Divide-and-Conquer Algorithms

To solve a problem of size n:

divide the problem into subproblems, each of size $< n$,

conquer: solve each subproblem recursively (and independently of the other subproblems)

combine/merge the solutions to the subproblems into a solution to the original problem.

When applying this technique, our task is:

* how do we divide the problem; into how many subproblems?
* how to combine/merge?

Example: Merge-Sort

To sort n numbers:

if $n \le 1$: nothing to do

if $n \ge 2$: divide the n numbers arbitrarily into 2 sequences, both of size $n/2$;
run Merge-Sort twice, once for each sequence;
merge the two sorted sequences into one sorted sequence.

What is the running time?

Define

$T(n)$ = running time of Merge-Sort on an input of n numbers.

From 2402: merge step takes $O(n)$ time.

and 2804

$$T(n) \leq \begin{cases} c & \text{if } n=1, \\ cn + 2\cdot T(\frac{n}{2}) & \text{if } n \geq 2, \end{cases}$$

for some constant $c > 0$.

Solve this recurrence by <u>unfolding</u>:

Assume $n = 2^k$, $c = 1$.

$$\begin{aligned} T(n) &\leq n + 2\cdot T(\tfrac{n}{2}) \\ &\leq n + 2\cdot\left[\tfrac{n}{2} + 2\cdot T(\tfrac{n}{4})\right] \\ &= 2n + 4\cdot T(\tfrac{n}{4}) \\ &\leq 2n + 4\left[\tfrac{n}{4} + 2\cdot T(\tfrac{n}{8})\right] \\ &= 3n + 8\cdot T(\tfrac{n}{8}) \\ &\leq 3n + 8\left[\tfrac{n}{8} + 2\cdot T(\tfrac{n}{16})\right] \\ &= 4n + 16\cdot T(\tfrac{n}{16}) \end{aligned}$$

$$\leq \ldots$$

$$\leq kn + 2^k \cdot T\left(\frac{n}{2^k}\right)$$

$$= kn + n \cdot T(1)$$

$$= kn + n$$

$$= n \log n + n$$

$$\leq 2n \log n.$$

For general $c > 0$: $T(n) \leq 2cn \log n$.

$\therefore$ Running time of Merge-Sort is $O(n \log n)$ (if n is a power of 2).

For general n :

$$T(n) \leq \begin{cases} c & \text{if } n = 1 \\ cn + T\left(\lfloor \frac{n}{2} \rfloor\right) + T\left(\lceil \frac{n}{2} \rceil\right) & \text{if } n \geq 2. \end{cases}$$

By induction: $T(n) = O(n \log n)$.

Multiplying Integers

Input: n-bit integers x and y.

Output: Product xy.

School method: $O(n^2)$ bit-operations.

Can we do better? Yes, using divide-and-conquer.

Assume n is a power of 2.

$$x = \overbrace{\boxed{\quad x_L \quad}}^{\leftarrow \frac{n}{2} \rightarrow}\overbrace{\boxed{\quad x_R \quad}}^{\leftarrow \frac{n}{2} \rightarrow} = 2^{n/2} x_L + x_R$$

$$y = \boxed{\quad y_L \quad}\boxed{\quad y_R \quad} = 2^{n/2} y_L + y_R$$

$$xy = 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R$$

To compute xy:

* recursively compute $x_L y_L$, $x_L y_R$, $x_R y_L$, and $x_R y_R$
* "multiply" $x_L y_L$ by 2^n: add n many 0's at the end: $O(n)$ time
* compute $x_L y_R + x_R y_L$ using one addition: $O(n)$ time;

  "multiply" by $2^{n/2}$: $O(n)$ time
* two more additions give us xy: $O(n)$ time.

Define

$T(n)$ = # bit-operations to multiply two n-bit integers.

$$T(n) \leq \begin{cases} 1 & \text{if } n=1, \\ cn + 4 \cdot T\left(\frac{n}{2}\right) & \text{if } n \geq 2. \end{cases}$$

Assume $n = 2^k$, $c = 1$.

Unfold:

$$T(n) \leq n + 4 \cdot T\left(\frac{n}{2}\right)$$

$$\leq n + 4\left[\frac{n}{2} + 4 \cdot T\left(\frac{n}{4}\right)\right]$$

$$= (1+2)n + 4^2 \cdot T\left(\frac{n}{4}\right)$$

$$\leq (1+2)n + 4^2\left[\frac{n}{4} + 4 \cdot T\left(\frac{n}{8}\right)\right]$$

$$= (1+2+4)n + 4^3 \cdot T\left(\frac{n}{2^3}\right)$$

$$\leq (1+2+4)n + 4^3\left[\frac{n}{2^3} + 4 \cdot T\left(\frac{n}{2^4}\right)\right]$$

$$= (1+2+4+8)n + 4^4 \cdot T\left(\frac{n}{2^4}\right)$$

$$= (1+2+2^2+2^3)n + 4^4 \cdot T\left(\frac{n}{2^4}\right)$$

$$\leq (1+2+2^2+2^3+2^4)n + 4^5 \cdot T\left(\frac{n}{2^5}\right)$$

$$\leq \dots$$

$$\leq \underbrace{(1+2+2^2+\dots+2^{k-1})}_{2^k-1 \,=\, n-1}n + \underbrace{4^k}_{n^2} \cdot \underbrace{T\left(\frac{n}{2^k}\right)}_{T(1)=1}$$

$$= (n-1)n + n^2$$

$$\leq 2n^2$$

$\therefore T(n) = O(n^2)$ $\therefore$ no improvement!

Why is the running time $O(n^2)$:

To multiply two n-bit integers:

* 4 multiplications of $\frac{n}{2}$-bit integers ← expensive

* $O(n)$ extra work ← cheap

Karatsuba (1960):

* replace 4 by 3
* do a bit more extra work, but still $O(n)$.

$$xy = 2^{n} x_L y_L + 2^{\frac{n}{2}} \left[(x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R \right] + x_R y_R$$

To compute xy :

* recursively compute $x_L y_L$, $x_R y_R$, and
  $(x_L + x_R)(y_L + y_R)$ [3 recursive calls]
* combine all results in $O(n)$ time.

The running time $T(n)$ satisfies:

$$T(n) \leq \begin{cases} 1 & \text{if } n = 1, \\ cn + 3 \cdot T\left(\frac{n}{2}\right) & \text{if } n \geq 2. \end{cases}$$

Assume $n = 2^k$, $c = 1$.

Unfolding, as on pages 18 – 19, gives

$$T(n) \leq \left[1 + \frac{3}{2} + \left(\frac{3}{2}\right)^2 + \ldots + \left(\frac{3}{2}\right)^{k-1}\right] n$$

$$+ \ 3^k \cdot \underbrace{T\left(\frac{n}{2^k}\right)}_{= T(1) = 1}$$

Recall:

$$\boxed{1 + x + x^2 + \dots + x^{k-1} = \frac{x^k - 1}{x - 1} \quad \text{for } x \neq 1}$$

$$T(n) \leq \frac{\left(\frac{3}{2}\right)^k - 1}{\frac{3}{2} - 1} \cdot n + 3^k$$

$$= 2\left[\left(\frac{3}{2}\right)^k - 1\right] \cdot n + 3^k$$

$$\leq 2 \cdot \frac{3^k}{2^k} \cdot n + 3^k \qquad [n = 2^k]$$

$$= 3 \cdot 3^k \qquad \boxed{\text{Recall: } x = 2^{\log x},\ x > 0}$$

$$3^k = 2^{k \log 3} = \left(2^k\right)^{\log 3} = n^{\log 3}$$

$$\therefore\ T(n) \leq 3 \cdot n^{\log 3} = O\left(n^{\log 3}\right)$$

$$\log 3 \approx 1.58$$