Data Structures – LECTURE 3

Recurrence equations

- Formulating recurrence equations
- Solving recurrence equations
- The master theorem (simple and extended versions)
- Examples: Merge-Sort and Quick-Sort

Complexity analysis of an algorithm

Two main methods:

- Direct counting: sum of the individual steps times the number of times executed $T(n) = \Sigma\, c_i t_i$

  Best for repeated iterations (loops).
- Recurrence equation: an equality or inequality describing the function in terms of its behavior on smaller inputs: $T(n) = T(n-1) + c$; $T(1) = 1$.

  → the solution of the equation is $T(n) = O(n^2)$.

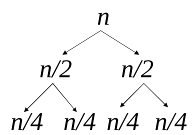
  Best for recursive functions and structures.

Recurrence equations

- Simplifying assumptions
  - n is sufficiently large
  - $T(1) = \Theta(1)$ for sufficiently small n. A value changes the solution of the equation, but usually only by a constant factor, so the order of growth is unchanged
  - Choose n according to boundary conditions: n is even ($n=2k$), a power of two ($n=2^k$) where $k>0$ is an integer
- Formulation: be very careful with the constants! $T(n)$ is not the same as $T(n/2)$!

Formulating recurrence equations

- Consider
  - in how many sub-problems the problem is split
  - what is the size of each sub-problem
  - how much work is required to combine the results of each sub-problem
- Recursion tree



Common recurrence equations (1)

- Factorial: multiply n by $(n-1)!$

  $T(n) = T(n-1) + O(1)$ → $O(n)$
- Fibonacci: add $fibonacci(n-1)$ and $fibonacci(n-2)$

  $T(n) = T(n-1) + T(n-2)$ → $O(2^n)$
- Sequential search: see if the first element is the one we are looking for, and if not, recursively call with one element less:

  $T(n) = T(n-1) + O(1)$ → $O(n)$
- Insertion sort: find the place of the first element in the sorted list, and recursively call with one element less:

  $T(n) = T(n-1) + O(n)$ → $O(n^2)$

Common recurrence equations (2)

- Binary search: see if the root of the tree is the one we are looking for, and if not, recursively call with either the left or right subtree, which has half the elements

  $T(n) = T(n/2) + O(1)$ → $O(\lg n)$
- Binary tree traversal: visit all the nodes of a tree by recursively visiting the nodes of the left and right tree:

  $T(n) = 2T(n/2) + O(1)$ → $O(n)$
- Merge Sort: split the list into two equal-sized parts, recursively sort each, and merge the resulting lists:

  $T(n) = 2T(n/2) + O(n)$ → $O(n \lg n)$

Solving recurrence equations

- Substitution: guess a bound and use mathematical induction to prove the guess correct.
- Recursion-tree: convert the recurrence into a tree whose nodes represent the costs at each level and use bounding summations to solve the recurrence.
- Master method: apply a theorem for recurrences of the form $T(n) = aT(n/b) + f(n)$
  where a, b are constants and $f(n)$ is a function.

The substitution method

The solution to the equation $T(n) = 2T(n/2) + n$ is $O(n \lg n)$ for $n \geq 2$; assume $T(1) = 1$

Prove: $T(n) \leq c(n \lg n)$ for $c \geq 2$

Base case: $T(2) \leq c\, 2\lg 2$, which holds for $c \geq 2$ since $T(2) = 3$

General case:

Assume that it holds for $n/2$, that is: $T(n/2) \leq 2(cn/2 \lg (n/2))$

Substitute into the recurrence relation and prove for n:

$$\begin{aligned} T(n) &\leq 2(cn/2 \lg (n/2) + n \\ &\leq cn \lg (n/2) + n \\ &\leq cn \lg n - cn \lg 2 + n \\ &\leq cn \lg n - cn + n \\ &\leq cn \lg n \quad \text{for } c \geq 1 \end{aligned}$$

Finding patterns in recurrences (1)

Write several elements of the recursion, and see if you can find a pattern. Once you find the pattern, prove it is true by substitution (induction)

$T(n) = T(n-1) + n$

$T(n-1) = T(n-2) + (n-1)$

$T(n-2) = T(n-3) + (n-2)$

$T(n-3) = T(n-4) + (n-3)$

Now substitute:

$$\begin{aligned} T(n) &= T(n-1) + n \\ &= [T(n-2) + (n-1)] + n \\ &= [[T(n-3) + (n-2)] + (n-1)] + n \\ &= [[[T(n-4) + (n-3)] + (n-2)] + (n-1)] + n \\ &= T(n-k) + \textstyle\sum_{i=1}^{k}(n-i+1) = T(n-k) + nk - ((k-1)k)/2 \end{aligned}$$

Finding patterns in recurrences (2)

$T(n) = T(n-k) + nk - ((k-1)k)/2$

At the end of the recursion, $k = n - 1$ and $T(1) = 1$, so we get:

$$\begin{aligned} T(n) &= 1 + n^2 - n + n^2/2 - 3n/2 - 1 \\ &= n^2/2 - n/2 \\ &= O(n^2) \end{aligned}$$

So the guess is that $O(n^2)$ is the solution to the recurrence $T(n) = T(n-1) + n$

The master theorem (simple version)

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

$$T(n) = aT(n/b) + n^c \qquad \text{where } c \geq 0$$

Then

1. $T(n) = \Theta(n^c)$ when $a/b^c < 1$ ($\log_b a < c$)
2. $T(n) = \Theta(n^c \log_b n)$ when $a/b^c = 1$ ($\log_b a = c$)
3. $T(n) = \Theta(n^{\log_b a})$ when $a/b^c > 1$ ($\log_b a > c$)

Recurrence equation of master theorem

$T(n) = aT(n/b) + n^c$

$T(n/b) = aT(n/b^2) + (n/b)^c$

$T(n/b^2) = aT(n/b^3) + (n/b^2)^c$

$T(n/b^4) = aT(n/b^5) + (n/b^4)^c$

Now substitute:

$$\begin{aligned} T(n) &= aT(n/b) + n^c \\ &= a[aT(n/b^2) + (n/b)^c] + n^c \\ &= a[a[aT(n/b^3) + (n/b^2)^c] + (n/b)^c] + n^c \\ &= a^kT(n/b^k) + n^c[1 + a(1/b)^c + a^2(1/b^2)^c + \ldots a^{k-1}(1/b^{k-1})^c] \\ &= a^kT(n/b^k) + \sum_{i=0}^{k-1} a^i \left(\frac{n}{b^i}\right)^c \end{aligned}$$

$k = \log_b n$, is the depth of the recursion

Recursion-tree for the equation

i=0 $\quad n \quad a \quad a\left(\frac{n}{b}\right)^c$

i=1 $\quad \left(\frac{n}{b}\right)^c \left(\frac{n}{b}\right)^c \left(\frac{n}{b}\right)^c \cdots \left(\frac{n}{b}\right)^c \quad a$

i=2 $\quad a \quad \left(\frac{n}{b^2}\right)^c \cdots \left(\frac{n}{b^2}\right)^c \quad \cdots \quad a \quad \left(\frac{n}{b^2}\right)^c \cdots \left(\frac{n}{b^2}\right)^c \quad a^2 \quad a^2\left(\frac{n}{b^2}\right)^c$

... $\quad \cdots \quad \cdots \quad \cdots$

i=k $\quad \left(\frac{n}{b^k}\right)^c \left(\frac{n}{b^k}\right)^c \left(\frac{n}{b^k}\right)^c \cdots \left(\frac{n}{b^k}\right)^c \cdots \left(\frac{n}{b^k}\right)^c \quad a^k \quad \Theta(n^{\log_b a})$

k= log$_b$*n, the depth of the recursion*

$$\frac{n}{b^{\log_b n}} = \Theta(1) \qquad \Theta(n^{\log_b a}) + \sum_{i=0}^{\log_b n-1} a^i \left(\frac{n}{b^i}\right)^c$$

Useful math to know…

Logarithms

- $\log_c(ab) = \log_c a + \log_c b$
- $\log_b a^n = n\log_b a$
- $\log_b a^n = n\log_b a$
- $\log_b(1/a) = -\log_b a$
- $\log_b a = 1 / \log_a b$
- $a = b^{\log_b a}$
- $a^{\log_b c} = c^{\log_b a}$

Geometric series: $\sum_{i=0}^{k} c^i = \frac{c^{k+1}-1}{c-1}$

See Appendix A in book for many useful tricks!

Master theorem proof

The number of comparisons is:

$$\Theta(n^{\log_b a}) + \sum_{i=0}^{\log_b n-1} a^i \left(\frac{n}{b^i}\right)^c =$$

$$\Theta(n^{\log_b a}) + n^c \sum_{i=0}^{\log_b n-1} \left(\frac{a}{b^c}\right)^i$$

which depends on the value of $\frac{a}{b^c}$

Master theorem: cases (1)

Case 1: $\frac{a}{b^c} < 1$ $\qquad n^c\left(\sum_{i=0}^{\log_b n-1} \left(\frac{a}{b^c}\right)^i\right)$

$$1 = \left(\frac{a}{b^c}\right)^0 < \sum_{i=0}^{\log_b n-1} \left(\frac{a}{b^c}\right)^i = \frac{1-\left(\frac{a}{b^c}\right)^{\log_b n}}{1-\left(\frac{a}{b^c}\right)} < \frac{1}{1-\left(\frac{a}{b^c}\right)} < const.$$

Therefore, $T(n) = \Theta(n^c)$

Master theorem: cases (2)

Case 2: $\frac{a}{b^c} = 1$ $\qquad n^c\left(\sum_{i=0}^{\log_b n-1} \left(\frac{a}{b^c}\right)^i\right)$

$$\sum_{i=0}^{\log_b n-1} \left(\frac{a}{b^c}\right)^i = \log_b n$$

Therefore, $T(n) = \Theta(n^c \log_b n)$

Master theorem: cases (3)

Case 3: $\frac{a}{b^c} > 1$ $\qquad n^c\left(\sum_{i=0}^{\log_b n-1} \left(\frac{a}{b^c}\right)^i\right)$

$$\sum_{i=0}^{\log_b n-1} \left(\frac{a}{b^c}\right)^i = \Theta\left(\left(\frac{a}{b^c}\right)^{\log_b n}\right)$$

$$n^c\Theta\left(\left(\frac{a}{b^c}\right)^{\log_b n}\right) = \Theta\left(n^c\left(\frac{a}{b^c}\right)^{\log_b n}\right)$$

$$n^c\left(\frac{a}{b^c}\right)^{\log_b n} = \frac{n^c}{b^{c\log_b n}}.a^{\log_b n} = \frac{n^c}{n^c}.n^{\log_b a} = n^{\log_b a}$$

Therefore, $T(n) = \Theta(n^{\log_b a})$

Example: Merge-Sort

The recurrence equation is:

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

Here, $a = 2$, $b = 2$, and $f(n) = n$ and $c = 1$
Case 2 applies

$$T(n) = \Theta(n^{\log_2 2} \lg n)$$

Conclusion:

$$T(n) = \Theta(n \lg n)$$

The master theorem (general version)

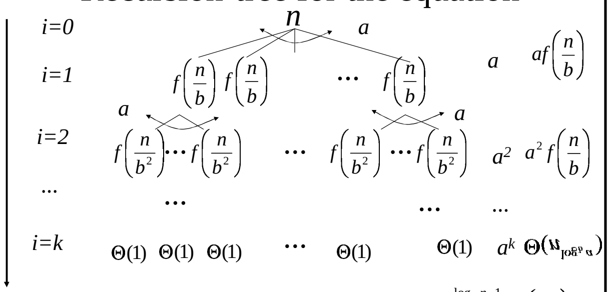
Let $a \geq 1, b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on non-negative integers by the recurrence:

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

where n/b is either $\lceil n/b \rceil$ or $\lfloor n/b \rfloor$

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

Recursion-tree for the equation



$k = \log_b n$, *the depth of the recursion*

$$\Theta(n^{\log_b a}) + \sum_{i=0}^{\log_b n - 1} a^i f\left(\frac{n}{b^i}\right)$$

The master theorem – intuition

Compares two terms: $O\left(n^{\log_b a}\right)$ and $f(n)$

1. when $O\left(n^{\log_b a}\right)$ dominates, the complexity is
$$T(n) = \Theta\left(n^{\log_b a}\right)$$
2. when $f(n)$ dominates, the complexity is
$$T(n) = \Theta\left(f(n)\right)$$
3. when they are comparable, there is a lg n penalty
$$T(n) = \Theta\left(n^{\log_b a} \lg n\right) = \Theta\left(f(n) \lg n\right)$$

see book for formal proof!

Simple example 1

Solve the recurrence

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

Here $a = 9$, $b = 3$, $f(n) = n$ and

$$n^{\log_b a} = n^{\log_3 9} = n^2$$

Case 1 applies: $\varepsilon = 1$

$$f(n) = O\left(n^{2-\varepsilon}\right)$$

Conclusion:

$$T(n) = \Theta\left(n^2\right)$$

Simple example 2

Solve the recurrence

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

Here $a = 1$, $b = 3/2$, $f(n) = 1$ and

$$n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$$

Case 2 applies:

$$f(n) = \Theta(1)$$

Conclusion:

$$T(n) = \Theta\left(n^0 \lg n\right) = \Theta(\lg n)$$

Simple example 3

Solve the recurrence

$$T(n) = 3T(n/4) + n\lg n$$

Here $a = 3$, $b = 4$, $f(n) = n \lg n$ and

$$n^{\log_b a} = n^{\log_4 3} = O\left(n^{0.793}\right)$$

Case 3 applies:

$$f(n) = \Omega\left(n^{0.792+\varepsilon}\right), \quad \varepsilon > 0$$

$$3(n/4)\lg(n/4) \le (3/4)n\lg n$$

Conclusion:

$$T(n) = \Theta(n\lg n)$$

Simple example 4

Solve the recurrence

$$T(n) = 2T(n/2) + n\lg n$$

Here $a = 2$, $b = 2$, $f(n) = n \lg n$ and

$$n^{\log_b a} = n^{\log_2 2} = O\left(n^1\right)$$

None of the three cases apply!

Case 3: $2(n/2) \lg (n/2) \le cn \lg n$ for c < 1 does not hold!

Conclusion: the master theorem cannot be used :-(

Recurrence equations to remember

- $T(n) = T(n-1) + O(1)$ → $O(n)$
- $T(n) = T(n-1) + O(n)$ → $O(n^2)$
- $T(n) = 2T(n-1) + O(1)$ → $O(2^n)$
- $T(n) = T(n/2) + O(1)$ → $O(\lg n)$
- $T(n) = 2T(n/2) + O(1)$ → $O(n)$
- $T(n) = 2T(n/2) + O(n)$ → $O(n \lg n)$