Recursion tree method

Solve \( T(n) = 2 \cdot T(n/2) + \Theta(n) \)
Recursion tree method – TopHat Question 2

Solve \( T(n) = 2 \cdot T(n/2) + \Theta(n) \)

Question. How many layers?
A. constant
B. \( \log_2 n \)
C. \( \log_4 n \)
D. \( n \)
E. \( n^2 \)
Recursion tree method – TopHat Question 3

Solve \( T(n) = 2 \cdot T(n/2) + \Theta(n) \)

Question. How many leaves?
A. constant
B. \( \log_2 n \)
C. \( \log_4 n \)
D. \( n \)
E. \( n^2 \)
Recursion tree method

Solve \( T(n) = 2 \cdot T(n/2) + \Theta(n) \)

\[
\begin{align*}
T(n) &= T(n/2) + T(n/2) + \cdots + T(n/2) + T(n/2) \\
&= \sum_{i=0}^{\log_2 n} T(n/2^i)
\end{align*}
\]

inputs of length \( n \)

inputs of length \( n/2 \)

inputs of length \( n/4 \)

base case: input \( n=1 \)

#leaves = \( n \)
Recursion tree method — analysis idea

- charge each operation to the function call (i.e. tree node) at which it is executed
- for Mergesort on a subarray of length \( k \) we do \( O(k) \) work to compute the split point and do the merge
- work in further recursive calls is counted separately

\[
T(n) = 2 \cdot T(n/2) + cn \text{ where } c > 0
\]

use \( cn \) instead of \( \Theta(n) \)
Recursion tree method — analysis idea

- charge each operation to the function call (i.e. tree node) at which it is executed
- for Mergesort on a subarray of length \( k \) we do \( O(k) \) work to compute the split point and do the merge
- work in further recursive calls is counted separately

\[
T(n) = 2 \cdot T(n/2) + cn \quad \text{where} \quad c > 0
\]
Recursion tree method — analysis idea

- charge each operation to the function call (i.e. tree node) at which it is executed
- for Mergesort on a subarray of length $k$ we do $O(k)$ work to compute the split point and do the merge
- work in further recursive calls is counted separately

$$T(n) = 2 \cdot T(n/2) + cn \text{ where } c > 0$$

use $cn$ instead of $\Theta(n)$

<table>
<thead>
<tr>
<th>Level</th>
<th>Work per Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$c \cdot n$</td>
</tr>
<tr>
<td>2</td>
<td>$2 \cdot c \cdot n/2$</td>
</tr>
<tr>
<td>3</td>
<td>$4 \cdot c \cdot n/4$</td>
</tr>
<tr>
<td>4</td>
<td>$2^k \cdot c \cdot n/2^k$</td>
</tr>
<tr>
<td>$k$</td>
<td>$\Theta(n/2^k)$</td>
</tr>
</tbody>
</table>

Total: $\Theta(n \log n)$
binary search

Input: sorted array A, integer x
Output: the index in A where the int x is located or “x is not in A”

1. Divide: compare x to middle element
2. Conquer: recursively search one subarray
3. Combine: trivial

example: find 16
Algorithm 1: BinarySearch( sorted array A, int p, int r, query )

/* find the index of query in A (if present) */

1  q ← \[ \frac{p+r}{2} \];
2  middle ← A[q];
3  if query == middle then
4      return q;
5  if query < middle then
6      BinarySearch(A, p, q, query);
7  else
8      BinarySearch(A, p, q, query);
9  return q not in A;
Algorithm 1: BinarySearch( sorted array \( A \), \( p \), \( r \), query )

/* find the index of query in \( A \) (if present) */

1. \( q \leftarrow \lfloor \frac{p+r}{2} \rfloor \);
2. middle \( \leftarrow A[q] \);
3. if \( \text{query} == \text{middle} \) then
   4. \( \text{return } q \);
5. if \( \text{query} < \text{middle} \) then
   6. \( \text{BinarySearch}(A, p, q, \text{query}) \);
   7. else
   8. \( \text{BinarySearch}(A, p, q, \text{query}) \);
9. \( \text{return } q \) not in \( A \);

\[ T(n) = 1 \cdot T(n/2) + \Theta(1) \]

- # of subproblems
- subproblem size
- work dividing and combining

O(c)

recursive call on array half the size
Reminder: our goal is to find a closed form formula for the recurrence method:

• substitute the recursive formula until we get a good guess
• use induction to prove the formula

\[
T(n) = T(n/2) + c = T(n/4) + 2c = \ldots
\]

\[
= T(n/2^3) + 3c = \ldots = T(n/2^{\log n}) + (\log n)c = \Theta(\log n)
\]
proof by “telescoping”

claim. \( T(n) = \Theta(\log n) \)

proof. by induction on \( n \).

• base case: for \( n = 2 \) (or any constant) it takes const comparisons and \( \log n = \text{const} \)

• suppose that the formula is true for every \( k < n \)

• proof for \( n \): substitute \( k = n/2 \)

\[
T(n) = T(n/2) + \Theta(1) = \Theta(\log(n/2)) + \Theta(1) = \Theta(\log n)
\]
Proof by “telescoping” for mergesort

**Proposition.** If $T(n)$ satisfies the following recurrence, then $T(n) = n \log_2 n$.

$$T(n) = \begin{cases} 
1 & n = 1 \\
2T \left( \frac{n}{2} \right) + cn & \text{otherwise}
\end{cases}$$

**Pf.**

$$T(n) \leq 2 \cdot T \left( \frac{n}{2} \right) + cn \leq 4 \cdot T \left( \frac{n}{4} \right) + 2cn \leq 8 \cdot T \left( \frac{n}{8} \right) + 3cn \leq 2^4 \cdot T \left( \frac{n}{2^4} \right) + 4cn \leq \ldots \leq 2^k T \left( \frac{n}{2^k} \right) + kcn$$

The last line corresponds to the base case. We know that the base case is $T(1) = 0$

$$T(1) = T \left( \frac{n}{2^k} \right) \implies k = \log(n)$$

by substituting $\log(n)$ for $k$ we get

$$T(n) \leq 2^{\log(n)} \cdot T \left( \frac{n}{2^{\log(n)}} \right) + \log(n) \cdot n = n + O(n \log n)$$
Master Theorem

For recurrences defined as

\[ T(n) = aT(n/b) + f(n) \]

in which \( a > 0, b \geq 1 \) are constants

Then the overall computational cost is given by

\[ T(n) = \Theta(n^{\log_b a}) + \sum_{k=1}^{\log_b n} a^{k-1} f(n/b^{k-1}) \]

leaves

internal nodes

Such that

\[
T(n) = \begin{cases} 
\Theta(n^{\log_b a}) & \text{if } f(n) = O(n^{\log_b a-\epsilon}) \\
\Theta(n^{\log_b a \log n}) & \text{if } f(n) = \Theta(n^{\log_b a}) \\
\Theta(f(n)) & \text{if } f(n) = \Omega(n^{\log_b a+\epsilon}) \text{ and } af(n/b) \leq cf(n)
\end{cases}
\]

in which \( a \geq 1, b > 1, c < 1 \) and \( \epsilon > 0 \) are constants.
Solve the following recurrences

1. $T(n) = 4T(n/2) + n$
2. $T(n) = 4T(n/2) + n^2$
3. $T(n) = 4T(n/2) + n^3$
Asymptotic time to execute integer operations

Basic operations:
adding, subtracting, multiplying, dividing

Complexity of mathematical operations:
• Usually in this class we count them as constant time
• Today: we look at algorithms for efficiently implementing these basic operations.
• Today: in our analysis one computational step is counted as one bit operation.

Unit of measurement will be bit operations.

Input: two $n$-bit long integers $a$, $b$
Output: arithmetic operation on the two integers
Integer addition

Addition. Given two \( n \)-bit integers \( a \) and \( b \), compute \( a + b \).

Subtraction. Given two \( n \)-bit integers \( a \) and \( b \), compute \( a - b \).

Grade-school algorithm. \( \Theta(n) \) bit operations.
Integer addition

Addition. Given two $n$-bit integers $a$ and $b$, compute $a + b$.

Subtraction. Given two $n$-bit integers $a$ and $b$, compute $a - b$.

Grade-school algorithm. $\Theta(n)$ bit operations.

\[
\begin{array}{c}
2 & 1 & 3 \\
1 & 2 & 5 \\
\hline
3 & 3 & 8
\end{array}
\quad
\begin{array}{c}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0
\end{array}
\]
**Integer addition**

**Addition.** Given two \( n \)-bit integers \( a \) and \( b \), compute \( a + b \).

**Subtraction.** Given two \( n \)-bit integers \( a \) and \( b \), compute \( a – b \).

**Grade-school algorithm.** \( \Theta(n) \) bit operations.

\[
\begin{align*}
\phantom{+} & \phantom{=} a_{n-1} a_{n-2} \ldots a_2 a_1 a_0 \\
+ & \phantom{=} b_{n-1} b_{n-2} \ldots b_2 b_1 b_0 \\
\hline
\text{bits of } a + b & \phantom{=} \\
\end{align*}
\]

**Remark.** Grade-school addition and subtraction algorithms are asymptotically optimal.
Integer multiplication

Multiply $213_{10} \times 125_{10} = 11010101_2 \times 01111101_2$

* (subscript refers to decimal and binary representation)

Grade-school algorithm:
Integer multiplication

Multiply $213_{10} \times 125_{10} = 11010101_{2} \times 01111101_{2}$

- (subscript refers to decimal and binary representation)

Grade-school algorithm:

\[
\begin{array}{ccc}
2 & 1 & 3 \\
\times & 1 & 2 & 5 \\
\hline
1 & 0 & 6 & 5 \\
4 & 2 & 6 \\
+ & 2 & 1 & 3 \\
\hline
2 & 6 & 6 & 2 & 5 \\
\end{array}
\]
Multiplication. Given two \( n \)-bit integers \( a \) and \( b \), compute \( a \times b \).

Grade-school algorithm.

Question: What is the running time?

A. \( O(1) \)

B. \( O(\log n) \)

C. \( O(n) \)

D. \( O(n^2) \)
**Integer multiplication**

**Multiplication.** Given two $n$-bit integers $a$ and $b$, compute $a \times b$.

**Grade-school algorithm.** $\Theta(n^2)$ bit operations.

\[
\begin{array}{cccccccc}
1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
\times & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\
\hline
1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{array}
\]

$n$ bits each

total of $O(n^2)$ bits

$n$ intermediate products

$2n$ additions
Integer multiplication

Multiplication. Given two $n$-bit integers $a$ and $b$, compute $a \times b$.

Grade-school algorithm. $\Theta(n^2)$ bit operations.

Conjecture. [Kolmogorov 1952] Grade-school algorithm is optimal.

Theorem. [Karatsuba 1960] Conjecture is wrong. (his result $O(n^{\log_2 3}) = O(n^{1.59\ldots})$)
Multiplying large integers

Input: \( n \) bit integers \( a \) and \( b \)
Compute: \( ab \)

computing \( ab = \) multiplying two \( n \) bit integers \( \approx O(n^2) \) time

Divide-and-conquer:
- subproblem: same problem on smaller input
  - input size here is the number of bits in the integers
  - goal: compute \( ab \) by multiplying \( n/2 \) bit integers and then combine them
Divide: n bits to n/2

Input: n-bit integers a, b
Compute: ab

How to divide n-bit integers into n/2-bit ints?

$3557_{10} = 3500 + 57 = 35 \times 10^{2} + 57$

$3557_{10} = 110111100101_{2} = 110111000000_{2} + 100101_{2} = 110111_{2} \times 2^{6} + 100101_{2}$

Note: in decimal multiplying by $10^k$ shifts the number k bits to the left. Same in binary, multiplying by $2^k$ shifts k bits to the left (glues k 0s at the end.)
**Question.** Write the representation of the base-3 number $120120_3$ as two parts of \( n/2 \) digit base-3 integers.

A. \( 2 \cdot 120_3 \)

B. \( 3 \cdot 120_3 + 120_3 \)

C. \( 120_3 \cdot 3^3 + 120_3 \)

D. \( 120_3 \cdot 3^6 + 120_3 \cdot 3^3 \)
Divide: n bits to n/2

Input: integers $a, b$ of length $n$ bits

How to “divide”? What are the subproblems?

- split $a$ in to two substrings $A_1, A_0$ of length $n/2$
  
  $a = a_{n-1}a_{n-2} \ldots a_{n/2-1}a_{n/2}a_{n-2} \ldots a_0$ 

  $A_1 = a_{n-1}a_{n-2} \ldots a_{n/2}$ \hspace{1cm} $A_0 = a_{n/2-1} \ldots a_1a_0$

- then $a = A_12^{n/2} + A_0$

- same for $b$: $b = B_12^{n/2} + B_0$
Multiplying large integers using D&C

Input: $n$ bit integers $a$ and $b$

Compute: $ab$

- $A_1, A_0, B_1, B_0$ are $n/2$ bit integers

$ab$ consists of multiplying 2 $n$ bit integers $\sim O(n^2)$ time

$$a = A_12^{n/2} + A_0 \quad b = B_12^{n/2} + B_0$$

Compute:

$$ab = (A_12^{n/2} + A_0)(B_12^{n/2} + B_0) =$$
**Multiplying large integers using D&C**

**Input:** $n$ bit integers $a$ and $b$

**Compute:** $ab$

- $A_1, A_0, B_1, B_0$ are $n/2$ bit integers

$ab$ consists of multiplying $2$ $n$ bit integers $\sim O(n^2)$ time

$a = A_1 2^{n/2} + A_0 \quad b = B_1 2^{n/2} + B_0$

Compute $ab = A_1 B_1 2^n + (A_1 B_0 + A_0 B_1)2^{n/2} + A_0 B_0$

- this formula consists of 4 multiplications of $n/2$ bit numbers and 3 additions

**Divide and conquer approach:** multiply $n/2$ bit integers recursively
Multiplying large integers using D&C

(reminder: \( ab = A_1 B_1 2^n + (A_1 B_0 + A_0 B_1)2^{n/2} + A_0 B_0 \))

Algorithm 1: Multiply(ints \( a, b, n \))

1. if \( n == 1 \) then
2.     return \( ab \);
3. \( m \leftarrow \lfloor \frac{n}{2} \rfloor \);
4. \( A_0 \leftarrow \lfloor a/2^m \rfloor \) /* integer division */
5. \( B_0 \leftarrow \lfloor b/2^m \rfloor \) /* integer division */
6. \( A_1 \leftarrow a \mod 2^m \);
7. \( B_1 \leftarrow b \mod 2^m \);
8. \( x \leftarrow \text{Multiply}(A_1, B_1, m) \);
9. \( y \leftarrow \text{Multiply}(A_1, B_0, m) \);
10. \( z \leftarrow \text{Multiply}(A_0, B_1, m) \);
11. \( w \leftarrow \text{Multiply}(A_0, B_0, m) \);
12. return \( x2^m + (y + z)2^m + w \);

Note that lines 4-7 can be implemented by simple bit-shifts — no real division is taking place!

Recurrence:
Solve the recurrence

\[ T(n) \leq \begin{cases} 
1 & n = 1 \\
4T\left(\frac{n}{2}\right) + O(n) & \text{otherwise}
\end{cases} \]
Solve the recurrence

\[
T(n) \leq \begin{cases} 
1 & n = 1 \\
4T\left(\frac{n}{2}\right) + O(n) & \text{otherwise}
\end{cases}
\]

Use telescoping and solve \( T(n) = 4 \cdot T\left(\frac{n}{2}\right) + cn \)

\[
T(n) = 4 \cdot T\left(\frac{n}{2}\right) + cn = 4 \left( 4 \cdot T\left(\frac{n}{4}\right) + c\frac{n}{2} \right) + cn = (2^2)^2 \cdot T\left(\frac{n}{2^2}\right) + 3cn = \ldots
\]

\[
= (2^2)^3 \cdot T\left(\frac{n}{2^3}\right) + 5cn = \ldots = (2^2)^k \cdot T\left(\frac{n}{2^k}\right) + (k + 1)cn = \ldots
\]

we want to substitute the base case \( T(1) \), which happens when \( k = \log n \)

\[
= (2^2)^{\log n}T(1) + (\log n)cn = n^2 + \Theta(1)n \log n = \Theta(n^2)
\]

Now we can prove by induction that \( T(n) = \Theta(n^2) \)
General formula for recurrences of specific form

Recurrence:

\[ T(n) = 4T\left(\frac{n}{2}\right) + cn = cn^{\log_2 4} = \Theta(n^2) \]

General form:

\[ T(n) = qT\left(\frac{n}{2}\right) + cn = cn^{\log_2 q} \text{ for } q > 2 \]

Discussed in detail: Kleinberg-Tardos pages 214 - 218
Multiplying large integers using D&C

(reminder: \(ab = A_1B_12^n + (A_1B_0 + A_0B_1)2^{n/2} + A_0B_0\))

Algorithm 1: Multiply(ints \(a, b, n\))

```plaintext
1 if \(n == 1\) then
2    return \(ab\);
3 \(m \leftarrow \left\lfloor \frac{n}{2} \right\rfloor\);
4 \(A_0 \leftarrow \left\lfloor a/2^m \right\rfloor\) /* integer division */
5 \(B_0 \leftarrow \left\lfloor b/2^m \right\rfloor\) /* integer division */
6 \(A_1 \leftarrow a \mod 2^m\);
7 \(B_1 \leftarrow b \mod 2^m\);
8 \(x \leftarrow \text{Multiply}(A_1, B_1, m)\);
9 \(y \leftarrow \text{Multiply}(A_1, B_0, m)\);
10 \(z \leftarrow \text{Multiply}(A_0, B_1, m)\);
11 \(w \leftarrow \text{Multiply}(A_0, B_0, m)\);
12 return \(x2^m + (y + z)2^m + w\);
```

Note that lines 4-7 can be implemented by simple bit-shifts — no real division is taking place!

Recurrence: \(T(n) = 4T\left(\frac{n}{2}\right) + cn\)  \(T(n) = \Theta(n^2)\)
Karatsuba’s algorithm

Input:  \( n \) bit integers \( a \) and \( b \)

Compute: \( ab \)

\[
a = A_1 2^{n/2} + A_0
\]

\[
b = B_1 2^{n/2} + B_0
\]

Compute \( ab = A_1 B_1 2^n + (A_1 B_0 + A_0 B_1)2^{n/2} + A_0 B_0 \)

Karatsuba’s trick: \((A_1 + A_0)(B_1 + B_0) = A_1 B_1 + A_0 B_0 + (A_1 B_0 + A_0 B_1)\)
Karatsuba’s algorithm

Input: $n$ bit integers $a$ and $b$
Compute: $ab$

$a = A_12^{n/2} + A_0$

$b = B_12^{n/2} + B_0$

Compute $ab = A_1B_12^n + (A_1B_0 + A_0B_1)2^{n/2} + A_0B_0$

Karatsuba’s trick: $(A_1+A_0)(B_1+B_0) = A_1B_1 + A_0B_0 + (A_1B_0 + A_0B_1)$
Karatsuba’s algorithm

Input: \( n \) bit integers \( a \) and \( b \)

Compute: \( ab \)

\[
a = A_12^{n/2} + A_0
\]
\[
b = B_12^{n/2} + B_0
\]

Compute \( ab = A_1B_12^n + (A_1B_0 + A_0B_1)2^{n/2} + A_0B_0 \)

Karatsuba’s trick: \((A_1+A_0)(B_1+B_0) = A_1B_1 +  A_0B_0 + (A_1B_0 + A_0B_1)\)

Thus we get \( x = A_1B_1 \), \( y = A_0B_0 \), \( z = (A_1+A_0)(B_1+B_0) \) multiply \( n/2 \) bit integers \( n/2 \) bits!

\[
ab = A_1B_12^n + (A_1B_0 + A_0B_1)2^{n/2} + A_0B_0 = x2^n + (z - x - y)2^{n/2} + y
\]

The new formula contains only 3 multiplications of \( n/2 \) bits.
Algorithm 1: Karatsuba(ints \(a, b, n\))

1. if \(n == 1\) then
2.    return \(ab\);
3. \(m \leftarrow \left\lfloor \frac{n}{2} \right\rfloor\);
4. \(A_0 \leftarrow \lfloor a/2^m \rfloor \) /* integer division */
5. \(B_0 \leftarrow \lfloor b/2^m \rfloor \) /* integer division */
6. \(A_1 \leftarrow a \mod 2^m;\)
7. \(B_1 \leftarrow b \mod 2^m;\)
8. \(x \leftarrow \text{Karatsuba}(A_1, B_1, m);\)
9. \(y \leftarrow \text{Karatsuba}(A_0, B_0, m);\)
10. \(z \leftarrow \text{Karatsuba}(A_1 + A_0, B_1 + B_0, m);\)
11. return \(x2^m + (z - x - y)2^m + y;\)
Algorithm 1: Karatsuba(ints a, b, n)

1  if $n == 1$ then
2       return $ab$;
3  $m \leftarrow \left\lfloor \frac{n}{2} \right\rfloor$;
4  $A_0 \leftarrow \lfloor a/2^m \rfloor$ /* integer division */
5  $B_0 \leftarrow \lfloor b/2^m \rfloor$ /* integer division */
6  $A_1 \leftarrow a \mod 2^m$;
7  $B_1 \leftarrow b \mod 2^m$;
8  $x \leftarrow$ Karatsuba($A_1$, $B_1$, $m$);
9  $y \leftarrow$ Karatsuba($A_0$, $B_0$, $m$);
10 $z \leftarrow$ Karatsuba($A_1 + A_0$, $B_1 + B_0$, $m$);
11 return $x2^{2m} + (z - x - y)2^m + y$;

Question: What is the recurrence for this algorithm?

A. $T(n) = \Theta(n)$            C. $T(n) = 3 \cdot T \left( \frac{n}{2} \right) + \Theta(n)$

B. $T(n) = 2 \cdot T \left( \frac{n}{3} \right) + \Theta(n)$            D. $T(n) = 3 \cdot T \left( \frac{n}{3} \right) + \Theta(1)$
Algorithm 1: Karatsuba(ints $a$, $b$, $n$)

1. if $n == 1$ then
2.   return $ab$;
3. $m \leftarrow \lfloor \frac{n}{2} \rfloor$;
4. $A_0 \leftarrow \lfloor a/2^m \rfloor$ /* integer division */
5. $B_0 \leftarrow \lfloor b/2^m \rfloor$ /* integer division */
6. $A_1 \leftarrow a \mod 2^m$;
7. $B_1 \leftarrow b \mod 2^m$;
8. $x \leftarrow$ Karatsuba($A_1$, $B_1$, $m$);
9. $y \leftarrow$ Karatsuba($A_0$, $B_0$, $m$);
10. $z \leftarrow$ Karatsuba($A_1 + A_0$, $B_1 + B_0$, $m$);
11. return $x2^{2m} + (z - x - y)2^m + y$;

$T(n) = 3T(\frac{n}{2}) + cn$

$T(n) = \Theta(n^{\log_2 3}) = \Theta(n^{1.59...})$
Recursion tree method

Solve \( T(n) = 3T\left(\frac{n}{2}\right) + cn \) where \( c \geq 2 \) const
Recursion tree method - TopHat

Solve \( T(n) = 3T\left(\frac{n}{2}\right) + cn \) where \( c \geq 2 \) const

Question: what is the height of the tree and number of nodes at layer k? (you may assume the root is at layer 0)

A. \( \log_3 n \) & \( n^k \)
B. \( \log_2 n \) & \( 3^k \)
C. \( \log_2 n \) & \( n^3 \)
D. \( \log_3 n \) & \( k^3 \)
Recursion tree method

Solve \( T(n) = 3T\left(\frac{n}{2}\right) + cn \) where \( c \geq 2 \) const

\[
h = \log_2 n \quad c = \log_2 n \quad c = \log_2 n \quad c = \log_2 n \quad c = \log_2 n
\]

\[
(\frac{3}{2})^{\log_3 n} = \frac{1}{n}3^{\log_2 n} = 3^{\log_3 n} \cdot 3^{\frac{1}{\log_3 2}} = n3^{\log_2 3}
\]

\[
(\frac{3}{2})^{\log_3 n} cn = cn^{\log_2 3} = cn^{1.59...}
\]
General formula for recurrences of specific form

Recurrence for Karatsuba’s:

$$T(n) = 3T\left(\frac{n}{2}\right) + cn = cn^{\log_2 3}$$

General form:

$$T(n) = qT\left(\frac{n}{2}\right) + cn = cn^{\log_2 q} \text{ for } q > 2$$

Discussed in detail: Kleinberg-Tardos pages 214 - 218
## History of asymptotic complexity of integer multiplication

<table>
<thead>
<tr>
<th>Year</th>
<th>Algorithm</th>
<th>Order of Growth</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>grade-school</td>
<td>$\Theta(n^2)$</td>
</tr>
<tr>
<td>1962</td>
<td>Karatsuba–Ofman</td>
<td>$\Theta(n^{1.585})$</td>
</tr>
<tr>
<td>1963</td>
<td>Toom–3, Toom–4</td>
<td>$\Theta(n^{1.465})$, $\Theta(n^{1.404})$</td>
</tr>
<tr>
<td>1966</td>
<td>Toom–Cook</td>
<td>$\Theta(n^{1+\varepsilon})$</td>
</tr>
<tr>
<td>1971</td>
<td>Schönhage–Strassen</td>
<td>$\Theta(n \log n \log \log n)$</td>
</tr>
<tr>
<td>2007</td>
<td>Furer</td>
<td>$n \log n \ 2^{O(\log^* n)}$</td>
</tr>
<tr>
<td>2019</td>
<td>Harvey, van der Hoeven</td>
<td>$\Theta(n \log n)$</td>
</tr>
</tbody>
</table>

Faster than grade-school algorithm for about 320–640 bits.

Remark. GNU Multiple Precision Library uses one of five different algorithms depending on size of operands.