Chapter 3

Hybrid Algorithms

The idea behind the design of hybrid algorithms is very simple. Assume that, for a given computational problem Π, we are given two algorithms, a recursive algorithm A_1 and another algorithm A_2 (usually iterative, although not necessarily so). Let A_1 be asymptotically faster than A_2, with the former exhibiting higher constants in its running time than the latter. We can create a recursive *hybrid algorithm* H by modifying the code of A_1 in the base case as follows: we introduce a new value of the instance size, say n_0, under which we solve directly using algorithm A_2. The remaining code stays unchanged, apart from substituting recursive calls to A_1 with recursive calls to H. Therefore the code for H will look as follows:

```
$H(i)$
**if** size($i$) $\leq n_0$
  **then return** $A_2(i)$
$\star$ DIVIDE step of $A_1$ $\star$
$\star$ RECURSE step of $A_1$ (but using $H$ for the calls)$\star$
$\star$ CONQUER step of $A_1$ $\star$
```

Our objective in designing H is to obtain the best from both algorithms, namely, keeping the asymptotic behaviour of A_1 but lowering its constants thanks to the use of A_2 on smaller instances.

Note that in the code above, the “switching point” n_0 between the application of the two algorithms is a parameter whose actual value must be determined and fixed by the analysis, so to minimize the running time of the hybrid algorithm. In what follows, we will discuss how to proceed analytically to obtain the best choice of n_0 for the case of square matrix multiplication, where we can obtain a hybrid algorithm from Strassen’s algorithm and the naive, iterative algorithm based on the definition.

3.1 Matrix Multiplication

The hybrid algorithm H-SMUL for matrix multiplication obtained from Strassen's algorithm SMUL and definition-based algorithm MUL is:

```
H-SMUL($A, B$)
$n \leftarrow \text{rows}(A)$
if $n \leq n_0$
    then return MUL($A, B$)
$\star$ DIVIDE step of SMUL $\star$
$\star$ RECURSE step of SMUL (but using H-SMUL for the calls) $\star$
$\star$ CONQUER step of SMUL $\star$
```

Recall that, when n is a power of two, the respective running times $T_S(n)$ and $T_M(n)$ of SMUL and MUL are

$$\begin{aligned} T_S(n) &= 7n^{\log_2 7} - 6n^2 \\ T_M(n) &= 2n^3 - n^2 \end{aligned}$$

hence SMUL features a better asympotic complexity but rather larger constants than MUL. In fact, the first value of n (power of two) for which SMUL is faster than MUL is as high as 1024! Let us now formulate a recurrence $T(n, n_0)$ to evaluate the running time of algorithm H-SMUL.

$$T(n, n_0) = \begin{cases} 2n^3 - n^2 & n \leq n_0 \\ 7T(n/2, n_0) + (9/2)n^2 & n > n_0 \end{cases}$$

Observe that this recurrence has two parameters: n and n_0. Since Strassen's algorithm only works for matrices which are a power of two, it is reasonable to assume that also n_0 be a power of two. Let us now proceed to determine an analytic solution to the above recurrence as a function of its two parameters. From the recursion tree associated with the recurrence, we can collect the following information for values of $n > n_0$, given below in tabular form for convenience:

tree level	**instance size**	**# nodes in level**	**work per node**
0	n	1	$(9/2)n^2$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
ℓ	$n/2^\ell$	7^ℓ	$(9/2)(n^2/4^\ell)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\log_2(n/n_0)$	n_0	$(n/n_0)^{\log_2 7}$	$2{n_0}^3 - {n_0}^2$

In the above table, the expression for the last level is obtained by observing that the leaves are at a level $\hat{\ell}$ such that $n/2^{\hat{\ell}} = n_0$, whence $\hat{\ell} = \log(n/n_0)$ (from now on, we omit the base 2 of the logarithm for brevity). By summing over all levels the component-wise product of the last two columns of the table, we obtain:

$$\begin{aligned} T(n,n_0) &= (9/2)n^2\left(\sum_{\ell=0}^{\log(n/n_0)-1}(7/4)^\ell\right) + (n/n_0)^{\log 7}(2{n_0}^3 - {n_0}^2) \\ &= 6n^2\left((7/4)^{\log(n/n_0)} - 1\right) + n^{\log 7}\frac{(2n_0-1)}{{n_0}^{\log 7 - 2}} \\ &= 6n^2\left(\left(\frac{n}{n_0}\right)^{\log(7)-2} - 1\right) + n^{\log 7}\frac{(2n_0-1)}{{n_0}^{\log 7 - 2}} \\ &= n^{\log 7}\frac{2n_0+5}{{n_0}^{\log 7 - 2}} - 6n^2. \end{aligned}$$

Observe that $T(n,1) = 7n^{\log 7} - 6n^2 = T_S(n)$, while $T(n,n) = 2n^3 - n^2 = T_M(n)$, which is to be expected, since the choice $n_0 = 1$ yields an algorithm which is identical to SMUL, while choosing $n_0 = n$ implies that we always invoke MUL. Also, observe that different values of n_0 yield different values of the multiplicative constant in the leading term of $T(n,n_0)$.

In order to determine an optimal value for n_0, we study the sign of the partial derivative of $T(n,n_0)$ with respect to n_0. Easy calculations show that

$$\frac{\delta T(n,n_0)}{\delta n_0} = \frac{n^{\log 7}}{{n_0}^{\log 7 - 1}}\left(2(3-\log 7)n_0 - 5(\log 7 - 2)\right)$$

whence

$$\frac{\delta T(n,n_0)}{\delta n_0} \geq 0 \Leftrightarrow n_0 \geq \frac{5(\log 7 - 2)}{2(3-\log 7)} \simeq 10.48\ldots$$

Since we have to pick a value of n_0 which is a power of two, we check $T(n,n_0)$ choosing for n_0 the powers of two immediately smaller and larger than 10.48, and pick the value yielding the smallest running time. It turns out that the best switching point is $n_0 = 8$, which yields a running time for H-SMUL

$$T(n,8) \simeq 3.92n^{\log 7} - 6n^2.$$

Observe that we have succeeded in almost halving the multiplicative constant of the leading term over $T_S(n)$.

Recall that our cost model disregards a number of aspects of a real machine that make multiplicative constants not particularly significant. This means that the above analysis guarantees that the running time of Strassen's algorithm can be improved through hybridization *only in principle*, but cannot give the most appropriate value of n_0 for a real implementation. In fact, on different machines the best algorithm is likely to be obtained with respect to different values of n_0. Therefore, when implementing the algorithm in practice, the above analysis must be complemented with a number of experiments aiming at determining the best switching point empirically.

Exercise 3.1 Let n be a power of 2. Consider an algorithm A_1 solving a computational problem Π in time $T_1(n) = n^2$. Suppose that we can devise, for the same problem, a divide-and-conquer algorithm A_2 that, for $n > 1$, generates two instances of Π of size $n/2$, with divide and conquer phases requiring time $w(n) = 8n$.

(a) Evaluate the running time $T_2(n)$ of algorithm A_2 under the assumption that $T_2(1) = 0$.

(b) Consider the hybrid algorithm H obtained from A_2 and A_1, and let n_0 be the switching point between the two algorithms. Determine the running time $T(n, n_0)$ of H.

(c) Find the value of n_0 that minimizes $T(n, n_0)$ What is the running time of the resulting hybrid algorithm? (*Hint:* Pay attention with the derivative of $\log_2 n_0$.)