Generating Functions and Linear Recurrences

Kieran Rimmer [*]

September 2020

Merge sort is a divide and conquer algortithm wherein a data structure to be sorted is recursively split in two until sorting is trivial, and then then sorted substructres are merged until the full structure is sorted.

For simplicity, let us consider only data which is sized in powers of two. Then the worst case running time

$$T_n$$

for a naive implementation of merge sort can be approximated with:

$$\begin{aligned} T_n &= c_1 + T_{n-1} + T_{n-1} + 2^n c_2 \\ &= c_1 + 2 \cdot T_{n-1} + 2^n c_2 \end{aligned} \tag{1}$$

This relation yeids a sequence, which has a corresponding *generating function*:

$$< T_0, T_1, T_2, T_3 \cdots > \leftrightarrow F(x) = T_0 + T_1 \cdot x + T_2 \cdot x^2 + T_3 \cdot x^3 \dots \tag{2}$$

The sequence can be broken down into a sum of sequences, for which convenient closed form summations exist *if convergence is assumed*:

$$\begin{array}{rl} < c_1, c_1, c_1, c_1 \cdots > & \leftrightarrow c_1/(1-x) \\ < c_2, 2 \cdot c_2, 4 \cdot c_2 \dots 2^n \cdot c_2 > & \leftrightarrow c_2/(1-2x) \\ + < 0, 2 \cdot T_0, 2 \cdot T_1 \dots 2 \cdot T_{n-1} > & \leftrightarrow 2xF(x) \\ \hline \end{array}$$

$$< T_0, T_1, T_2, T_3 \cdots > \leftrightarrow F(x) = c_1/(1-x) + c_2/(1-2x) + 2xF(x) \tag{3}$$

[*]just some Jimbo

Then:

$$\begin{aligned} F(x) &= c_1/(1-x) + c_2/(1-2x) + 2xF(x) \\ F(x)\cdot(1-2x) &= c_1/(1-x) + c_2/(1-2x) \\ F(x) &= \frac{c_1}{(1-x)(1-2x)} + \frac{c_2}{(1-2x)^2} \end{aligned} \tag{4}$$

Using the technique of partial fractions:

$$\begin{aligned} \frac{c_1}{(1-x)(1-2x)} &= \frac{\gamma_1}{(1-x)} + \frac{\gamma_2}{(1-2x)} \\ &= \frac{\gamma_1\cdot(1-2x) + \gamma_2\cdot(1-x)}{(1-x)(1-2x)} \\ &= \frac{\gamma_1 + \gamma_2 - (2\gamma_1 + \gamma_2)x}{(1-x)(1-2x)} \end{aligned} \tag{5}$$

Then:

$$\begin{aligned} \gamma_1 + \gamma_2 &= c_1 \\ 2\gamma_1 + \gamma_2 &= 0 \\ \gamma_1 &= -c_1 \\ \gamma_2 &= 2c_1 \\ \frac{c_1}{(1-x)(1-2x)} &= \frac{-c_1}{(1-x)} + \frac{2c_1}{(1-2x)} \end{aligned} \tag{6}$$

Theorem 1.1: if $a, b \in N$ and $b > a \geq 0$, then for any

$$n \geq a$$

the

$$nth$$

coefficient of

$$\frac{cx^a}{(1-\alpha x)^b}$$

is

$$\frac{c(n-a+b-1)!\alpha^{n-a}}{(n-a)!\cdot(b-1)!}$$

where

. Proof is given in course materials.

So:

$$
\begin{aligned}
F(x) =& \frac{-c_1}{(1-x)} + \frac{2c_1}{(1-2x)} + \frac{c_2}{(1-2x)^2} \\
T_n =& \frac{-c_1(n-0+1-1)!1^{n-0}}{(n-0)! \cdot (1-1)!} \\
&+ \frac{2c_1(n-0+1-1)!2^{n-0}}{(n-0)! \cdot (1-1)!} \\
&+ \frac{c_2(n-0+2-1)!2^{n-0}}{(n-0)! \cdot (2-1)!} \\
=& -c_1 + 2^{n+1}c_1 + 2^n(n+1)c_2
\end{aligned} \tag{7}
$$

This can be proven by induction. Let the inductive predicate be:

$$P(n) ::= \forall n \in N. T_n = -c_1 + 2^{n+1}c_1 + 2^n(n+1)c_2$$

Base case

$$n = 0$$

:

$$
\begin{aligned}
T_n &= -c_1 + 2c_1 + 2^0(0+1)c_2 \\
&= c_1 + c_2
\end{aligned} \tag{8}
$$

Case holds.

Inductive step:

$$
\begin{aligned}
T_{n+1} &= c_1 + 2 \cdot T_n + 2^{n+1}c_2 \\
&= c_1 + 2 \cdot (-c_1 + 2^{n+1}c_1 + 2^n(n+1)c_2) + 2^{n+1}c_2 \text{ (by the inductive hypothesis)} \\
&= c_1 + (2^{n+2} - 2)c_1 + 2^{n+1}(n+1)c_2 + 2^{n+1}c_2 \\
&= (2^{n+2} - 1)c_1 + 2^{n+1}(n+2)c_2 \\
&= (2^{(n+1)+1} - 1)c_1 + 2^{n+1}((n+1)+1)c_2
\end{aligned} \tag{9}
$$

Case holds.