Chapter 3

Recursive Analysis - Recitation

3.1 Bounding recursive functions by hands.

Our main tool for dealing with recursive relations is the Master Theorem, which was proven in the lecture. In order to gain a better understanding, let's revisit the calculation in the proof for a specific case. Suppose that your algorithm analysis has yielded the following recursive relation:

Example 3.1.1. $T(n) = \begin{cases} 4T\left(\frac{n}{2}\right) + c \cdot n & \textit{for } n > 1 \\ 1 & \textit{else} \end{cases}$. *Thus, the running time is given by*

$$T(n) = 4T\left(\frac{n}{2}\right) + c \cdot n = 4 \cdot 4T\left(\frac{n}{4}\right) + 4c \cdot \frac{n}{2} + c \cdot n = \ldots =$$

$$\overbrace{4^h T(1)}^{\text{critical}} + c \cdot n\left(1 + \frac{4}{2} + \left(\frac{4}{2}\right)^2 \ldots + \left(\frac{4}{2}\right)^{h-1}\right) = 4^h + c \cdot n \cdot \frac{2^h - 1}{2 - 1}$$

We will call the number of iteration till the stopping condition the recursion height, and we will denote it by h . *What should be the recursion height?* $2^h = n \Rightarrow h = \log(n)$. *So in total we get that the algorithm running time equals* $\Theta\left(n^2\right)$.

Question, Why is the term $4^h T(1)$ so critical? Consider the case $T(n) = 4T\left(\frac{n}{2}\right) + c$ .One popular mistake is to forget the final term, which yields a linear solution $\Theta(n)$ (instead of quadric $\Theta(n^2)$).

Example 3.1.2. $T(n) = \begin{cases} 3T\left(\frac{n}{2}\right) + c \cdot n & \textit{for } n > 1 \\ 1 & \textit{else} \end{cases}$, *and then the expanding yields:*

$$T(n) = 3T\left(\frac{n}{2}\right) + c \cdot n = 3^2 T\left(\frac{n}{2^2}\right) + \frac{3}{2}cn + c \cdot n = \overbrace{3^h T(1)}^{\text{critical}} + cn\left(1 + \frac{3}{2} + \left(\frac{3}{2}\right)^2 + \ldots + \left(\frac{3}{2}\right)^{h-1}\right)$$

$$h = \log_2(n) \Rightarrow T(n) = 3^h T(1) + c \cdot n \cdot \left(\left(\frac{3}{2}\right)^{\log_2 n}\right) / \left(\frac{3}{2} - 1\right) = \theta\left(3^{\log_2(n)}\right) = \theta\left(n^{\log 3}\right)$$

where $n^{\log 3} \sim n^{1.58} < n^2$.

The Master Theorem is a crucial tool for dealing with recursive relations, as it was proven in our lecture. To gain a better understanding, let's revisit the calculation in the proof for a specific case. Suppose that your algorithm analysis has yielded the following recursive relation:

Example 3.1.3. $T(n) = \begin{cases} 4T\left(\frac{n}{2}\right) + c \cdot n & \textit{for } n > 1 \\ 1 & \textit{else} \end{cases}$. *This means that the running time is given by*

$$T(n) = 4T\left(\frac{n}{2}\right) + c \cdot n = 4 \cdot 4T\left(\frac{n}{4}\right) + 4c \cdot \frac{n}{2} + c \cdot n = \dots =$$
$$\overbrace{4^h T(1)}^{\textit{critical}} + c \cdot n \left(1 + \frac{4}{2} + \left(\frac{4}{2}\right)^2 \dots + \left(\frac{4}{2}\right)^{h-1}\right) = 4^h + c \cdot n \cdot \frac{2^h - 1}{2 - 1}$$

We define the number of iterations until the stopping condition is reached as the recursion height, denoted by h. To determine the recursion height, we can use the equation $2^h = n \Rightarrow h = \log(n)$. Therefore, the algorithm's running time is $\Theta\left(n^2\right)$.

Question: Why is the term $4^h T(1)$ so critical? Consider the case $T(n) = 4T\left(\frac{n}{2}\right) + c$ .One common mistake is to forget the final term, which results in a linear solution $\Theta(n)$ instead of a quadratic solution $\Theta(n^2)$.

Example 3.1.4. $T(n) = \begin{cases} 3T\left(\frac{n}{2}\right) + c \cdot n & \textit{for } n > 1 \\ 1 & \textit{else} \end{cases}$, *and expanding this yields:*

$$T(n) = 3T\left(\frac{n}{2}\right) + c \cdot n = 3^2 T\left(\frac{n}{2^2}\right) + \frac{3}{2}cn + c \cdot n = \overbrace{3^h T(1)}^{\textit{critical}} + cn\left(1 + \frac{3}{2} + \left(\frac{3}{2}\right)^2 + \dots + \left(\frac{3}{2}\right)^{h-}\right.$$
$$h = \log_2(n) \Rightarrow T(n) = 3^h T(1) + c \cdot n \cdot \left(\left(\frac{3}{2}\right)^{\log_2 n}\right) / \left(\frac{3}{2} - 1\right) = \theta\left(3^{\log_2(n)}\right) = \theta\left(n^{\log 3}\right)$$

where $n^{\log 3} \sim n^{1.58} < n^2$.

3.2 Master Theorem, one Theorem to bound them all.

As you might already notice, the same pattern has been used to bound both algorithms. The master theorem is the result of the recursive expansion. it classifies recursive functions at the form of $T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$, for positive function $f : \mathbb{N} \to \mathbb{R}^+$.

Master Theorem, simple version.

First, Consider the case that $f = n^c$. Let $a \geq 1, b > 1$ and $c \geq 0$. then:

1. if $\frac{a}{b^c} < 1$ then $T(n) = \Theta(n^c)$ **(f wins)**.
2. if $\frac{a}{b^c} = 1$ then $T(n) = \Theta(n^c \log_b(n))$.

3. if $\frac{a}{b^c} > 1$ then $T(n) = \Theta\left(n^{\log_b(a)}\right)$ **(f loose)**.

Example 3.2.1. *$T(n) = 4T\left(\frac{n}{2}\right) + d \cdot n \Rightarrow T(n) = \Theta\left(n^2\right)$ according to case (3). And $T(n) = 3T\left(\frac{n}{2}\right) + d \cdot n \Rightarrow T(n) = \Theta\left(n^{\log_2(3)}\right)$ also due to case (3).*

Master Theorem, strong version.

Now, let's generalize the simple version for arbitrary positive f and let $a \geq 1$, $b > 1$.

1. if $f(n) = O\left(n^{\log_b(a)-\varepsilon}\right)$ for some $\varepsilon > 0$ then $T(n) = \theta\left(n^{\log_b(a)}\right)$ **(f loose)**.
2. if $f(n) = \Theta\left(n^{\log_b(a)}\right)$ then $T(n) = \Theta\left(n^{\log_b(a)} \log(n)\right)$
3. if there exist $\varepsilon > 0, c < 1$ and $n_0 \in \mathbb{N}$ such that $f(n) = \Omega\left(n^{\log_b(a)+\varepsilon}\right)$ and for every $n > n_0$ $a \cdot f\left(\frac{n}{b}\right) \leq cf(n)$ then $T(n) = \theta(f(n))$ **(f wins)**.

Example 3.2.2. *1. $T(n) = T\left(\frac{2n}{3}\right) + 1 \to f(n) = 1 = \Theta\left(n^{\log_{\frac{3}{2}}(1)}\right)$ matches the second case. i.e $T(n) = \Theta\left(n^{\log_{\frac{3}{2}}(1)} \log n\right)$.*

2. $T(n) = 3T\left(\frac{n}{4}\right) + n \log n \to f(n) = \Omega\left(n^{\log_4(3)+\varepsilon}\right)$ and notice that $f\left(a\frac{n}{b}\right) = \frac{3n}{4} \log\left(\frac{3n}{4}\right)$. Thus, it's matching to the third case. $\Rightarrow T(n) = \Theta(n \log n)$.

3. $T(n) = 3T\left(n^{\frac{1}{3}}\right) + \log \log n$. let $m = \log n \Rightarrow T(n) = T(2^m) = 3T\left(2^{\frac{m}{3}}\right) + \log m$. denote by $S = S(m) = T(2^m) \to S(m) = 3T\left(2^{\frac{m}{3}}\right) + \log m = 3S\left(\frac{m}{3}\right) + \log m$. And by the fact that $\log m = O\left(m^{\log_3(3)-\varepsilon}\right) \to T(n) = T(2^m) = S(m) = \Theta(m) = \Theta(\log(n))$.

3.3 Recursive trees.

There are still cases which aren't treated by the *Master Theorem*. For example consider the function $T(n) = 2T\left(\frac{n}{2}\right) + n \log n$. Note, that $f = \Omega\left(n^{\log_b(a)}\right) = \Omega(n)$. Yet for every $\varepsilon > 0 \Rightarrow f = n \log n = O\left(n^{1+\varepsilon}\right)$ therefore the third case doesn't hold. How can such cases still be analyzed?

Recursive trees Recipe

1. draw the computation tree, and calculate it's height. in our case, $h = \log n$.
2. calculate the work which done over node at the k-th level, and the number of nodes in each level. in our case, there are 2^k nodes and over each we perform $f(n) = \frac{n}{2^k} \log\left(\frac{n}{2^k}\right)$ operations.
3. sum up the work of the k-th level.

4. finally, the total time is the summation over all the $k \in [h]$ levels.

applying the above, yields

$$T(n) = \sum_{k=1}^{\log n} n \cdot \log\left(\frac{n}{2^k}\right) = n \sum_{k=1}^{\log n} \left(\log n - \log 2^k\right) = n \sum_{k=1}^{\log n} (\log n - k) =$$
$$= \Theta\left(n \log^2(n)\right)$$

Example 3.3.1. *Consider merge sort variation such that instead of splitting the array into two equals parts it's split them into different size arrays. The first one contains $\frac{n}{10}$ elements while second contains the others $\frac{9n}{10}$ elements.*

Result: returns the sorted permutation of $x_1 \ldots x_n \in \mathbb{R}^n$

if $n \leq 10$ **then**
  return bubble-sort $(x_1 \ldots x_n)$
end

else
  define $S_l \leftarrow x_1 \ldots x_{\frac{n}{10}-2}, x_{\frac{n}{10}-1}$
  define $S_r \leftarrow x_{\frac{n}{10}}, x_{\frac{n}{10}+1} \ldots, x_n$

  $R_l \leftarrow$ non-equal-merge(S_l)
  $R_r \leftarrow$ non-equal-merge(S_r)

  return Merge(R_l, R_r)
end

Note, that the master theorem achieves an upper bound,

$$T(n) = n + T\left(\frac{n}{10}\right) + T\left(\frac{9n}{10}\right) \leq n + 2T\left(\frac{9n}{10}\right) \Rightarrow T(n) = O\left(n^{\log_{\frac{10}{9}}(2)}\right) \sim O\left(n^6\right)$$

Yet, that bound is far from been tight. Let's try to count the operations for each node. Let's try another direction.

Claim 3.3.1. *Let n_i be the size of the subset which is processed at the i-th node. Then for every k:*

$$\sum_{i \in k \text{ level}} n_i \leq n$$

Proof. Assuming otherwise implies that there exist index j such that x_j appear in at least two different nodes in the same level, denote them by u, v. As they both are in the same level, non of them can be ancestor of the other. denote by $m \in \mathbb{N}$ the input size of the sub array which is processed by the the lowest common ancestor of u and v, and by $j' \in [m]$ the position of x_j in that sub array. By the definition of the algorithm it steams that $j' < \frac{m}{10}$ and $j' \geq \frac{m}{10}$. contradiction. The height of the tree is bounded by $\log_{\frac{9}{10}}(n)$. Therefore the total work equals $\Theta(n \log n)$. Thus, the total running time equals to:

$$T(n) = \sum_{k \in \text{levels}} \sum_{i \in \text{k level}} f(n_i) = \sum_{k \in \text{levels}} \sum_{i \in \text{k level}} n_i \leq n \log n$$

□

3.4 Appendix. Recursive Functions In Computer Science. (Beyond the scope of the 2024 course.)

Example 3.4.1 (Leading Example. numbers multiplication.). *Let x, y be an n'th digits numbers over $\mathbb{F}_2^n$. It's known that summing such a pair requires a linear number of operations. Write an algorithm that calculates the multiplication $x \cdot y$.*

Example 3.4.2 (Long multiplication.). *To understand the real power of the dividing and conquer method, let's first examine the known solution from elementary school. In that technics, we calculate the power order and the value of the digit separately and sum up the results at the end. Formally: $x \leftarrow \sum_{i=0}^{n} x_i 2^i$ Thus,*

$$x \cdot y = \left(\sum_{i=0}^{n} x_i 2^i\right)\left(\sum_{i=0}^{n} y_i 2^i\right) = \sum_{i,j \in [n] \times [n]} x_i y_j 2^{i+j}$$

the above is a sum up over n^2 numbers, each at length n and therefore the total running time of the algorithm is bounded by $\theta(n^3)$. ***[COMMENT]*** *notice that adding 1 to $1111111111...1$ requires $O(n)$.*

Example 3.4.3 (Recursive Approach.). *We could split x into the pair x_l, x_r such that $x = x_l + 2^{\frac{n}{2}} x_r$. Then the multiplication of two n-long numbers will be reduced to sum up over multiplication of a quartet. Each at length $\frac{n}{2}$. Thus, the running time is given by*

$$\begin{aligned} x \cdot y = \left(x_l + 2^{\frac{n}{2}} x_r\right)\left(y_l + 2^{\frac{n}{2}} y_r\right) &= x_l y_l + 2^{\frac{n}{2}}(x_l y_r + x_r y_l) + 2^n x_r y_r \\ \Rightarrow T(n) = 4T\left(\frac{n}{2}\right) + c \cdot n &= 4 \cdot 4T\left(\frac{n}{4}\right) + 4c \cdot \frac{n}{2} + c \cdot n = \ldots = \\ c \cdot n \left(1 + \frac{4}{2} + \left(\frac{4}{2}\right)^2 \ldots + \left(\frac{4}{2}\right)^{h-1}\right) &+ 4^h T(1) = n^2 + c \cdot n \cdot \frac{2^h - 1}{2 - 1} \end{aligned}$$

We will call the number of iteration till the stopping condition the recursion height, and we will denote it by h . What should be the recursion height? $2^h = n \Rightarrow h = \log(n)$. So in total we get that multiplication could be achieved by performs $\Theta\left(n^2\right)$ operations.

Karatsuba algorithm. It was once thought that multiplication could not be done in less than $\Omega\left(n^2\right)$ time; however, Karatsuba discovered an algorithm [KO63] that proved this wrong. Let $z_0, z_1 z_3$ be defined as follows:

$$z_0, z_1, z_2 \leftarrow x_l y_r, x_l y_r + x_r y_l, x_r y_r$$

Result: returns the multiplication $x \cdot y$ where $x, y \in \mathbb{F}_2^n$

if $x, y \in \mathbb{F}_2$ **then**
    return $x \cdot y$
end

else
    define $x_l, x_r \leftarrow x$ and $y_l, y_r \leftarrow x$ // $O(n)$.

    calculate $z_0 \leftarrow \text{Karatsuba}(x_l, y_l)$
        $z_2 \leftarrow \text{Karatsuba}(x_r, y_r)$
        $z_1 \leftarrow \text{Karatsuba}(x_r + x_l, y_l + y_r) - z_0 - z_2$

    return $z_0 + 2^{\frac{n}{2}} z_1 + 2^n z_2$ // $O(n)$.
end

Notice that $z_1 = (x_l + x_r)(y_l + y_r) - z_0 - z_1$. summarize the above yields the following pseudo code.

Let's analyze the running time of the algorithm above, assume that $n = 2^m$ and then the recursive relation is

$$T(n) = 3T\left(\frac{n}{2}\right) + c \cdot n = 3^2 T\left(\frac{n}{2^2}\right) + \frac{3}{2}cn + c \cdot n = cn\left(1 + \frac{3}{2} + \left(\frac{3}{2}\right)^2 + \ldots + \left(\frac{3}{2}\right)^{h-1}\right) +) + 3$$

$$h = \log_2(n) \Rightarrow T(n) = n^{\log_2 3} + c \cdot n \cdot \left(\left(\frac{3}{2}\right)^{\log_2 n}\right) / \left(\frac{3}{2} - 1\right) = \theta\left(3^{\log_2(n)}\right) = \theta\left(n^{\log 3}\right)$$

where $n^{\log 3} \sim n^{1.58} < n^2$.

Bibliography

[KO63] Anatolij A. Karatsuba and Yu. Ofman. "Multiplication of Multidigit Numbers on Automata". In: *Soviet physics. Doklady* 7 (1963), pp. 595–596. URL: https://api.semanticscholar.org/CorpusID:117858583.