Time Complexity of Recursive Algorithms

Mohammed Yaseen Mowzer

April 11, 2014

Definitions

The asymptotic tight bound

$$\Theta(g(n)) = f(n) \iff \exists c_1 \exists c_2 \forall n \geq n_0 \text{ s.t. } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$$

The asymptotic upper bound

$$O(g(n)) = f(n) \iff \exists c_1 \forall n \geq n_0 \text{ s.t. } 0 \leq c_1 g(n) \leq f(n)$$

The asymptotic lower bound

$$\Omega(g(n)) = f(n) \iff \exists c_1 \forall n \geq n_0 \text{ s.t. } 0 \leq f(n) \leq c_1 g(n)$$

Mergesort

Divide Divide the n-element sequence to be sorted into two subsequences of $n/2$ elements each.

Conquer Sort the two sequences recursively using mergesort.

Combine Merge the two sorted sequences to produce the answer.

The General Recurrence Relation

- We can define the time complexity in terms of a recurrence relation.
- If we conquer a problems of size n/b
- If D(n) is the time taken to divide the problem
- If C(n) is the time take to combine the problem

$$T(n) = aT(n/b) + D(n) + C(n)$$

Recurrence for Mergesort

- We conquer 2 problems of size $n/2$
- The time it takes to divide the sequence is constant $\Theta(1)$
- The time it takes to combine the problem is $\Theta(n)$

$$
\begin{aligned}
T(n) &= 2T(n/2) + \Theta(1) + \Theta(n) \\
T(n) &= 2T(n/2) + \Theta(n)
\end{aligned}
$$

because a linear function plus a constant function is a linear function.

Methods for Finding the Time Complexity

Recursion-tree method Convert the recurrence into a tree whose nodes represent the costs incurred at various levels of the recursion.

Subsitution method Guess a bound then prove it with mathematical induction.

Master method provides bounds for any recurrence of the form

$$T(n) = aT(n/b) + f(n)$$

Substitution method

$$
\begin{aligned}
T(n) &= 2T(n/2) + \Theta(n) \\
T(n) &= 2T(n/2) + cn \\
T(n) &= 2T(n/2) + n \\
T(n) &= n + 2(n/2 + 2T(n/4)) \\
T(n) &= n + n + 4(n/4 + 2T(n/8)) \\
T(n) &= n + n + n + 8(n/8 + 2T(n/16))
\end{aligned}
$$

We can guess that $T(n) = O(n\log(n))$

Induction

Assume $T(\lfloor n/2 \rfloor) = O(n \log n)$

$$
\begin{array}{rcl}
T(n) & \leq & 2T(\lfloor n/2 \rfloor) + n \\
T(n) & = & 2O(\lfloor n/2 \rfloor \log \lfloor n/2 \rfloor) + n \\
T(n) & = & 2c\lfloor n/2 \rfloor \log \lfloor n/2 \rfloor + n \\
T(n) & \leq & cn \log n/2 + n \\
T(n) & = & cn \log n - cn \log 2 + n \\
T(n) & = & cn \log n - cn + n \\
T(n) & \leq & cn \log n \\
T(n) & = & O(n \log n)
\end{array}
$$

Induction must be Precise

Suppose you had the recurrence

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$$

We can guess that the time complexity is $T(n) = O(n)$. We are required to show that $T(n) \leq cn$ for some c.

$$\begin{aligned} T(n) &\leq c\lfloor n/2 \rfloor + c\lceil n/2 \rceil + 1 \\ T(n) &= cn + 1 \end{aligned}$$

This does not satisfy the induction. We cannot drop the $+1$ in this case.

Assume Stronger Induction Hypothesis

Assume $T(k) \leq ck - d$ for some k. We now need to prove $T(n) \leq cn - d$

$$
\begin{aligned}
T(n) &\leq (c\lfloor n/2 \rfloor - d) + (c\lceil n/2 \rceil - d) + 1 \\
&= cn = 2d + 1 \\
&\leq cn - d
\end{aligned}
$$

Which is what we were required to prove.

The Master Method

The master method can be used to solve all recurrences of the form

$$T(n) = aT(n/b) + f(n)$$

where $a \geq 1$ and $b > 1$.
The master method consists of three cases.

The Master Theorem

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n)$$

where n/b is either $\lfloor n/b \rfloor$ and $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b(a)-\varepsilon})$ for $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$
2. If $f(n) = \Theta(n^{\log_n(a)})$ then $T(n) = \Theta(n^{\log_b a} \log_2 n)$
3. If $f(n) = \Omega(n^{\log_b(a)+\varepsilon})$ for $\varepsilon > 0$ and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$

Limitations on the master theorem

- In case 1 $f(n)$ is not just asymptotically smaller it is polynomially smaller than $n^{\log_b a}$
- In case 3 $f(n)$ is not just asymptotically larger it is polynomially larger than $n^{\log_b a}$
- The master theorem fails when $f(n)$ is smaller than $n^{\log_b a}$ but not polynomially smaller.
- The master theorem fails when $f(n)$ is larger than $n^{\log_b a}$ but not polynomially larger.
- The master theorem fails when the *regularity condition* for case 3, $af(n/b) < c(f(n))$, does not hold.

Using the master method

$$T(n) = 9\,T(n/3) + n$$

So $a = 9$, $b = 3$, $f(n) = n$
$O(n^{\log_b a}) = O(n^{\log_3 9}) = O(n^2)$. So $f(n) = O(n^{2-\varepsilon})$
This is case 1 of the master theorem. So $T(n) = \Theta(n^2)$

Using the master method

$$T(n) = T(2n/3) + 1$$

So $a = 1$, $b = 3/2$, $f(n) = 1$
$O(n^{\log_b a}) = O(n^{\log_{3/2} 1}) = O(n^0)$. So $f(n) = \Theta(1)$
This is case 2 of the master theorem. So $T(n) = \Theta(\log_2(n))$

Using the master method

$$T(n) = 3\,T(n/4) + n\log_2 n$$

So $a = 3$, $b = 4$, $f(n) = 1$
$O(n^{\log_b a}) = O(n^{\log_4 3}) \approx O(n^0.793)$. So $f(n) = \Omega(n^{0.793+\varepsilon})$
$af(n/b) = 3(n/4)log_2(n/4) \leq (3/4)n\log_2(n) = cf(n)$ for $c = 3/4$
This is case 3 of the master theorem. So $T(n) = \Theta(n\log_2(n))$

Using the master method

$$T(n) = 2\,T(n/2) + n\log_2 n$$

So $a = 2$, $b = 2$, $f(n) = n\log_2 n$
$O(n^{\log_b a}) = O(n)$
This satisfies none of the conditions, since f(n) is asymptotically bigger than n so it will not satisfy case 1 or 2 but it is not polynomially bigger so it will not satisfy case 3. So the master method does not apply.

Bibliography

Thomas H. Corman, *Introduction to alogorithms*, Ch. 3, 4