Divide and Conquer Algorithms and Recurrence Relations

Debdeep Mukhopadhyay

IIT Madras

Divide & Conquer Algorithms

- Many types of problems are solvable by reducing a problem of size *n* into some number *a* of independent subproblems, each of size $\leq \lceil n/b \rceil$, where $a \geq 1$ and $b > 1$.
- The time complexity to solve such problems is given by a recurrence relation:
  - $T(n) = a\,T(\lceil n/b \rceil) + g(n)$

Time for each subproblem

Time to combine the solutions of the subproblems into a solution of the original problem.

Why the name?

- Divide: This step divides the problem into one or more substances of the same problem of smaller size
- Conquer: Provides solutions to the bigger problem by using the solutions of the smaller problem by some additional work.

Divide and Conquer Examples

- **Binary search:** Break list into 1 sub-problem (smaller list) (so a=1) of size $\leq\lceil n/2\rceil$ (so b=2).
  - So $T(n) = T(\lceil n/2\rceil) + 2$ ($g(n)=c$ constant)
  - g(n)=2, because two comparisons are needed to conquer. One to decide which half of the list to use. Second to decide whether any term in the list remain.

Find the maximum and minimum of a sequence

- If n=1, the number is itself min or max
- If n>1, divide the numbers into two lists. Decide the min & max in the first list. Then choose the min & max in the second list.
- Decide the min & max of the entire list.
- Thus,

$$\boxed{T(n)=2T(n/2)+2}$$

Fast Multiplication Example

- The ordinary grade-school algorithm takes $\Theta(n^2)$ steps to multiply two n-digit numbers.
  - Can we do better?
- Let's find an asymptotically *faster* algorithm!
- To find the product cd of two $2n$-digit base-b numbers, $c=(c_{2n-1}c_{2n-2}\ldots c_0)_b$ and $d=(d_{2n-1}d_{2n-2}\ldots d_0)_b$, first, we break c and d in half:

$$c=b^nC_1+C_0, \qquad d=b^nD_1+D_0$$

Derivation of Fast Multiplication

$$\begin{aligned}
cd &= (b^n C_1 + C_0)(b^n D_1 + D_0) \\
&= b^{2n} C_1 D_1 + b^n (C_1 D_0 + C_0 D_1) + C_0 D_0 \\
&= b^{2n} C_1 D_1 + C_0 D_0 + \\
&\quad b^n (C_1 D_0 + C_0 D_1 + (C_1 D_1 - C_1 D_1) + (C_0 D_0 - C_0 D_0)) \\
&= (b^{2n} + b^n) C_1 D_1 + (b^n + 1) C_0 D_0 + \\
&\quad b^n (C_1 D_0 - C_1 D_1 - C_0 D_0 + C_0 D_1) \\
&= (b^{2n} + b^n) C_1 D_1 + (b^n + 1) C_0 D_0 + \\
&\quad b^n (C_1 - C_0)(D_0 - D_1)
\end{aligned}$$

(Multiply out polynomials)

Zero

(Factor last term)

Three multiplications, each with *n*-digit numbers

Recurrence Rel. for Fast Mult.

Notice that the time complexity $T(n)$ of the fast multiplication algorithm obeys the recurrence:

- $T(2n)=3T(n)+\Theta(n)$
  i.e.,
- $T(n)=3T(n/2)+\Theta(n)$

  So $a=3$, $b=2$.

Time to do the needed adds & subtracts of n-digit and $2n$-digit numbers

Solving the R.R

- We have seen some approaches before.
- We shall discuss some more useful techniques
- Let, $n=b^k$, k is a positive integer
  - $f(n)=af(n/b)+g(n)$

    $=a^2f(n/b^2)+ag(n/b)+g(n)$

    $=a^3f(n/b^3)+a^2g(n/b^2)+ag(n/b)+g(n)$

    … $=a^kf(n/b^k)+\Sigma_0^{k-1}a^jg(n/b^j)$.

  If $n=b^k$, we have $f(1)$ in place of n/b^k.

Theorem

- Let f be a non-decreasing function satisfying: f(n)=af(n/b)+c, where n is divisible by b, a≥1, b is an integer greater than 1, and c is a positive real number.
- Then

$$f(n) = \begin{cases} O(n^{\log_b a}), a > 1 \\ O(\log_b n), a = 1 \end{cases}$$

Theorem contd.

- When $n=b^k$, we have further:

$$f(n) = C_1 n^{\log_b a} + C_2,$$

where $C_1 = f(1) + c/(a-1), C_2 = -c/(a-1)$

Examples

- $f(n)=5f(n/2)+3$, $f(1)=7$. Find $f(2^k)$, k is a positive integer
- $f(n)=5^k f(1)+3(1+5+5^2+\ldots+5^{k-1})$

  $=5^k f(1)+3(5^k-1)/4$ [GP series]

  $=5^k[f(1)+3/4]-3/4$

Since, f(n) is a non-decreasing function,

f(n) is $O(n^{\log_2 5})$.

Examples

- **Estimate the number of searches in Binary Search**

Solve: $f(n)=f(n/2)+2$

$a=1 \Rightarrow f(n)=O(\log_2 n)$

- **Estimate the number of comparsons to find the min-max of a sequence (using the algo previously stated)**

Solve: $f(n)=2f(n/2)+2$

$f(n)= O(n^{\log_2 2}) = O(n)$

The Master Theorem

Consider a function $f(n)$ that, for all $n=b^k$ for all $k\in\mathbf{Z}^+$, satisfies the recurrence relation:

$$f(n) = af(n/b) + cn^d$$

with $a\geq 1$, integer $b>1$, real $c>0$, $d\geq 0$. Then:

$$\boxed{f(n)\in\begin{cases}O(n^d) & \text{if } a<b^d\\ O(n^d\log n) & \text{if } a=b^d\\ O(n^{\log_b a}) & \text{if } a>b^d\end{cases}}$$

Master Theorem Example

- Recall that complexity of fast multiply was:

  $T(n) = 3T(n/2) + \Theta(n)$

- Thus, a=3, b=2, d=1. So $a > b^d$, so case 3 of the master theorem applies, so:

$$\boxed{T(n) = \mathrm{O}(n^{\log_b a}) = \mathrm{O}(n^{\log_2 3})}$$

which is $\mathrm{O}(n^{1.58\ldots})$, so the new algorithm is strictly faster than ordinary $\Theta(n^2)$ multiply!