Oscar, Nathan, Brayden

Math 251, Fri 12-Nov-2021 -- Fri 12-Nov-2021
Discrete Mathematics
Fall 2021

Friday, November 12th 2021

Wk 11, Fr
Topic:: Divide-and-conquer algorithms
Read:: Rosen 8.3

Binary search is an example of divide-and-conquer algorithm

- At each stage, a comparison leads to "halving" the problem
- More specifically, if

    n = size of the list (suppose it is even), and
    f(n) = worst-case count of operations needed to find key in list of size n

  then

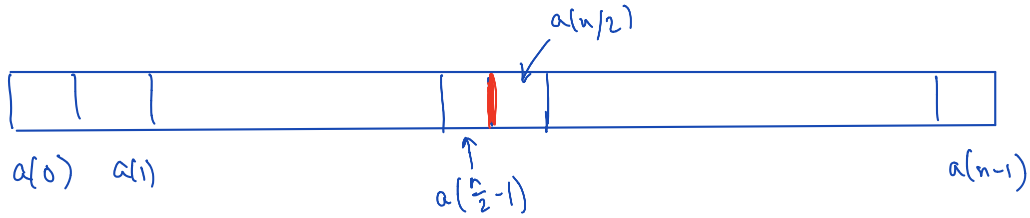
    f(n) = f(n/2) + 2

Speaking generally, if we continue to define
  n = size of the problem/data, and
  f(n) = count of operations needed to carry out algorithm on data of size n
and if our algorithm has the property that
  $\boxed{f(n) = a\ f(n/b) + g(n)}$
then our algorithm is classified as a divide-and-conquer algorithm.

The relation
  f(n) = a f(n/b) + g(n)
with a, b constants, is called a divide-and-conquer relation.

Binary search

- start with a sorted list of size n (even)



- looking for a key k

  Compare k with $a\left(\frac{n}{2}-1\right)$

  Another comparison to see if a half-list is of length 1

At cost of 2 comparisons

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

Ch. 8 focus: Solving recurrences. 2 approaches

1. linear, const. coeff., homog., k^{th} degree (8.2A)
2. iterative approach

   relevant for DAC recurrences

Ex.) Suppose $\boxed{f(n) = a f(n/b) + c}$ (we have a DAC recurrence, $g(n) = c$).

and $n = b^k$. Use iteration to solve the recurrence:

$$f(b^k) = a f(b^k/b) + c = a f(b^{k-1}) + c \quad \text{(1 iteration)}$$

$$= a\left[a f(b^{k-1}/b) + c\right] + c = a^2 f(b^{k-2}) + ac + c \quad \text{(2nd iteration)}$$

$$= a^2\left[a f(b^{k-2}/b) + c\right] + ac + c = a^3 f(b^{k-3}) + a^2c + ac + c$$

$$= \cdots$$

$$= a^k f(b^0) + a^{k-1}c + a^{k-2}c + \cdots + a^2c + ac + c$$

$$= a^k f(1) + c\underbrace{\left(1 + a + a^2 + \cdots + a^{k-1}\right)}_{\text{terms in geometric seq.}}$$

requires $a \neq 1$

$$= a^k f(1) + c\,\frac{a^k - 1}{a - 1} = a^k f(1) + \frac{ca^k}{a-1} - \frac{c}{a-1}$$

$$= a^k\left(f(1) + \frac{c}{a-1}\right) - \frac{c}{a-1}$$

Now use 2 things

1. Assumed $n = b^k$ which means $k = \log_b n$

$$\Rightarrow \quad a^k = a^{\log_b n}$$

2. $a^{\log_b n} = n^{\log_b a}$. (Surprising?)

Together with what came before

$$\boxed{f(n) = n^{\log_b a}\left(f(1) + \frac{c}{a-1}\right) - \frac{c}{a-1}}$$

Theorems from Rosen, 5th Ed., Section 8.3

Theorem 1: Suppose f is an increasing function which satisfies the recurrence relation

$$f(n) = af(n/b) + c,$$

whenever n is an integer divisible by (integer) $b > 1$. Suppose $a \geqslant 1$ and $c > 0$. Then

$$f(n) \text{ is } \begin{cases} O(n^{\log_b a}), & \text{if } a > 1, \\ O(\log n), & \text{if } a = 1. \end{cases}$$

In addition, when $n = b^k$ for integer $k > 0$, we have

$$f(n) \;=\; \left(f(1) + \frac{c}{a-1}\right) n^{\log_b a} - \frac{c}{a-1}.$$

Theorem 2 (Master Theorem): Let f be an increasing function that satisfies the recurrence relation

$$\boxed{f(n) = af(n/b) + cn^d,}$$

whenever $n = b^k$, where k is a positive integer, $a \geqslant 1$, b is an integer greater than 1, and $c > 0$, $d \geqslant 0$ are real numbers. Then

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

1. $T(n) = 3T(n/2) + n^2$

2. $f(n) = 3f(n/3) + n/2$

3. $f(n) = 4f(n/2) + n/2$

Taking these three DAC recurrences in turn, the Master Theorem says

1. $b=2$, $c=1$, $d=2$, $a=3$ $\rightarrow$ $3 < 2^2$, so $T(n)$ is $O(n^2)$.
2. $a=3$, $b=3$, $c=1/2$, $d=1$ $\rightarrow$ $3 = 3^1$, so $f(n)$ is $O(n \log n)$.
3. $a=4$, $b=2$, $c=1/2$, $d=1$ $\rightarrow$ $4 > 2^1$, so $f(n)$ is $O(n^{\log_2 4}) = O(n^2)$.

Divide and Conquer

Suppose $f(n)$ is the count of operations required, using a certain algorithm, to perform a task of size n (n is a measure on the input to the algorithm). If f satisfies a recurrence relation of the form

$$f(n) = af(n/b) + g(n), \tag{1}$$

with $a, b > 0$, called a **divide-and-conquer** recurrence relation, then the algorithm is said to be a **divide-and-conquer** algorithm.

Example 1:

1. **Binary search**. Take $f(n)$ to be the number of comparisons required to find a search key in an ordered list of length n using the binary search algorithm. (See Section 2.1). Then $f(n) = f(n/2) + 2$.
2. **Fast integer multiplication**. Let $f(n)$ be the count of bit operations required to multiply two $(2n)$-bit integers. Let a, b be two such integers with binary representations

$$a = (a_{2n-1} \dots a_2 a_1 a_0)_2 \qquad \text{and} \qquad b = (b_{2n-1} \dots b_2 b_1 b_0)_2,$$

and write $a = A_0 + 2^n A_1$, $a = B_0 + 2^n B_1$, so that each of A_0, A_1, B_0, B_1 are n-bit numbers; note that

$$A_0 = (a_{n-1} \dots a_2 a_1 a_0)_2 \qquad \text{and} \qquad A_1 = (a_{2n-1} \dots a_{n+2} a_{n+1} a_n)_2,$$

with similar relationships between the binary represenions for B_0, B_1 and b. By writing

$$\begin{aligned} ab &= (A_0 + 2^n A_1)(B_0 + 2^n B_1) = 2^{2n} A_1 B_1 + 2^n (A_0 B_1 + A_1 B_0) + A_0 B_0 \\ &= (2^{2n} + 2^n) A_1 B_1 - 2^n A_1 B_1 + 2^n (A_0 B_1 + A_1 B_0) - 2^n A_0 B_0 + (2^n + 1) A_0 B_0 \\ &= (2^{2n} + 2^n) A_1 B_1 + 2^n (A_1 - A_0)(B_0 - B_1) + (2^n + 1) A_0 B_0 \end{aligned}$$

and interpreting multiplications like $2^k C$ as a *sliding* of bits k places to the left (rather than actual multiplication), we see that the problem of multiplying two $(2n)$-bit integers a and b has been replaced with three multiplications involving n-bit integers, along with several slidings, subtractions and additions, the count of which is proportional to n. Thus,

$$f(2n) = 3f(n) + Cn.$$

3. Consider the number of comparisons required to sort a list of n items via the *merge sort* algorithm described in Section 3.5 (Rosen, 7^{th} ed.). This algorithm, for even n, divides the list into two lists of size $n/2$ and, once the two sub-lists are sorted, requires fewer than n comparisons to merge the two sorted sub-lists into one complete (and sorted) list. Thus, the number of comparisons used by the algorithm on a list of size n is less than $M(n)$, a function which satisfies the divide-and-conquer relation

$$M(n) = 2M(n/2) + n.$$

■

Some relevant details

Logarithms. Write $r = log_b x$ when $b^r = x$. Said another way, $\log_b x$ returns the number r for which $b^r = x$. Some properties that arise from this idea:

1. $b^{\log_b x} = x$, akin to saying the number of ounces in a 32-ounce jar is 32.
2. $\log_b(xy) = \log_b x + \log_b y$, since
$$b^{\log_b x + \log_b y} \;=\; b^{\log_b x} \cdot b^{\log_b y} \;=\; xy.$$
3. $\log_b(x/y) = \log_b x - \log_b y$, demonstrated similarly.
4. $\log_b(x^r) = r \log_b x$, since
$$b^{r \log_b x} \;=\; (b^{\log_b x})^r \;=\; x^r.$$
5. $\log_a x = \log_b x / \log_b a$, since
$$b^{(\log_a x)(\log_b a)} \;=\; (b^{\log_b a})^{\log_a x} \;=\; a^{\log_a x} \;=\; x.$$
Thus, $(\log_a x)(\log_b a)$ is the exponent to which, when b is raised, yields x—i.e., it equals $\log_b x$.
6. For positive real numbers a, b, and c,
$$\boxed{a^{\log_b c} \;=\; c^{\log_b a}.}$$
$$c^{\log_b n} = n^{\log_b a}$$
This is true because
$$\log_a\left(c^{\log_b a}\right) \;=\; (\log_b a)\,(\log_a c) \;=\; \log_b c,$$
by Property 5 above. This means that $\log_b c$ is the power to which you must raise a in order to produce $c^{\log_b a}$.
7. $O(\log_b n)$ is independent of base b. That is, if a is any other base, and if $|f(n)| \leqslant C|\log_b n|$ (the meaning of $O(\log_b n)$), then by Property 5 above,
$$|f(n)| \;\leqslant\; C|\log_b n| \;=\; \frac{C}{|\log_a b|}|\log_a n| \;=\; \tilde{C}|\log_a n|,$$
which shows f is $O(\log_a n)$ as well. Convention, then, is to write $O(\log n)$ without reference to a particular base b.

Question: For an integer n, how many stages of dividing into b parts, then subdividing those parts into b parts, and so on, may be carried out before all constituent parts are of size 1?

Answer: We can develop some intuition by investigating the number of ways to divide an integer by 2. The numbers $5, 6, 7$, and 8 each require 3 stages. The numbers $9, 10, 11, 12, 13, 14, 15$, and 16 require 4 stages. In general the integers $2^{k-1} < n \leqslant 2^k$ all require $k = \log_2 2^k = \lceil \log_2 n \rceil$ stages.

Speaking generally, if an integer n satisfies $b^{k-1} < n \leqslant b^k$ and, at each stage, is to be divided into b parts, then it requires $k = \log_b b^k = \lceil \log_b n \rceil$ stages.

Important theorems

When f satisfies the divide-and-conquer relation (1) and n has b^k as a factor, we have

$$\begin{aligned}
f(n) &= af(n/b) + g(n) \;=\; a\left(af(n/b^2) + g(n/b)\right) + g(n) \\
&= a^2 f(n/b^2) + ag(n/b) + g(n) \\
&= a^3 f(n/b^3) + a^2 g(n/b^2) + ag(n/b) + g(n) \;=\; \cdots \\
&= a^k f(n/b^k) + \sum_{j=0}^{k-1} a^j g(n/b^j).
\end{aligned}$$

In the special case where $g(n) = c$ (a constant), this becomes

$$f(n) = a^k f(n/b^k) + c\sum_{j=0}^{k-1} a^j = \begin{cases} a^k f(n/b^k) + ck, & \text{if } a = 1, \\ a^k f(n/b^k) + \frac{c(a^k-1)}{a-1}, & \text{if } a > 1. \end{cases} \tag{2}$$

This gives rise to the following theorem.

Theorem 3: Suppose f is an increasing function which satisfies the recurrence relation

$$f(n) = af(n/b) + c,$$

whenever n is an integer divisible by (integer) $b > 1$. Suppose $a \geqslant 1$ and $c > 0$. Then

$$f(n) \text{ is } \begin{cases} O(n^{\log_b a}), & \text{if } a > 1, \\ O(\log n), & \text{if } a = 1. \end{cases}$$

In addition, when $n = b^k$ for integer $k > 0$, we have

$$f(n) = \left(f(1) + \frac{c}{a-1}\right) n^{\log_b a} - \frac{c}{a-1}.$$

Proof: **Case**: $n = b^k$ (so $k = \log_b n$).
If $a = 1$, then Equation (2) says

$$f(n) = f(1) + ck = f(1) + c\log_b n,$$

showing f is $O(\log n)$.

Now suppose $a > 1$. Equation (2) says

$$f(n) = a^k f(1) + \frac{c(a^k-1)}{a-1} = a^{\log_b n}\left(f(1) + \frac{c}{a-1}\right) - \frac{c}{a-1} = n^{\log_b a}\left(f(1) + \frac{c}{a-1}\right) - \frac{c}{a-1}.$$

General Case. When n is not a power of b, there is an integer $k \geqslant 0$ such that $b^k < n < b^{k+1}$. We treat the case with $a > 1$ only. Because f is an increasing function,

$$f(n) \leqslant f(b^{k+1}) = C_1 a^{k+1} + C_2 = (C_1 a)a^k + C_2 = (C_1 a)a^{\log_b n} + C_2,$$

where $C_1 = f(1) + c/(a-1)$ and $C_2 = -c/(a-1)$. Hence, the result holds. $\square$

The previous result is applicable to the binary search algorithm which, as we found, gives rise to the recurrence relation $f(n) = f(n/2) + 2$. To draw conclusions about the divide-and-conquer recurrence relations of fast integer multiplication and the merge sort, we need a more general theorem.

Theorem 4 (Master Theorem): Let f be an increasing function that satisfies the recurrence relation

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

whenever $n = b^k$, where k is a positive integer, $a \geqslant 1$, b is an integer greater than 1, and $c > 0$, $d \geqslant 0$ are real numbers. Then

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

Proof: If $a = b^d$ and $n = b^k$, then

$$\begin{aligned} f(n) &= af(n/b) + cn^d = a\left[af(n/b^2) + c\left(\frac{n}{b}\right)^d\right] + cn^d \\ &= a^2 f(n/b^2) + ac\left(\frac{n}{b}\right)^d + cn^d \\ &= a^3 f(n/b^3) + a^2 c\left(\frac{n}{b^2}\right)^d + ac\left(\frac{n}{b}\right)^d + cn^d = \cdots \\ &= a^k f(1) + cn^d \sum_{j=0}^{k-1}\left(\frac{a}{b^d}\right)^j = (b^d)^k f(1) + cn^d \sum_{j=0}^{k-1} 1 \\ &= f(1)n^d + ckn^d = f(1)n^d + cn^d \log_b n. \end{aligned}$$

Now, assume $k \geqslant 0$ is such that $b^k < n \leqslant b^{k+1}$. Because f is an increasing function, we have

$$\begin{aligned} f(n) &\leqslant f(b^{k+1}) = f(1)b^{(k+1)d} + c(k+1)b^{(k+1)d} \\ &= f(1)b^d \cdot (b^k)^d + cb^d \cdot (b^k)^d + cb^d \cdot (b^k)^d k \\ &\leqslant [f(1) + c]an^d + can^d \log_b n. \end{aligned}$$

Thus, in the case $a = b^d$, we have the desired result, as the $n^d \log n$ term above dominates the n^d term. □

Examples:

1. Suppose $T(n) = 3T(n/2) + n^2$.

   By the Master Theorem, taking $a = 3$, $b = 2$, $c = 1$ and $d = 2$, we have $T(n)$ is $O(n^2)$, since $3 < 2^2$.
2. Suppose $f(n) = 3T(n/3) + n/2$.

   By the Master Theorem, taking $a = 3$, $b = 3$, $c = 1/2$ and $d = 1$, we have $T(n)$ is $O(n \log n)$, since $3 = 3^1$.
3. Suppose $f(n) = 4T(n/2) + n/2$.

   By the Master Theorem, taking $a = 4$, $b = 2$, $c = 1/2$ and $d = 1$, we have $T(n)$ is $O(n^{2 \log_2 2})$, since $4 > 2^1$.