Karatsuba Integer Multiplication

CS5800 Algorithms Lecture Notes - Consolidated with Codex

Multiplying long integers seems to require every digit of one number to interact with every digit of the other. Karatsuba shows how an algebraic rearrangement reduces the number of recursive products. Saving one product at each call changes the exponent of the running time.

Roadmap. The cost model; splitting integers; three products instead of four; worked examples; recursive pseudocode and correctness; recursion trees, the Master Theorem, and induction; implementation details and course connections.

1 What Is the Size of a Multiplication Problem?

1.1 Count digits, not the numerical value

The input consists of two nonnegative integers x and y, each represented using at most n digits in a fixed base $\beta \geq 2$. A shorter representation may be padded on the left with zeros. Thus

$$0 \leq x, y < \beta^n, \qquad 0 \leq xy < \beta^{2n}.$$

The product fits in at most $2n$ digits. We use decimal examples for readability; the same algorithm works with binary digits or fixed-size machine words.

Cost model. Accessing one digit and adding or multiplying a constant number of digits take $O(1)$ time. Addition, subtraction, and copying of $O(n)$ digits take $O(n)$ time, including carries or borrows. Multiplication by β^k is a digit shift; materializing an $O(n)$-digit shifted result costs at most $O(n)$.

An arbitrary n-digit multiplication is the problem we are solving, so it is *not* counted as a constant-time operation. Multiplying two fixed-width machine integers is a different cost model from multiplying integers whose representations grow with n.

1.2 The grade-school baseline

Ordinary multiplication forms a product for each pair of input digits. There are n^2 such pairs. The partial products can be accumulated with carry handling in $\Theta(n^2)$ digit operations.

A four-digit by four-digit multiplication forms 16 single-digit products. To improve this count, we must change which smaller products are computed.

Why not repeated addition? Adding x to itself y times can take exponentially many additions in the digit length of y. We measure input length, not input value.

Key point: Karatsuba uses three multiplications of approximately half-length operands, plus linear work. This leads to $\Theta(n^{\log_2 3}) \approx \Theta(n^{1.585})$ digit operations.

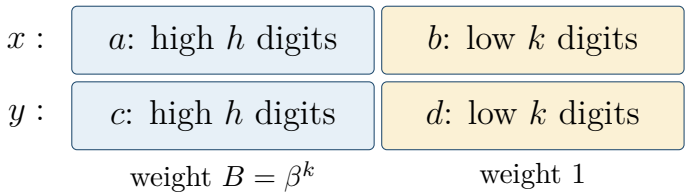
2 Split the Operands, Then Change the Algebra

2.1 High and low blocks

Choose a split width k and let $B = \beta^k$. Write

$$x = aB + b, \qquad y = cB + d, \qquad 0 \le b, d < B.$$

Here b, d hold the low k digits, and a, c hold the remaining high digits. For balanced recursion, take $k = \lfloor n/2 \rfloor$ and $h = n - k = \lceil n/2 \rceil$.



Expanding the product gives

$$xy = acB^2 + (ad + bc)B + bd.$$

Computing ac, ad, bc, and bd recursively uses four smaller products. Ignoring rounding temporarily, its recurrence is

$$T_4(n) = 4T_4(n/2) + \Theta(n) = \Theta(n^2).$$

The split alone has not improved the asymptotic cost.

2.2 Compute the cross-term sum without its separate products

We need $ad + bc$ in the final result. We do not need to obtain ad and bc separately. Compute

$$u = ac, \qquad v = bd, \qquad w = (a + b)(c + d).$$

Because $w = ac + ad + bc + bd$, the middle coefficient is

$$z = w - u - v = ad + bc.$$

Therefore

$$\boxed{xy = uB^2 + (w - u - v)B + v.}$$

Only u, v, w require recursive multiplication. Forming the sums, recovering z, and placing the results at the correct digit offsets require a constant number of operations on $O(n)$-digit values.

Key point: The improvement comes from eliminating a recursive multiplication at *every* non-base call. A few extra additions are worthwhile because each avoided product would itself generate a whole subtree of work.

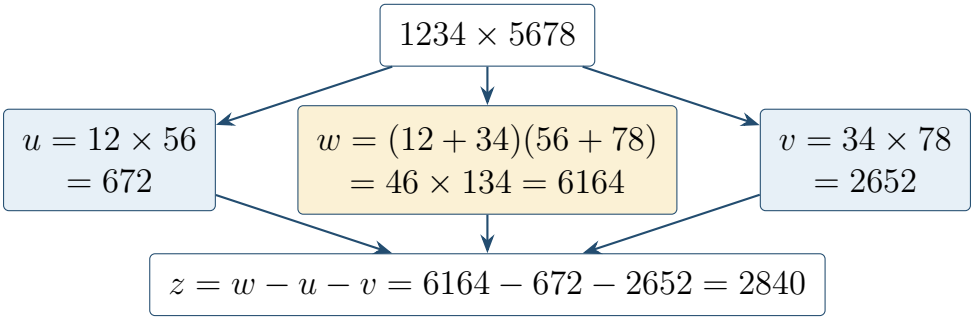
3 Worked Example: From Three Products to the Answer

3.1 Multiply 1234 by 5678

With $\beta = 10$, $k = 2$, and $B = 100$,

$$1234 = 12 \cdot 100 + 34, \qquad 5678 = 56 \cdot 100 + 78.$$

The three colored boxes below represent multiplication calls. On longer inputs, these calls apply the same method recursively.



Recombine using the block weights:

$$\begin{aligned} 1234 \cdot 5678 &= 672 \cdot 100^2 + 2840 \cdot 100 + 2652 \\ &= 6{,}720{,}000 + 284{,}000 + 2652 \\ &= \boxed{7{,}006{,}652}. \end{aligned}$$

As an algebra check, the omitted cross products would have been $12 \cdot 78 = 936$ and $34 \cdot 56 = 1904$, whose sum is indeed 2840. The algorithm does not compute these two products.

3.2 Recombination is addition with carries

The coefficients u, z, v are not individual base-100 digits: $z = 2840$ and $v = 2652$ exceed 99. Do not concatenate the strings 672, 2840, and 2652.

To see the carries in blocks, first write $2652 = 26 \cdot 100 + 52$. The middle coefficient becomes $2840 + 26 = 2866 = 28 \cdot 100 + 66$. The high coefficient becomes $672 + 28 = 700$. Thus

$$xy = 700 \cdot 100^2 + 66 \cdot 100 + 52 = 7{,}006{,}652.$$

The shifted additions in the formula automatically perform this carry propagation. A digit-array implementation must preserve it explicitly.

4 A Recursive Algorithm with Explicit Sizes

We retain an allocated width n, allowing leading zeros. Keeping widths explicit makes the recurrence independent of whether a particular block happens to be small. Let the fixed base β be understood by every call.

Algorithm: Karatsuba(x, y, n)

Precondition: $n \geq 1$ *and* $0 \leq x, y < \beta^n$. *Return the exact product.*

1: **if** $n \leq 3$ **then**
2: **return** grade-school product of x and y
3: **end if**
4: $k \leftarrow \lfloor n/2 \rfloor$; $h \leftarrow n - k$; $B \leftarrow \beta^k$
5: $a \leftarrow \lfloor x/B \rfloor$; $b \leftarrow x \bmod B$ ▷ split digit blocks
6: $c \leftarrow \lfloor y/B \rfloor$; $d \leftarrow y \bmod B$
7: $u \leftarrow \textsc{Karatsuba}(a, c, h)$
8: $v \leftarrow \textsc{Karatsuba}(b, d, k)$
9: $w \leftarrow \textsc{Karatsuba}(a + b, c + d, h + 1)$
10: $z \leftarrow w - u - v$
11: **return** $u\beta^{2k} + z\beta^k + v$ ▷ shift and add, including carries

The divisions and remainders by B mean splitting at a digit boundary, not invoking general long division. Likewise, β^k indicates a digit position, not a general exponentiation routine.

4.1 Why the third call has one extra digit

Each high or low block is less than β^h, so

$$a + b < 2\beta^h \leq \beta^{h+1}, \qquad c + d < 2\beta^h \leq \beta^{h+1}.$$

The sum operands therefore fit in $h + 1$ digits. In the worked example, $56 + 78 = 134$: two-digit blocks can produce a three-digit sum. Dropping the leading carry would change the answer.

The cutoff $n \leq 3$ ensures strict size reduction: for $n \geq 4$, even the largest child width $\lceil n/2 \rceil + 1$ is less than n. Any larger fixed cutoff is also valid.

4.2 Odd lengths and unequal lengths

For $x = 12345$, $y = 6789$, pad the second operand as 06789. With $n = 5$ and $k = 2$,

$$x = 123 \cdot 100 + 45, \qquad y = 67 \cdot 100 + 89.$$

Then $u = 8241$, $v = 4005$, $w = 168 \cdot 156 = 26208$, and $z = 13962$, giving

$$xy = 8241 \cdot 10000 + 13962 \cdot 100 + 4005 = 83{,}810{,}205.$$

Both operands must use the *same* split width. The high block may have one more digit than the low block; the formula remains exact.

5 Correctness and an Alternative Identity

5.1 Correctness by strong induction on the allocated width

Base cases. For $n \leq 3$, direct multiplication returns the correct product.

Induction hypothesis. Assume every call of width smaller than n returns the product of its two operands.

Inductive step. For $n \geq 4$, splitting gives $x = aB + b$ and $y = cB + d$ exactly. The high blocks fit in h digits, the low blocks in k digits, and the sums in $h + 1$ digits. All three widths are smaller than n, so the induction hypothesis gives

$$u = ac, \qquad v = bd, \qquad w = (a + b)(c + d).$$

The algorithm returns

$$\begin{aligned} uB^2 + (w - u - v)B + v &= acB^2 + (ad + bc)B + bd \\ &= (aB + b)(cB + d) = xy. \end{aligned}$$

This establishes correctness and, together with the strict reduction in width, termination. Leading zeros do not affect the argument.

5.2 Using differences instead of sums

The identity can also be written with

$$t = (a - b)(c - d), \qquad z = u + v - t.$$

Indeed, $t = ac - ad - bc + bd$. For equally sized high and low blocks, the magnitudes $|a - b|$ and $|c - d|$ fit in the original half-width. This avoids the extra magnitude digit introduced by sums, but requires handling signs. GMP describes this version in its implementation notes [4].

For the main example, $a - b = -22$ and $c - d = -22$. Their product is $t = 484$, so

$$z = 672 + 2652 - 484 = 2840.$$

If the differences have opposite signs, their product is negative. For example, in $3412 \cdot 5678$, $a - b = 22$ and $c - d = -22$, giving

$$u = 1904, \qquad v = 936, \qquad t = -484, \qquad z = 1904 + 936 - (-484) = 3324.$$

An unsigned recursive multiplication can compute $|a - b|\,|c - d|$; separately record the sign and then add or subtract the magnitude as appropriate.

Key point: The sum and difference versions are two ways to recover the same cross-term sum. Our main pseudocode uses sums for a direct derivation; its analysis explicitly accounts for the possible extra digit.

6 The Recurrence and Its Recursion Tree

6.1 First isolate the central recurrence

Ignore rounding and the possible extra digit for a moment. Each call makes three half-size calls and performs $\Theta(n)$ additional work:

$$T(n) = 3T(n/2) + \Theta(n).$$

The linear term includes splitting, additions, subtractions, shifted additions, and carry propagation. The multiplications of smaller *operands* belong inside the recursive terms.

Master Theorem. Here the number of calls is $a = 3$, the shrink factor is $b = 2$, and $f(n) = \Theta(n)$. Since

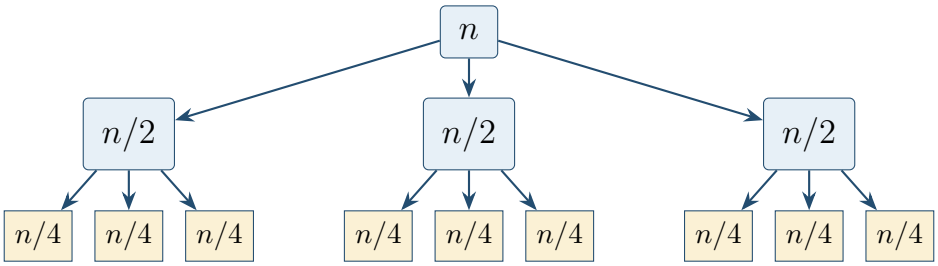
$$n^{\log_b a} = n^{\log_2 3}, \qquad 1 < \log_2 3 \approx 1.585,$$

the nonrecursive work is polynomially smaller than $n^{\log_b a}$. Case 1 gives

$$\boxed{T(n) = \Theta(n^{\log_2 3}).}$$

Section 8 justifies this bound for the actual child widths in the pseudocode.

6.2 Why the work grows toward the bottom



Nodes show operand widths, not the numerical values of the operands.

Level	Number of calls	Width per call	Total nonrecursive work
0	1	n	cn
1	3	$n/2$	$cn(3/2)$
2	9	$n/4$	$cn(3/2)^2$
j	3^j	$n/2^j$	$cn(3/2)^j$

Each call is cheaper at the next level, but there are three times as many calls and each is only half as large. The total per level increases by a factor of 3/2.

7 Solve the Recurrence by Iteration and Induction

7.1 Iteration includes both internal work and leaves

For the simplified recurrence $T(n) = 3T(n/2) + cn$, assume $n = 2^L$ and $T(1) = t_0 > 0$. After j expansions,

$$T(n) = 3^j T(n/2^j) + cn \sum_{r=0}^{j-1} (3/2)^r.$$

At $j = L = \log_2 n$, every leaf has width 1, so

$$\begin{aligned} T(n) &= 3^L t_0 + cn \frac{(3/2)^L - 1}{(3/2) - 1} \\ &= t_0 n^{\log_2 3} + 2c\Big(n^{\log_2 3} - n\Big) \\ &= (t_0 + 2c) n^{\log_2 3} - 2cn. \end{aligned}$$

Both the leaf term and the sum of internal work are $\Theta(n^{\log_2 3})$. The additions are not free, even though including them leaves the exponent unchanged.

7.2 Why a direct induction guess needs some slack

Write $\alpha = \log_2 3$. An attempt to prove $T(n) \le Cn^\alpha$ by substituting the same bound for each child yields

$$T(n) \le 3C(n/2)^\alpha + cn = Cn^\alpha + cn.$$

This does not finish the proof: the positive linear term has nowhere to go. The theorem may be true even when this induction hypothesis is too weak.

Strengthen the target to

$$T(n) \le Cn^\alpha - dn$$

for a positive constant d. Then the induction step gives

$$\begin{aligned} T(n) &\le 3\Big(C(n/2)^\alpha - dn/2\Big) + cn \\ &= Cn^\alpha - (3d/2 - c)n \\ &\le Cn^\alpha - dn \qquad \text{when } d \ge 2c. \end{aligned}$$

Choose $C \ge t_0 + d$ to cover the base case. For the lower bound, simply count $3^L = n^\alpha$ leaves, each costing t_0. This proves the matching Θ bound for the simplified recurrence.

7.3 Compare with the four-product method

The analogous four-product tree has $4^L = n^2$ leaves and total internal work

$$cn \sum_{r=0}^{L-1} 2^r = \Theta(n^2).$$

Saving one of four calls is not merely a one-time 25% improvement. It replaces 4^L by 3^L across L recursive levels, changing the power of n.

8 Rounding and Carries Do Not Change the Exponent

This section justifies the half-size simplification used in the preceding analysis.

8.1 The actual child widths

For the padded-width pseudocode, $h = \lceil n/2 \rceil$ and $k = \lfloor n/2 \rfloor$. With standard linear-cost digit-array operations, its recurrence is

$$T(n) = T(h) + T(k) + T(h+1) + \Theta(n), \qquad n \geq 4.$$

The third term allows for a carry. Even with power-of-two input lengths, sums inside later calls may gain a digit.

8.2 Upper bound: follow the largest possible width

Along any path, the next width satisfies

$$s_{j+1} \leq \lceil s_j/2 \rceil + 1 \leq s_j/2 + 3/2.$$

Unrolling this inequality from $s_0 = n$ gives

$$s_j \leq \frac{n}{2^j} + 3\left(1 - \frac{1}{2^j}\right) = \frac{n-3}{2^j} + 3.$$

At depth $H = \lceil \log_2 n \rceil$, this bound is strictly less than 4. Since widths are integers, every remaining call is a base case by that depth.

There are at most 3^j calls at depth j. Charging $O(s_j)$ per call, including a constant for a base case, bounds the total work by

$$\begin{aligned} O\left(\sum_{j=0}^{H} 3^j (n/2^j + 3)\right) &= O\big(n(3/2)^H + 3^H\big) \\ &= O(n^{\log_2 3}). \end{aligned}$$

8.3 Lower bound for this padded implementation

Every child has width at least $\lfloor s/2 \rfloor$ when its parent has width s. Thus, while full levels exist, every width at depth j is at least $\lfloor n/2^j \rfloor$.

For $n \geq 4$, put $D = \lfloor \log_2(n/4) \rfloor$. At every level through D, this minimum is at least 4, so no branch has stopped before that level. There are $3^D = \Omega(n^{\log_2 3})$ calls there. Each costs at least a constant, proving the matching lower bound.

Key point: The displayed pseudocode has $\Theta(n^{\log_2 3})$ cost in the padded digit model. The same upper bound holds if an implementation skips zero products or removes leading zeros; particular inputs may then finish sooner.

9 Implementation and Broader Connections

9.1 Use a fixed cutoff and account for temporary storage

At small widths, extra additions, allocations, and function calls can outweigh the saved multiplication. A practical implementation switches to grade-school multiplication below a tuned cutoff. GMP's documentation discusses this tradeoff [4]; there is no universal crossover size independent of representation and implementation.

The cutoff changes constants, not the exponent, when it is fixed as n grows. It should also guarantee that recursive widths decrease.

Space. Execute the three calls sequentially and reclaim their temporary workspaces after each returns. A width-n call needs $O(n)$ local digits, including partial products retained while computing the next one. Along one active recursion path, widths are approximately $n, n/2, n/4, \ldots$, with bounded rounding corrections. Their sum is $O(n)$.

Thus this implementation can use $O(n)$ auxiliary digit storage and $O(\log n)$ stack frames. Keeping every node's temporary arrays alive throughout the entire recursion tree would use more space; a time recurrence alone does not specify memory usage.

9.2 Signs, zeros, and unbalanced inputs

For signed integers, multiply the magnitudes and restore the sign at the end. The sum-form core therefore needs only nonnegative operands. Zero inputs and leading zeros are already handled by the pseudocode; an early return for zero is an optional optimization.

If one operand is much shorter, padding both to the larger width remains correct but can waste work. For example, multiplying a one-digit value by an n-digit value can be done in $O(n)$ time. Specialized handling of strongly unequal lengths is outside our balanced-input analysis.

9.3 A polynomial interpretation of the three products

Introduce two linear polynomials

$$f(t) = at + b, \qquad g(t) = ct + d.$$

Their product has three coefficients:

$$f(t)g(t) = ut^2 + zt + v.$$

The constant coefficient is $v = bd$, the leading coefficient is $u = ac$, and evaluating at $t = 1$ gives $w = (a + b)(c + d) = u + z + v$. These three pieces of information determine the middle coefficient. Finally evaluate at $t = B$ to recover the integer product.

This viewpoint also extends to multiplying longer polynomials by splitting their coefficient arrays. In that setting, coefficient-operation counts require a specified cost model for the coefficients; growing integer coefficients cannot automatically be treated as constant-time values.

10 Course Placement, Practice, and Reading

10.1 Recommended home: Module 2, Recurrences

Karatsuba belongs with divide-and-conquer, recursion trees, the Master Theorem, and substitution proofs. A useful teaching sequence is: derive the four-product recurrence, find the three-product identity, solve the new recurrence, and then discuss carries and rounded sizes.

The course's earlier Strassen example offers a useful comparison: seven half-size matrix products and quadratic combination work give $T(n) = 7T(n/2) + \Theta(n^2) = \Theta(n^{\log_2 7})$. Both methods reduce the number of recursive products, with different combination costs [3].

Why not dynamic programming? The algorithm is faster because an algebraic identity removes one recursive product. It does not rely on storing solutions to repeated subproblems in a memoization table. Retaining u and v for recombination is ordinary local computation, rather than the repeated-state reuse that motivates DP. Matrix-chain multiplication in the DP module chooses a parenthesization; Karatsuba changes how a product itself is computed.

10.2 Practice without solutions

1. Trace one Karatsuba step for $2468 \cdot 1357$. Show the three products, the middle coefficient, and the shifted additions.
2. Split $9999 \cdot 9999$ after the low two digits. Which operands need an extra digit? Explain why concatenating the partial results fails.
3. Suppose an alternative identity required five half-size products and linear combination work. Derive and solve its recurrence. Would recursion improve on grade-school multiplication?
4. Explain why using the Karatsuba identity only at the root, followed by grade-school multiplication in all children, still has quadratic running time.

10.3 Sources and reading guide

These notes develop the examples and proofs in a consistent digit-cost model. The references offer complementary presentations.

[1] Wikipedia: Karatsuba algorithm. Algebra, variants, and the $12345 \cdot 6789$ example. Wikipedia splits that example at $B = 1000$; our odd-width example uses $B = 100$, with the same product.

[2] Stanford CS161, Lecture 1: Karatsuba Integer Multiplication. Derives the improvement from four products to three and flags the extra digit in sum operands.

[3] Carnegie Mellon 15-451, Lecture 1: Introduction to Algorithms. Places Karatsuba alongside recurrence analysis and Strassen multiplication.

[4] GNU MP manual: Karatsuba Multiplication. Difference-based formulation, sign handling, and the practical cutoff tradeoff.