PhD Student, SDS, 224040366@link.cuhk.edu.cn)

Acknowledgement: The tutorial materials are based on previous TA's works. Thank them for their contribution!

Contents

- Asymptotic Analysis
  - Concepts
  - Practice Problems

- Complexity for recursion and divide-and-conquer
  - Concepts
  - Practice Problems

1. Asymptotic Analysis

- Evaluate the “efficiency” of an algorithm.

- Commonly used notations:
  - Big-Oh notation: measure the upper bound complexity.
  - Big-Omega notation: measure the lower bound complexity.(Transition between Big-Oh and Big-Omega)
  - Big-Theta notation: Where Upper & lower bounds meet(Growth rate tight)

If meet

- For some algorithm $f(n)$, assume we can prove the greens:
- $O(1)$ $O(\log n)$ $O(n)$ $O(n \cdot \log n)$ $O(n^2)$ $O(n^3)$ $O(2^n)$

^^^^^^^^

(minimum upper bound)

- $\Omega(1)$ $\Omega(\log n)$ $\Omega(n)$ $\Omega(n \cdot \log n)$ $\Omega(n^2)$ $\Omega(n^3)$ $\Omega(2^n)$

^^^^^^^^

(maximum lower bound)

- The minimum upper bound and maximum lower bound meet
- => $f(n)$ is $\Theta(n \cdot \log n)$

If can't meet

- For some algorithm f(n), assume we can prove the greens:
- $O(1)$ $O(\log n)$ $O(n)$ $O(n \cdot \log n)$ $O(n^2)$ $O(n^3)$ $O(2^n)$

  ^^^^^

  (minimum upper bound)

- $\Omega(1)$ $\Omega(\log n)$ $\Omega(n)$ $\Omega(n \cdot \log n)$ $\Omega(n^2)$ $\Omega(n^3)$ $\Omega(2^n)$

  ^^^^^^^^^

  (maximum lower bound)

- The minimum upper bound and maximum lower bound don't meet
- =>Usually, **just use the upper bound**. We will say f(n) is $O(n^2)$

Some nice property for calculation

For Big-Oh and Big-Omega, we have:

1. Polynomial Rule: Only the biggest matter
2. Product Rule: the big multiplies the big
3. Sum Rule: the bigger of the two big
4. (Log Rule): Log only beats constant
5. (Exponential Rule): Exponential beats power functions

Practice Problems(1)

```
for (int i = 0; i < n; i++) {
    // Some O(1) operation
}
```

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        // Some O(1) operation
    }
}
```

Practice Problems(1)

```
for (int i = 0; i < n; i++) {
    // Some O(1) operation
}
```

$O(n)$

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        // Some O(1) operation
    }
}
```

$O(n^2)$

Practice Problems(2)

```
for (int i = 0; i < n; i+=(n/2)) {
    // Some O(1) operation
}
```

```
while (n > 0) {
    if (n % 2 == 1)
        res = res * a;
    a = a * a;
    n = n / 2;
}
```

Practice Problems(2)

```
for (int i = 0; i < n; i+=(n/2)) {
    // Some O(1) operation
}
```

$O(1)$

```
while (n > 0) {
    if (n % 2 == 1)
        res = res * a;
    a = a * a;
    n = n / 2;
}
```

This algorithm is called “Quick Power”

$O(log\ n)$

More Info: https://www.rookieslab.com/posts/fast-power-algorithm-exponentiation-by-squaring-cpp-python-implementation

Practice Problems(3)

check and prove $g(n) = (0.1n^2 + n\log n) \cdot (n\log n + \sqrt{n}) = \Theta(n^3 \cdot \log n)$.

Practice Problems(3)

check and prove $g(n) = (0.1n^2 + n\log n) \cdot (n\log n + \sqrt{n}) = \Theta(n^3 \cdot \log n)$.

Idea：

1. "Product rule" apply for both Big-Oh and Big-Omega
2. Apply "Product Property": 1^{st} term's : 0.1n^2 2^{nd} term's : nlogn
3. 0.1n^2 is both 1^{st} term's Big-Oh and Big-Omega, so is nlogn
4. Thus, the product of them is also both Big-Oh and Big-Omega → Big-Theta!

Two key points:

1. Apply the rule to simplify the problem;
2. When proving Θ(·), we need to prove O(·) and Ω(·) together.

2. Complexity analysis for recursion and divide-and-conquer

- To calculate the complexity for recursion and divide-and-conquer algorithm:
- Step 1: Get the recursive expression (looks like: $g(n) = g(n-1) + O(n)$, $g(n) = g(n/2) + O(n)$)
- Step 2:
  - Method 1: Unfold $g(n)$ to $g(1)$ by hand and get the answer.
  - Method 2: Master theorem

- Recurrence: $T(n) \leq a \cdot T(n/b) + O(n^d)$
- An algorithm that divides a problem of size n into a subproblems, each of size n / b

$$T(n) = \begin{cases} O(n^d \log n) & \text{if } a = b^d \\ O(n^d) & \text{if } a < b^d \\ O(n^{\log_b(a)}) & \text{if } a > b^d \end{cases}$$

a: number of subproblems (branching factor)
b: factor by which input size shrinks (shrinking factor)
d: need to do $O(n^d)$ work to create subproblems + "merge" their solutions

From Prof. Fang's slides – lec 6 – page 19.

4-1 *Recurrence examples*

Give asymptotic upper and lower bounds for $T(n)$ in each of the following recurrences. Assume that $T(n)$ is constant for $n \le 2$. Make your bounds as tight as possible, and justify your answers.

Let you be familiar with it!

a. $T(n) = 2T(n/2) + n^4$.

b. $T(n) = T(7n/10) + n$.

c. $T(n) = 16T(n/4) + n^2$.

d. $T(n) = 7T(n/3) + n^2$.

e. $T(n) = 7T(n/2) + n^2$.

f. $T(n) = 2T(n/4) + \sqrt{n}$.

g. $T(n) = T(n-2) + n^2$.

- Recurrence: $T(n) \le a \cdot T(n/b) + O(n^d)$
- An algorithm that divides a problem of size n into a subproblems, each of size n / b

$$T(n) = \begin{cases} O(n^d \log n) & \text{if } a = b^d \\ O(n^d) & \text{if } a < b^d \\ O(n^{\log_b(a)}) & \text{if } a > b^d \end{cases}$$

a: number of subproblems (branching factor)
b: factor by which input size shrinks (shrinking factor)
d: need to do $O(n^d)$ work to create subproblems + "merge" their solutions

Problem 4-1 (page 107, 3rd edition)

Question a-f: Use Master's Theorem.

a. By master theorem, $T(n) = \Theta(n^4)$.

b. By master theorem, $T(n) = \Theta(n)$.

c. By master theorem, $T(n) = \Theta(n^2 \lg n)$.

d. By master theorem, $T(n) = \Theta(n^2)$.

e. By master theorem, $T(n) = \Theta(n^{\lg 7})$.

f. By master theorem, $T(n) = \Theta(\sqrt{n} \lg n)$.

Question g: Expand the recursion

- $T(n) = T(n-2) + n^2$
- $\quad = T(n-4) + (n-2)^2 + n^2$
- $\quad = T(1) + 3^2 + \ldots\ldots + (n-4)^2 + (n-2)^2 + n^2$ (If n is odd)
- $\quad = T(2) + 4^2 + \ldots\ldots + (n-4)^2 + (n-2)^2 + n^2$ (If n is even)

- By the sum of the squares formula, We know that T(n) is $\Theta(n^3)$.
- Some small tricks in our case:
  - Even n: use formula with n=2k in it.
  - Odd n: use sum difference (total sum-even sum)

$1^2+2^2+3^2+4^2+\ldots+n^2$

Derivation:

$$\frac{n(n+1)(2n+1)}{6}$$

Question g: Expand the recursion (details)

- Tricks to calculate:
- \- Sum of even squares
- \- Sum of odd squares

$$T(n) = 2^2 + 4^2 + \cdots\cdots + n^2$$

let $n = 2k$:

$$T(n) = (2\times1)^2 + (2\times2)^2 + \cdots\cdots + (2k)^2$$

$$= 2^2(1^2 + 2^2 + \cdots\cdots + k^2)$$

$$= 4 \cdot \frac{k(k+1)(2k+1)}{6}$$

$$= \frac{n(n+2)(n+1)}{6} = \Theta(n^3)$$

$$T(n) = 1^2 + 3^2 + \cdots\cdots + n^2$$

$$= \underbrace{(1^2 + 2^2 + \cdots + n^2)}_{\text{known}} - \underbrace{(2^2 + 4^2 + \cdots + (n-1)^2)}_{\text{known above!}}$$

Look back: An example that upper/lower bounds does not meet (Will not be in exam)

If can't meet

- For some algorithm f(n), assume we can prove the greens:
- $O(1)$ $O(\log n)$ $O(n)$ $O(n \cdot \log n)$ $O(n^2)$ $O(n^3)$ $O(2^n)$

^^^^^

(minimum upper bound)

- $\Omega(1)$ $\Omega(\log n)$ $\Omega(n)$ $\Omega(n \cdot \log n)$ $\Omega(n^2)$ $\Omega(n^3)$ $\Omega(2^n)$

^^^^^^^^^

(maximum lower bound)

- The minimum upper bound and maximum lower bound don't meet
- =>Usually, **just use the upper bound**. We will say f(n) is $O(n^2)$

For expression like:

$T(n) = T(7n/10) + \log(n)$

Cannot use Master Theorem.

--But we can do scaling (放缩)

--So we can find a good upper bound, $O(n)$;

--And a good lower bound, $\Omega(\log n)$!

(Will not be tested. Just for fun!)

In Exercise 4-1 (b), we know:

$$T(n) = T\left(\frac{7n}{10}\right) + n \rightarrow O(n)$$

Question

But, if expression is

$$T(n) = T\left(\frac{7n}{10}\right) + \log(n)$$ ← We can't use Master Theorem

① $T(n) = T\left(\frac{7n}{10}\right) + \log(n) < T\left(\frac{7n}{10}\right) + n \rightarrow O(n)$

upper bound of this

also upper bound of original

By Master Theorem

② $T(n) = T\left(\frac{7n}{10}\right) + \log(n) > T\left(\frac{7n}{10}\right) + n^0 \rightarrow \Omega(\log n)$

lower bound of this

also lower bound of original

Conclusion:

By our knowledge so far, we can only conclude

$$T(n) = T\left(\frac{7n}{10}\right) + \log(n) = O(n), \Omega(\log n)$$

Thank you for coming!