3.1 Integer multiplication

Input: Two positive \( n \) digit integers, \( a \) and \( b \).
Output: \( a \times b \)

The naive approach is to simply multiply the two numbers which takes \( O(n^2) \) time because we need to multiply each digit in one number with those in the second number, put them in the correct position and add them as shown below.

\[
\begin{array}{c}
3 & 8 & 4 \\
\times & 5 & 6 \\
\hline
2 & 3 & 0 & 4 \\
1 & 9 & 2 & 0 \\
\hline
2 & 1 & 5 & 0 & 4 \\
\end{array}
\]

Can we do better?

Suppose we are given \( a = 123456 \) and \( b = 654321 \). We can rewrite \( a \) as \( 123000 + 456 \) and \( b \) as \( 654000 + 321 \). As a result, \( a \times b = 123 \times 654 \times 10^6 + (123 \times 321 + 456 \times 654) \times 10^3 + 456 \times 321 \).

Assume \( n \) is a power of 2. We partition \( a \) and \( b \) into lower and upper digits as \( a = a_1 \times 10^{n/2} + a_2 \) and \( b = b_1 \times 10^{n/2} + b_2 \). Thus, the product becomes \( A \times 10^n + (B + C) \times 10^3 + D \), where \( A = a_1 \times b_1, B = a_2 \times b_1, C = a_1 \times b_2 \) and \( D = a_2 \times b_2 \).

Recursion: Let \( T(n) \) be the running time to multiply two \( n \)-digit numbers, \( a \) and \( b \).

\[
Multiply(a, b):
\]

1. WLOG assume \( n = length(a) = length(b) \), can pad 0’s for shorter number
2. if \( length(a) <= 1 \) then return \( a \times b \)
3. Partition \( a, b \) into \( a = a_1 \times 10^{n/2} + a_2 \) and \( b = b_1 \times 10^{n/2} + b_2 \)
4. \( A = Multiply(a1, b1) \)
5. \( B = Multiply(a2, b1) \)
6. \( C = Multiply(a1, b2) \)
7. \( D = Multiply(a2, b2) \)
8. Return \( A \times 10^n + (B + C) \times 10^{n/2} + D \)

Thus,

\[
T(n) = 4T\left(\frac{n}{2}\right) + O(n)
\]
where $O(n)$ accounts for partitioning the given numbers, the addition operation(s) and shifting/padding. We call this the ‘merge’ time.

$$T(n) = 4T(n/2) + A \cdot n$$

(A*n indicates the merging cost at layer 0)

$$= 16T(n/4) + 4 \cdot A \cdot n/2 + A \cdot n$$

$$= 64T(n/8) + 16 \cdot A \cdot n/4 + 4 \cdot A \cdot n/2 + A \cdot n$$

We can assume $T(1) = 1$ since we are doing asymptotic analysis. Overall cost of the function is the sum of the merging cost of all layers. The number of layers $l = \log_2 n$.

$$T(n) = \sum_{i=0}^{\log_2 n} 4^i A \frac{n}{2^i}$$

$$= A n \sum_{i=0}^{\log_2 n} 2^i$$

$$= A n (2n - 1) = O(n^2)$$

### 3.2 Improved algorithm

We can improve the algorithm by doing one of the following:

1. Merging faster: However, this is not the bottleneck for integer multiplication. $O(n)$ is not large.
2. Make subproblems smaller: If we do this naively, then that would result in more number of subproblems which defeats the purpose.
3. Decrease the number of subproblem: We see the details below.

**Multiply(a, b):**

1. WLOG assume $n = length(a) = length(b)$, can pad 0’s for shorter number
2. if \( \text{length}(a) \leq 1 \) then return \( a \cdot b \)
3. Partition \( a, b \) into \( a = a_1 \cdot 10^{n/2} + a_2 \) and \( b = b_1 \cdot 10^{n/2} + b_2 \)
4. \( A = \text{Multiply}(a_1, b_1) \)
5. \( B = \text{Multiply}(a_2, b_2) \)
6. \( C = \text{Multiply}(a_1 + a_1, b_1 + b_2) \)
7. Return \( A \cdot 10^n + (C - A - B) \cdot 10^{n/2} + B \)

Thus,

\[
T(n) = 3T\left(\frac{n}{2}\right) + O(n)
\]

\[
T(n) = \sum_{i=0}^{\log_2 n} 3^i A \frac{n}{2^i}
\]

\[
= An \sum_{i=0}^{\log_2 n} \left(\frac{3}{2}\right)^i
\]

\[
= An \frac{(3/2)^{\log_2 n+1} - 1}{3/2 - 1}
\]

\[
= O\left(n^{3\log_2 n/2}\right)
\]

\[
= O\left(n \cdot n^{\log_2 3/2}\right)
\]

\[
= O\left(n^{\log_2 3}\right)
\]

\[
= O\left(n^{1.585}\right)
\]

Even faster algorithms use fast Fourier analysis, which is beyond the scope of this class.

### 3.3 Master theorem

The Master Theorem acts as a "cheat sheet" for basic recursions. Given \( T(n) = aT\left(\frac{n}{b}\right) + f(n) \):
1. If \( f(n) = O(n^c) \), \( c < \log_b a \) then \( T(n) = \Theta(n^{\log_b a}) \)

2. If \( f(n) = \Theta(n^c \log^t(n)) \), \( c = \log_b a \) then \( T(n) = \Theta(n^{\log_b a \log^{t+1}(n)}) \)

3. If \( f(n) = \Theta(n^c) \), \( c > \log_b a \) then \( T(n) = \Theta(n^c) \)

**Case 1:** Merge cost is dominated by the cost of the last layer.

- \( l := \) number of layers, and equals \( \log_b n \).
- Number of nodes in layer \( l \) equals \( a^l \), and the merge cost for this last layer equals \( n^{\log_b a} \).

Example. When \( a = 4, b = 2 \) and \( f(n) = n \): as seen for the first algorithm for integer multiplication, we get \( O(n^{\log_2 4}) \).

When \( a = 3, b = 2 \) and \( f(n) = n \): as seen for the second algorithm for integer multiplication, we get \( O(n^{\log_2 3}) \).

**Case 2:** Additional log factor shows up in the overall runtime because of the height of the recursion tree, i.e., the number of layers.

Example. \( a = 2, b = 2 \) and \( f(n) = n \): applies to mergesort and counting inversions which we saw in a previous lecture. Here, we get \( O(n \log n) \).

**Case 3:** Merge cost is dominated by the cost of the first layer.

Cannot improve further by reducing \( a \). Can instead try to improve the merge cost from something smaller than \( n^c \).

Example. Running time for the first attempt to counting the number of inversions. There, we had \( a = 2, b = 2 \) and \( f(n) = n^2 \), which gave an overall runtime of \( O(n^2) \).