Algorithms & Data Structures
Part 2 (weeks 6–10)
Set 3: Recurrences

Konrad Dabrowski
konrad.dabrowski@durham.ac.uk

Online Office Hour:
Mondays 13:30–14:30 from Week 7
See Duo for the Zoom link

Table of Contents

Intro

We've seen two types of algorithms: **iterative** and **recursive**.

To analyse iterative algorithms,

- look at loop structure,
- identify relevant operations, and
- essentially, count them.

One often ends up with some sort of sum (the more nested loops, the more nested sums).

Intro

To analyse recursive algorithms, first of all note that most of them are actually hybrids between iterative and strictly recursive (e.g. the *Partition* funcion in *QuickSort* is iterative, the rest of *QuickSort* is recursive.

Anyway,

- look at **recursive structure**, e.g. *MergeSort*:
  - split input of size n into two halves of equal size $n/2$ each
  - independently recurse into each half
  - merge the resulting sorted sequences
- this will normally give you a **recurrence**, pretty much trivially:

$$T(n) \leq \begin{cases} d & \text{if } n \leq c\text{, for constants } c, d > 0 \\ \underbrace{2 \cdot T(n/2)}_{\text{recursions}} + \underbrace{z \cdot n}_{\text{merging}} & \text{otherwise} \end{cases}$$

Intro

Two time-tested methods for solving such recurrences:

1. **Induction** (a.k.a. guess, substitute and verify)
2. **Master Theorem**

If guessing in (1) doesn't help then maybe **iterative substitution** (a.k.a. spot the pattern)

Solving recurrences by induction

Quite simple, really. Only need to

- "guess" correct solution, and
- verify base case(s) and step

Former is art, latter is maths.

We'll normally not spend too much time on base case(s) as when we're talking algorithms, it's quite clear that constant size input requires some constant number of steps, full stop.

Interesting technical bit is step (recursion as well as induction).

Solving recurrences by induction

Consider again recurrence for *MergeSort*:

$$T(n) \leq \begin{cases} d & \text{if } n \leq c \text{, for constants } c, d > 0 \\ 2 \cdot T(n/2) + z \cdot n & \text{otherwise} \end{cases}$$

Experience/intuition tells me that this smells like it ought to be $T(n) = O(n \log_2 n)$

If you haven't got that yet, try starting big and consecutively reduce upper bound

- $O(n^4)$ works, try $O(n^3)$.
- Works? Try $O(n^2)$.
- Still works? Try $O(n)$.
- No luck? Back up to $O(n \log n)$.

Something like that. Experience will come with practice.

Solving recurrences by induction

Now that we've got some guess, we need to see if it's correct.

Recall, we guess that $T(n) \leq \alpha n \log_2 n$ for some constant $\alpha > 0$.

Spare a second for base case: recurrence said $T(n) \leq d$ if $n \leq c$, for constants c and d. Make α big enough

$$d \leq \alpha n \log_2 n \leq \alpha c \log_2 c \quad \Leftrightarrow \quad \alpha \geq \frac{d}{c \log_2 c}$$

Voila!

Solving recurrences by induction

As for the rest: simply plug it (our guess, $T(n) \leq \alpha n \log_2 n$) in!

$$
\begin{aligned}
T(n) &\leq 2T(n/2) + zn \\
T(n) &\leq 2\alpha \frac{n}{2} \log_2 \frac{n}{2} + zn \\
T(n) &\leq 2\alpha \frac{n}{2} (\log_2(n) - \log_2(2)) + zn \\
T(n) &\leq 2\alpha \frac{n}{2} (\log_2(n) - 1) + zn \\
T(n) &\leq \alpha n (\log_2(n) - 1) + zn \\
T(n) &\leq \alpha n \log_2(n) - \alpha n + zn \\
T(n) &\leq \alpha n \log_2 n \qquad \text{if } \alpha n \geq zn \;\Leftrightarrow\; \alpha \geq z
\end{aligned}
$$

Altogether, it works for any $\alpha \geq \max\left\{ \frac{d}{c \log_2 c}, z \right\}$.

Solving recurrences by induction

The requirement $\alpha \geq z$ suggests there isn't much room for (asymptotic) improvement, as both are constants

There would be more if we had overshot the target, e.g. if our guess had been $T(n) \leq \alpha n^2$.

Observe:

$$\begin{aligned}
T(n) &\leq 2T(n/2) + zn \\
T(n) &\leq 2\alpha(n/2)^2 + zn \\
T(n) &\leq 2\alpha n^2/4 + zn \\
T(n) &\leq \alpha n^2/2 + zn \\
T(n) &\leq \alpha n^2 - \alpha n^2/2 + zn
\end{aligned}$$

True whenever
$\alpha n^2/2 \geq zn \quad \Leftrightarrow \quad \alpha n/2 \geq z \quad \Leftrightarrow \quad \alpha \geq 2z/n = \Theta(1/n)$

This isn't really any restriction for constant α. If you see something like this, try again with smaller guess.

Solving recurrences by induction

On the other hand, suppose we're being greedy, and are guessing $T(n) \leq \alpha n$

$$
\begin{aligned}
T(n) &\leq 2T(n/2) + zn \\
T(n) &\leq 2\alpha n/2 + zn \\
T(n) &\leq \alpha n + zn
\end{aligned}
$$

This is $\leq \alpha n$ if and only if $zn \leq 0$, which it isn't going to be any time soon!

Solving recurrences by iterative substitution

Expand recurrence, spot the pattern, then treat as above

Consider again *MergeSort*'s recurrence

$$T(n) \leq \begin{cases} d & \text{if } n \leq c \text{, for constants } c, d > 0 \\ 2 \cdot T(n/2) + z \cdot n & \text{otherwise} \end{cases}$$

Expand:

$$\begin{aligned} T(n) &\leq 2T(n/2) + zn \\ &\leq 2(2T(n/4) + zn/2) + zn = 4T(n/4) + zn + zn \\ &= 4T(n/4) + 2zn \leq 4(2T(n/8) + zn/4) + 2zn \\ &= 8T(n/8) + zn + 2zn = 8T(n/8) + 3zn \\ &\leq 8(2T(n/16) + zn/8) + 3zn = 16T(n/16) + zn + 3zn \\ &= 16T(n/16) + 4zn \end{aligned}$$

Looks a lot like we can play this game $\log_2 n$ many times, and we'll find that after i many times

- first term: $2^i T(n/2^i)$
- second term: izn

Therefore, after $\log_2 n$ many times,
$2^{\log_2 n} \cdot$ "base case value" $+ \log_2(n)zn = O(n \log n)$.

Solving recurrences using the Master Theorem

Can use if recurrence is of form

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

for $a \geq 1$ and $b > 1$. e.g. *MergeSort*: $a = 2$, $b = 2$, $f(n) = zn$

Three cases:

1. **If** $f(n) = O(n^{\log_b(a)-\epsilon})$ for some constant $\epsilon > 0$ **then** $T(n) = \Theta(n^{\log_b(a)})$.
2. **If** $f(n) = \Theta(n^{\log_b(a)} \cdot \log^k n)$ with $k \geq 0$ **then** $T(n) = \Theta(n^{\log_b(a)} \log^{k+1} n)$. [1]
3. **If** $f(n) = \Omega(n^{\log_b(a)+\epsilon})$ for some constant $\epsilon > 0$ **and** if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all n large enough **then** $T(n) = \Theta(f(n))$

With *MergeSort*, $n^{\log_b(a)} = n^{\log_2(2)} = n$ and $f(n) = zn$, thus Case 2 applies (with $k = 0$), and we find that $T(n) = \Theta(n \log n)$.

[1] $\log^k n = (\log n)^k$

Solving recurrences: QuickSort

Recurrence looks like so:

$$T(n) \leq \begin{cases} d & \text{if } n \leq c \text{, constants } c, d > 0 \\ T(q) + T(n-q-1) + z \cdot n & \text{for some } 0 \leq q < n \end{cases}$$

Again, base case isn't going to give us any trouble.

However, can't use expansion, and can't use Master Theorem.

Not easily, anyway.

QuickSort, worst-case performance

Let $T_w(n)$ be worst-case running time of (our) QS on input of size n. Then,

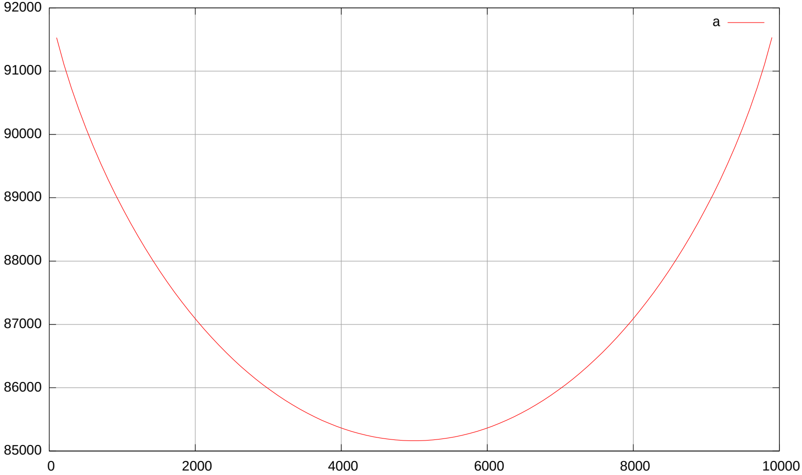
$$T_w(n) = \max_{0 \le q < n} \{T_w(q) + T_w(n-q-1)\} + zn$$

i.e. we **maximise** over all possible partitionings (q and $n-q-1$)

We guess $T_w(n) \le \alpha n^2$ for some constant $\alpha > 0$
Substitute guess into recurrence:

$$\begin{aligned}
T_w(n) &= \max_{0 \le q < n} \{T_w(q) + T_w(n-q-1)\} + zn \\
&\le \max_{0 \le q < n} \left\{\alpha q^2 + \alpha(n-q-1)^2\right\} + zn \\
&= \alpha \cdot \max_{0 \le q < n} \left\{q^2 + (n-q-1)^2\right\} + zn
\end{aligned}$$

Consider expression $q^2 + (n - q - 1)^2$ for $0 \leq q < n$. It's easy to see that the expression is maximal when $q = 0$ or $q = n - 1$



This implies (choosing $n-1$)

$$q^2 - (n-q-1)^2 \leq (n-1)^2 - (n-(n-1)-1)^2 = (n-1)^2$$

and therefore

$$\begin{aligned}
T_w(n) &\leq \alpha \cdot \max_{0 \leq q < n} \left\{ q^2 + (n-q-1)^2 \right\} + zn \\
&\leq \alpha \cdot (n-1)^2 + zn \\
&= \alpha \cdot (n^2 - 2n + 1) + zn \\
&\leq \alpha n^2 - 2\alpha n + \alpha + zn \\
&= \alpha n^2 - (2\alpha n - zn - \alpha) \\
&\leq \alpha n^2
\end{aligned}$$

for α large enough, that is, whenever
$2\alpha n - zn - \alpha \geq 0 \Leftrightarrow \alpha \geq \frac{zn}{2n-1} = \Theta(1)$

Thus, worst case running time of (our) QS is $O(n^2)$

Can run essentially the same argument for lower-bounding worst case, to get $\Theta(n^2)$

Table of Contents