Question 4 by Dan Nguyen (z5206032)

Recurrence

Using recurrence to solve a problem of size n reduces the problem to a many subproblems of a smaller size n/b. The overhead cost of reducing the problem and combining the solutions of the subproblems is $f(n)$.

The time complexity of the recurrence is given as:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n) \tag{1}$$

Reducing the $T(n/b)$ term $log_b a$ times yields:

$$T(n) \approx n^{log_b a} T(1) + \sum_{i=0}^{\lfloor log_b n \rfloor - 1} a^i f\left(\frac{n}{b^i}\right) \tag{2}$$

Solution

There is a given set of real numbers, $S = \{t_1, t_2, ..., t_n\}$ of size n.

There is a given polynomial, P of degree n and leading coefficient 1, with unknown coefficients:

$$P(x) = x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + ... + a_2x^2 + a_1x + a_0 \tag{3}$$

such that for each element of S inputted into P, the outputs are equivalent:

$$P(t_1) = P(t_2) = ... = P(t_n) \tag{4}$$

Consider Equation 3 in factored form which satisfies Equation 4:

$$P(x) = (x - t_1)(x - t_2)...(x - t_n) \tag{5}$$

Using divide-and-conquer, split Equation 5 into two subpolynomials i.e. two subproblems each of size $n/2$:

$$P(x) = P_1(x) \cdot P_2(x)$$

such that:

$$P_1(x) = (x - t_1)(x - t_2)...(x - t_{n/2}) \tag{6}$$

$$P_2(x) = (x - t_{n/2+1})(x - t_{n/2+2})...(x - t_n) \tag{7}$$

The division of polynomials into two subpolynomials has a time complexity of $O(1)$. The coefficients of P can be computed through the polynomial multiplication of P_1 and P_2 in the

expanded form (if the coefficients of the subpolynomials are known) using the Fast Fourier Transform (FFT) for a time complexity of $O(nlog(n))$.

The coefficients of P_1 and P_2 are found recursively using the same divide-and-conquer algorithm. The smallest subproblem of the divide-and-conquer algorithm has the form: $(x-t_k)(x-t_{k+1})$. A simple multiplication and addition of the roots (which are atomic operations) yields a degree 2 polynomial with known coefficients of the form: $x^2-(t_k+t_{k+1})x+t_kt_{k+1}$. The expanded polynomial is used to compute the FFT.

Therefore, the recurrence is:

$$T(n) = 2T\left(\frac{n}{2}\right) + nlog(n) \tag{8}$$

Using the alternative form of the recurrence (from Equation 2) yields:

$$\begin{aligned}
T(n) &\approx nT(1) + \sum_{i=0}^{\lfloor log_2 n \rfloor - 1} 2^i f\left(\frac{n}{2^i}\right) \\
&= f(n) + 2f\left(\frac{n}{2}\right) + 4f\left(\frac{n}{4}\right) + ... + \frac{n}{2}f\left(\frac{n}{n/2}\right) \\
&= nlog_2(n) + nlog_2\left(\frac{n}{2}\right) + nlog_2\left(\frac{n}{4}\right) + ... + nlog_2\left(\frac{n}{n/2}\right) \\
&= n\left(log_2\left(\frac{n}{1}\times\frac{n}{2}\times\frac{n}{4}\times...\times\frac{n}{n/2}\right)\right) \\
&= n\left(log_2\left(\frac{n^{log_2 n}}{n^{\frac{log_2(n)}{2}}}\right)\right) \\
&= n\left(log_2\left(n^{log_2 n}\right) - log_2\left(n^{\frac{log_2(n)}{2}}\right)\right) \\
&= nlog_2(n)log_2(n) - \frac{n}{2}log_2\left(\frac{n}{2}\right)log_2(n) \\
&\in O(nlog_2^2(n))
\end{aligned}$$

Therefore, the recurrence algorithm has a time complexity of $O(nlog_2^2(n))$ as required.