Recurrences

1 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).

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:

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

Three methods for solving such recurrencesL

- Induction (aka guess substitute and verify)
- Iterative substitution (spot the pattern)
- Master theorem

2 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).

Consider again the recurrence for mergesort

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

This looks like it ought to be $T(n) = O(n \log_2 n)$. This is found by working backwards from a large big O until it no longer works

Now that we've got some guess, we need to see if it's correct.
Recall, we guess that $T(n) \leqslant \alpha n \log_2 n$ for some constant $\alpha > 0$

In the base case, the recurrence said $T(n) \leqslant d$ if $n \leqslant c$ for constants c and d. Make α big enough:

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

Because of the relation between n and c, you can swap n for c in these inequalities

As for the rest, plug our result in

$$\begin{aligned}
&T(n) \leq 2T(n/2) + an - \text{assumption}\\
&T(n) \leq 2\alpha \tfrac{n}{2} \log_2 \tfrac{n}{2} + an - \text{Replace the T part with the guess}\\
&T(n) \leq 2\alpha \tfrac{n}{2} \left(\log_2(n) - \log_2(2)\right) + an\\
&T(n) \leq 2\alpha \tfrac{n}{2} \left(\log_2(n) - 1\right) + an\\
&T(n) \leq \alpha n \left(\log_2(n) - 1\right) + an\\
&T(n) \leq \alpha n \log_2 n - \alpha n + an\\
&T(n) \leq \alpha n \log_2 n \quad \text{if } \alpha n \geq an \Leftrightarrow \alpha \geq a - \text{Make sure last two terms are negative}
\end{aligned}$$

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

2.1 What happens if you make a bad guess?

2.1.1 Guess too high

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

There would be more if we had overshot our target, for example if we had guessed $T(n) \leqslant \alpha n^2$
Observe:

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

True whenever

$$\alpha n^2/2 \geq an \Leftrightarrow \alpha n/2 \geq a \quad \Leftrightarrow \quad \alpha \geq 2a/n = \Theta(1/n)$$

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

2.1.2 Guess too low

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

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

This is $\leqslant \alpha n$ iff $an \leqslant 0$ which isn't going to be any time soon

3 Iterative substitution

Expand recurrence, spot the pattern, solve algebraically

Again for mergesort:

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

Expand:

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

This looks like we can do this $\log_2 n$ many times, and we'll find that after i many times

- First term: $2^i T(n/2^i)$
- Second term ian

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

4 Master Theorem

Can use if the recurrence is of the form:

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

for $a \geqslant 1$ and $b > 1$
Eg. Mergesort a=2, b=2, f(n)=an
Three cases:

1. If $f(n) = O\left(n^{\log_b(a)-\epsilon}\right)$ for some constant $\epsilon > 0$ then $T(n) = \Theta\left(n^{\log_b(a)}\right)$
2. If $f(n) = \Theta\left(n^{\log_b(a)}\right)$ then $T(n) = \Theta\left(n^{\log_b(a)} \log n\right)$
3. If $f(n) = \Omega\left(n^{\log_b(a)+\epsilon}\right)$ 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 $f(n) = an$, thus case 2 applies, and we find that $T(n) = \Theta(n \log n)$

Recurrence looks like so:

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

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

4.1 Quicksort, worst case performance

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

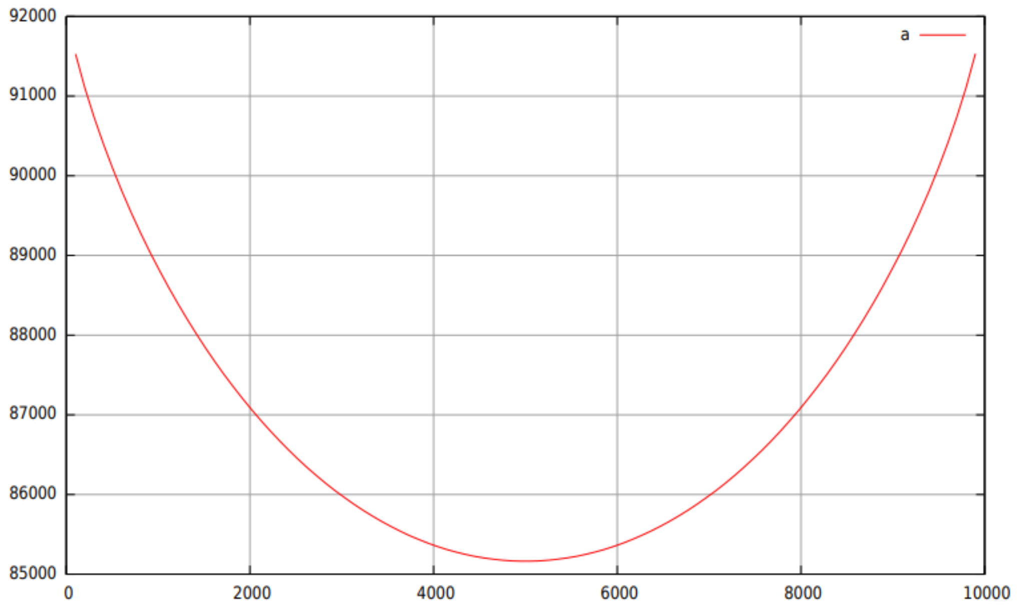
$$T_w(n) = \max_{0 \leq q < n} \{T_w(q) + T_w(n-q-1)\} + an$$

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

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

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

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


a
92000
91000
90000
89000
88000
87000
86000
85000
0
2000
4000
6000
8000
10000