Why Some Algorithms Divide Problems Equally

Nitin Verma
mathsanew.com

April 18, 2021

There are many algorithms which divide any problem of size n into two subproblems of equal size. *Merge-Sort* and *Binary-Search* are two examples. Some algorithms like *Quick-Sort* may not be able to control the two subproblems' sizes, but work more efficiently when the subproblems are of equal size. Whenever we need to divide a problem into two subproblems, why it is often good to have the two subproblems' sizes equal?

Suppose there is a problem for which we know how to divide any instance of the problem into two subproblems and combine their results to solve the original instance. Also, we are able to associate a *size* with each problem instance such that the size of the two subproblems (which need not be equal) add up to the original problem's size. We can recursively apply this dividing procedure until the problem instances are of size at most N_0. We refer such instances as "Base Instances" and solve them by some other method. In this article, we will refer to such a *Divide-and-Conquer* algorithm as "D2C" (as it divides into 2) algorithm.

For an initial problem instance of size n, there must be at least n/N_0 such base-instances. If every such base-instance requires amount of *work* (number of operations) at least W_0, then total work required by all of them is nW_0/N_0. Additionally, some work is also required to create subproblems and combine their results; but we do not account that here. So, we can conclude that the work required to solve the problem instance of size n, by any such D2C algorithm, is *at least* nW_0/N_0 which is linear in n.

Estimating Subproblem's Work

Suppose for a given problem of size n, we know a way to divide it into two subproblems and combine their results. But we need to find out, in what sizes should the two subproblems be created so as to minimize the total

work. For this, we will first need an estimation of how the work required by each subproblem relates to its size.

For a subproblem of size s ($s \geq 1$), let $f(s)$ denote an estimate of the amount of work required to solve it. Note that $f(s)$ will depend upon how exactly we solve the subproblem. We can recursively solve it by dividing it into two subproblems of certain sizes (say, equal sizes), and find an estimate based on that. As we have found above, any D2C algorithm will involve at least some work which is linear in s. Sometimes, the nature of the problem can also help us relate problem size to the work required; for example, sorting by comparison requires at least $\Theta(s \log_2(s))$ comparisons.

For this discussion, we only need to find out whether the estimated $f(s)$ follows the below two properties or not (instead of finding the precise $f(s)$), for any positive integers $s_1 < s_2$:

1. $f(s_1) \leq f(s_2)$. That is, the work required is non-decreasing with s.
2. If f' denotes the *Derivative* of f (*Differential Calculus*), then $f'(s_1) \leq f'(s_2)$. That is, the rate of increase of f (slope of its graph) is non-decreasing with s.

Some simple examples of such function f are ($c > 0$ is a constant):

$f(s) = cs$	$f'(s) = c$
$f(s) = cs^2$	$f'(s) = 2cs$
$f(s) = cs \log_2(s)$	$f'(s) = c \log_2(s) + c/\log_e(2)$

Optimizing Subproblems Sizes

Having estimated the work required by a subproblem, we will now try to find out the optimal sizes of the two subproblems. For this analysis, assume n to be even. So, the subproblems' sizes can also be expressed as $(n/2 - a)$ and $(n/2 + a)$, for some a with $0 \leq a < n/2$. The work required to solve the two subproblems is:

$$w(a) = f\left(\frac{n}{2} - a\right) + f\left(\frac{n}{2} + a\right)$$

We assume that the work required to create the two subproblems and combine their results remains same, whatever be the two subproblems' size distribution. So, to minimize the total work required for the size n problem, we want to pick a which minimizes $w(a)$. Consider the value of f at $(n/2 - a)$

and $n/2$. Due to the *Mean Value Theorem* from Differential Calculus, we know that there must exist b_1 in interval $(n/2 - a, n/2)$ such that:

$$f\left(\frac{n}{2}\right) - f\left(\frac{n}{2} - a\right) = f'(b_1)\left(\frac{n}{2} - \left(\frac{n}{2} - a\right)\right) = f'(b_1)a \tag{1}$$

(Note that although $n/2$ and a are integers, but b_1 need not be one.) Similarly, there must exist b_2 in interval $(n/2, n/2 + a)$ such that:

$$f\left(\frac{n}{2} + a\right) - f\left(\frac{n}{2}\right) = f'(b_2)\left(\left(\frac{n}{2} + a\right) - \frac{n}{2}\right) = f'(b_2)a \tag{2}$$

As $b_1 \leq b_2$, so due to property (2) of f, $f'(b_1) \leq f'(b_2)$. As $a \geq 0$, $f'(b_1)a \leq f'(b_2)a$. Thus, equations (1) and (2) imply:

$$f\left(\frac{n}{2}\right) - f\left(\frac{n}{2} - a\right) \leq f\left(\frac{n}{2} + a\right) - f\left(\frac{n}{2}\right)$$

$$\Leftrightarrow \qquad f\left(\frac{n}{2}\right) + f\left(\frac{n}{2}\right) \leq f\left(\frac{n}{2} - a\right) + f\left(\frac{n}{2} + a\right) = w(a)$$

So, we should pick $a = 0$ to minimize $w(a)$. That means to choose the two subproblems sizes as $n/2$ each. Note that, although we are interested in values of f only at integers, but we could bring in Differential Calculus as a useful tool.

Discarding of Subproblems

For some kinds of problems, we may need to solve only one of the two subproblems, discarding the other one. For example, Binary-Search algorithm discards half of the array (one of the two subproblems) after each comparison. In situations where we may end up discarding some of the subproblems, the number of base-instances may be less than linear in n, and so will be the total work required. For example, Binary-Search involves $O(\log_2(n))$ work.

How do we decide the two subproblems sizes in such algorithms? Say, we want to optimize the worse case where the "costlier" of the two subproblems gets picked, discarding the other one. The costlier subproblem requires work:

$$w'(a) = max\left(f\left(\frac{n}{2} - a\right), f\left(\frac{n}{2} + a\right)\right) = f\left(\frac{n}{2} + a\right)$$

Because, if f follows property (1), $f(n/2 + a)$ is costlier. To minimize it, we choose $a = 0$, i.e. the subproblems of size $n/2$ each.

■