Agenda:

- Loop Invariants (CLRS p.18-20)
- Asymptotic Growth of Functions (CLRS Ch.3)
  - Big-O, Big-Ω, Θ, little-o, little-ω

Why Prove Correctness

- Once you developed an algorithm, you at least need to show it does what it is supposed to do (and *never* errs!)
- What is the difference between *testing* and proving?
- To prove a program is correct, we start by wording correctness formally:

  Claim: For any instance I (satisfying ____________),
  `Algorithm-name`(I) returns ____________

- E.g., For any two non-negative integers a and b, `Multiply`(a, b) returns the product $a \times b$.

Basic Proofs

- For simple statements, just reason with the effect of code (using logic).
- `procedure Swap`(a, b)
  $temp \leftarrow a$
  $a \leftarrow b$
  $b \leftarrow temp$
- Claim: for any two pointers a and b, `Swap`(a, b) indeed assigns a the element that b pointed to originally, and assigns b the element that a pointed to originally.
- Proof: Assume that initially a points to object x and b points to object y.
  The first line creates a new pointer $temp$ that also points to x. The second line sets a to point to y (just like b). Finally the last line sets b to point to the same object as $temp$, i.e. x. So, at the end of the execution, a points to y and b points to x, as required. $\square$

Proving Correctness using Loop Invariants

- If a code is written using recursion, prove correctness using induction.
- For code written using loops, prove correctness by the loop invariant method.
- A **loop-invariant** is an assertion about the state of the code that is *always* true at the beginning of each loop-iteration.
- Not any assertion, but an assertion that *accurately* describes the *cumulative effect* of repeatedly iterating through the loop; an assertion we can also use to prove the correctness of the code.
- Step 1: Identify the loop invariant
  - Q1: Do I understand what the loop does?
  - Q2: Do I understand the cumulative effect of the loop?
  - Q3: Can I word exactly the cumulative effect of the loop?
- Step 2: Prove the loop invariant for
  - Initialization
  - Maintenance
  - Termination #1: Does the loop halt eventually?
  - Termination #2: How do I prove correctness from the LI?

Step #1: Identifying and Rigorously Stating the Loop Invariant

- Example

```
procedure FindSum(A, n)
sum ← A[1]
j ← 2
while (j ≤ n)
    sum ← sum + A[j]
    j ← j + 1
return sum
```

- Returns the sum of all elements in $A[1..n]$. How do we prove it?
- Q1: What does the loop do? A: Adds $A[j]$ to sum and increments j
- Q2: So what is always true **at the beginning of each loop iteration**?
  A: sum holds the summation of $A[1] + A[2] + ... + A[j-1]$
- How would that lead to desired conclusion when loop terminate?
  A: The loop exits at $j = n+1$, and we have $sum = A[1] + ... + A[n]$.
- So, the loop-invariant is:

  "At the beginning of each loop iteration, $sum = \sum_{i=1}^{j-1} A[i]$**"**

$j=2$ $sum = A[1]$

$j=3$ $sum = A[1]+A[2]$

$j=4$ $sum = A[1]+A[2]+A[3]$

...

$\sum_{i=1}^{j-1} A[i]$

$j=n+1$, $sum = \sum_{i=1}^{j-1} A[i]$ $(j-1)=n$

Step #1: Identifying and Rigorously Stating the Loop Invariant

- The same loop-invariant can be written in many equivalent forms
  - "At the beginning of each loop iteration, $sum = A[1] + A[2] + ... + A[j-1]$"
  - "At the beginning of each loop iteration sum is the summation of the elements in $A[1, ..., j-1]$"
  - "At the beginning of each loop iteration sum is the summation of the first $j-1$ elements in A"
  - or any other equivalent form

Step #1: Identifying and Rigorously Stating the Loop Invariant

- It DOES matter that the loop invariant is stated correctly and in a way that will give the correctness of the overall algorithm
  - "At the beginning of each loop iteration $sum = A[j]$" — WRONG
  - "At the beginning of each loop iteration sum is the summation of the elements in $A[1, ..., j]$" — WRONG
  - "At the beginning of each loop iteration $j > 0$" — UNINFORMATIVE
  - "At the beginning of each loop iteration $sum = sum^{\text{previous\_iteration}} + A[j-1]$" — UNINFORMATIVE
- To make sure you don't mess with the indices — check it! Plug-in values of j ($j = 1$, $j = 2$, . . ., $j = n$) and check.

Step #2: Proving Loop Invariants

- Once we have identified and stated the LI, it is time to prove it — and to use it to prove the correctness of the entire code.
- Proving LI means proving the following 4 parts

- Initialization:
  - Does LI hold before the loop starts?
- Maintenance:
  - If LI holds at the beginning of j'-th iteration, does it hold also at the beginning of the $(j'+1)$-th iteration?
- Termination #1:
  - Does the loop terminate?
- Termination #2:
  - When the loop terminates, does it prove the correctness of the overall algorithm / the claim we were making?

Step #2: Proving Loop Invariant

- Our loop-invariant: "At the beginning of each loop iteration, $sum = \sum_{i=1}^{j-1} A[i]$"
- Initially: Before the loop begins $sum = A[1] = A[1, ..., (2-1)]$
- Maintenance: Suppose that at the beginning of iteration j (the $(j-1)$-th iteration), $sum = \sum_{i=1}^{j-1} A[i]$.

  Then, at the beginning of iteration $j+1$ (j-th iteration),

  $sum^{\text{after}} = sum^{\text{before}} + A[j] \overset{\text{LI}}{=} \sum_{i=1}^{j-1} A[i] + A[j] = \sum_{i=1}^{j} A[i] = \sum_{i=1}^{j^{\text{after}}-1} A[i]$
- Termination #1: The loop terminates as we only increment j, so eventually we would have $j > n$
- Termination #2: When the while-loop terminates, $j = n+1$, in which case the LI implies $sum = \sum_{i=1}^{n} A[i]$. ~~We return $sum/n = \frac{1}{n}\sum_{i=1}^{n} A[i]$ which by definition is the average of all elements in $A[1, ..., n]$.~~

Loop Invariant

j=2 $\boxed{8}$ 2 4 7 3 5

j=3 $\boxed{2 \quad 8}$ 4 7 3 5

j=4 $\boxed{2 \quad 4 \quad 8}$ 7 3 5

Loop Invariants Example

<u>procedure InsertionSort(A)</u>

for (j from 2 to n)

  $key \leftarrow A[j]$ **insert $A[j]$ into sorted sublist $A[1..j-1]$

  $i \leftarrow j-1$

  while ($i > 0$ and $A[i] > key$)

    $A[i+1] \leftarrow A[i]$

    $i \leftarrow i-1$

  $A[i+1] \leftarrow key$

Loop Invariants Example

- To prove correctness - use two loop invariants, one *nested* inside another.
- What is the loop invariant of the for-loop?
- LI1: "At the beginning of each for-loop iteration $A[1, ..., j-1]$ contains the same elements that were there initially, only in order."
- Initialization: $j = 2$ and clearly $A[1]$ is a sorted array of size 1.
- Maintenance: TBD
- Termination #1: We don't alter j at the body of the loop + Termination of the while-loop (TBD)
- Termination #2: When the loop terminates, $j = n + 1$ so $A[1, ...n]$ (which is the whole array) is sorted.

More Loop Invariants Examples

- To prove the maintenance property of the LI for the for-loop we actually use a LI for the while-loop
- LI2: Let $A^{\text{before}}[1..j]$ denote the array before we started iterating through the `while` loop. Then at the beginning of each iteration of the `while` loop:
  (i) $A[1..i+1] = A^{\text{before}}[1..i+1]$
  (ii) $A[i+2..j] = A^{\text{before}}[i+1..j-1]$
- Initialization / maintenance / termination #1 of LI2:
  - HW

More Loop Invariants Examples

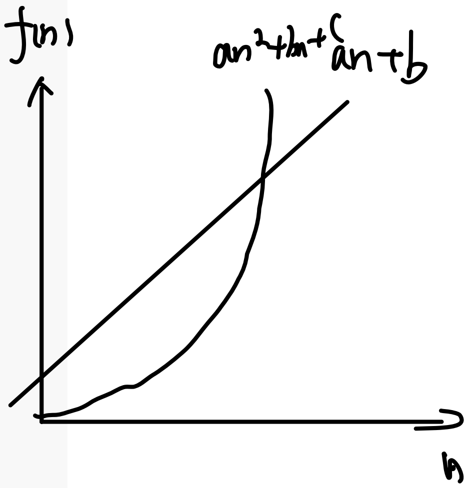
- To prove the maintenance property of the LI for the for-loop we actually use a LI for the while-loop
  - LI2: Let $A^{\text{before}}[1..j]$ denote the array before we started iterating through the `while` loop. Then at the beginning of each iteration of the `while` loop:
    (i) $A[1..i+1] = A^{\text{before}}[1..i+1]$
    (ii)$A[i+2..j] = A^{\text{before}}[i+1..j-1]$
- The termination #2 of LI2 is how to derive the maintenance property of LI from the termination of the while-loop.
- Termination #2: At the end of while loop, i is the largest entry in $\{1, 2, 3, ..., j-1\}$ for which $A[i] \leq key$ (or 0, if no such entry exists).
  So LI2 together with putting key at $A[i+1]$, we have that
  $$A[1,..j] = \left[A^{\text{before}}[1,..,i], key, A^{\text{before}}[i+1,..,j-1]\right]$$
  As $A^{\text{before}}[1,..j-1]$ was sorted & by definition of $i \Rightarrow A[1,..j]$ is sorted.
  □

Loop invariant vs. Mathematical induction

- Arguing correctness
  - When recursion is involved, use induction
  - When loop is involved, use loop invariant (and induction)
- Common points
  - initialization vs. base step
  - maintenance vs. inductive step
- Difference
  - termination vs. infinite

Asymptotic notation for Growth of Functions: Motivations

- Analysis of algorithms becomes analysis of functions
- The (WC) running time of InsertionSort is characterized by a quadratic function $f(n) = an^2 + bn + c$
- For some sort algorithms (e.g., mergeSort, later) the running time is $g(n) = cn \log n$.
- Which algorithm runs faster? In what sense?



Asymptotic notation for Growth of Functions: Motivations

- To simplify algorithm analysis, want function notation which indicates *rate of growth* (a.k.a., *order* of complexity), and denotes a set of functions

- $O(f(n))$ — **read as "big O of $f(n)$"** $h(n) \leq O < f(n)$
- $\Omega(f(n))$ — **read as "big Omega of $f(n)$"** $h(n) \leq \Omega\ cf(n)$
- $\Theta(f(n))$ — **read as "Theta of $f(n)$"** $h(n) \leq \theta < f(n)$
- $o(f(n))$ — **read as "little o of $f(n)$"**
- $\omega(f(n))$ — **read as "little omega of $f(n)$"**

Big-O Notation: $O(f(n))$

- (Roughly) The set of functions which, as n gets large, grow no faster than a constant times $f(n)$.

- Definition: A function $h(n) : \mathbb{N} \to \mathbb{R}$ belongs to $O(f(n))$ if there exist constants $c > 0$ and $n_0 \in \mathbb{N}$ such that for all $n \geq n_0$ it holds that $0 \leq h(n) \leq cf(n)$ (we can omit "$0 \leq$" in the sequel).

- Examples:

$$482n^2 \in O(n^2) \qquad 482n^2 \in O(n^3)$$

$$482n^2 \in O(n^{2.5}) \qquad 482n^2 \in O(n^{2.001})$$

$$n^3 + 255n^2 + n^{2.999} \in O(n^3)$$

$$h(n) = \begin{cases} 5^n, & n \leq 10^{120} \\ n^2, & n > 10^{120} \end{cases} \in O(n^2)$$

$6n^3+2n^2+n+5 \leq O(n^3)$

$6n^3+2n^2+n+5 \leq n^3$

$6+\frac{2}{n}+\frac{1}{n^2}+\frac{5}{n^3} \leq c$

$c=9 \quad n=3$

Big-O Notation: $O(f(n))$

Inverse: A function $h(n) \notin O(f(n))$ if no matter what $c > 0$ and $n_0 \in \mathbb{N}$ we choose, we can always find a large enough $n > n_0$ s.t. $h(n) > cf(n)$. That is, h is NOT upper bounded by f within a constant factor.

- [Examples:]

$$482n^2 \notin O(n) \qquad \frac{1}{482}n^2 \notin O(n^{1.99999})$$

$$n^2 \notin O(n^p) \text{ for any } p < 2$$

$$n^3 + 255n^2 + n^{2.999} \notin O(n^{2.99999})$$

$$h(n) = \begin{cases} n^2, & n \text{ is even} \\ n^3, & n \text{ is odd} \end{cases} \notin O(n^2)$$

- The class of constant functions is expressed by $O(1)$. The notation comes from $O(n^0)$ for degree-0 polynomial.

$2n^2 = O(n^3)$

$n^2/2 = \Theta(n^2)$

$\neq o(n^2)$

Definitions

- $O(f(n))$ is the set of functions $h(n)$ that
  - roughly, grow no faster than $f(n)$ $\lesssim$
  - Formally: $h(n) \in O(f(n))$ if $\exists c > 0, n_0 \in \mathbb{N}$, such that for all $n \geq n_0$ we have $h(n) \leq cf(n)$.

- $\Omega(f(n))$ is the set of functions $h(n)$ that
  - roughly, grow at least as fast as $f(n)$ $\gtrsim$
  - Formally: $h(n) \in \Omega(f(n))$ if $\exists c > 0, n_0 \in \mathbb{N}$, such that for all $n \geq n_0$ we have $h(n) \geq cf(n)$.
  - $h(n) \in \Omega(f(n))$ if and only if $f(n) \in O(h(n))$

- $\Theta(f(n))$ is the set of functions $h(n)$ that
  - roughly, grow at the same rate as $f(n)$ $\eqsim$
  - Formally: $h(n) \in \Theta(f(n))$ if $\exists c_0 > 0, c_1 > 0, n_0 \in N$, such that for all $n \geq n_0$ we have $c_0 f(n) \leq h(n) \leq c_1 f(n)$.
  - $\Theta(f(n)) = O(f(n)) \cap \Omega(f(n))$

Definitions (Cont'd):

- $o(f(n))$ is the set of functions $h(n)$ that
  - roughly, grow strictly slower than $f(n)$
  - Formally: $h(n) \in o(f(n))$ if $\lim_{n\to\infty} \frac{h(n)}{f(n)} = 0$
  - I.e. for every $\epsilon > 0$, there exists $n_\epsilon \in \mathbb{N}$ such that for every $n \geq n_\epsilon$ it holds that $\frac{h(n)}{f(n)} < \epsilon$
  - Subset of $O(f(n))$

- $\omega(f(n))$ is the set of functions $h(n)$ that
  - roughly, grow strictly faster than $f(n)$
  - Formally: $h(n) \in \omega(f(n))$ if $\lim_{n\to\infty} \frac{h(n)}{f(n)} = \infty$
  - I.e. for every $M > 0$, there exists $n_M \in \mathbb{N}$ such that for all $n \geq n_M$ it holds that $\frac{h(n)}{f(n)} > M$.
  - Subset of $\Omega(f(n))$
  - $h(n) \in \omega(f(n))$ if and only if $f(n) \in o(h(n))$

Note:

- the textbook overloads "="
  - Textbook uses $g(n) = O(f(n))$
  - But we define $O(f(n))$ as a *set* of functions.
  - Both are by now correct
  - My advice: use $g(n) \in O(f(n))$.

Examples:

- Which of the following belongs to $O(n^3)$, $\Omega(n^3)$, $\Theta(n^3)$, $o(n^3)$, $\omega(n^3)$?

  1. $f_1(n) = 19n$
  2. $f_2(n) = 77n^2$
  3. $f_3(n) = 6n^3 + n^2 \log n$
  4. $f_4(n) = 11n^4$

Answers:

- $f_1, f_2, f_3 \in O(n^3)$
  $f_1(n) \leq 19n^3$, for all $n \geq 0$ — $c_0 = 19$, $n_0 = 0$
  $f_2(n) \leq 77n^3$, for all $n \geq 0$ — $c_0 = 77$, $n_0 = 0$
  $f_3(n) \leq 6n^3 + n^2 \cdot n$, for all $n \geq 1$, since $\log n \leq n$
- $f_3, f_4 \in \Omega(n^3)$
  $f_3(n) \geq 6n^3$, for all $n \geq 1$, since $n^2 \log n \geq 0$
  $f_4(n) \geq 11n^3$, for all $n \geq 0$

Answers (Cont'd):

- $f_3 \in \Theta(n^3)$ (why?)
- $f_1, f_2 \in o(n^3)$
- $f_1(n)$: $\lim_{n\to\infty} \frac{19n}{n^3} = \lim_{n\to\infty} \frac{19}{n^2} = 0$

  $f_2(n)$: $\lim_{n\to\infty} \frac{77n^2}{n^3} = \lim_{n\to\infty} \frac{77}{n} = 0$

  $f_3(n)$: $\lim_{n\to\infty} \frac{6n^3+n^2 \log n}{n^3} = \lim_{n\to\infty} 6 + \frac{\log n}{n} = 6$

  $f_4(n)$: $\lim_{n\to\infty} \frac{11n^4}{n^3} = \lim_{n\to\infty} 11n = \infty$
- $f_4 \in \omega(n^3)$

More big-O Notation Properties

- **Reflexivity:** For any function f it holds that $f(n) \in O(f(n))$ (the same goes for $\Omega(\cdot), \Theta(\cdot)$)

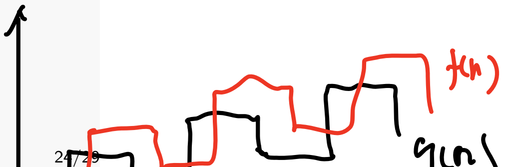
- **Additivity:** If $f(n), g(n) \in O(h(n))$ then $f(n)+g(n) \in O(h(n))$ (same goes for all other notations; the same holds for any constant number of functions)

- BUT doesn't hold for $\underbrace{f(n) + f(n) + ... + f(n)}_{g(n)}$

- **Multiplicative:** If $f_1(n) \in O(f_2(n))$ and $g_1(n) \in O(g_2(n))$ and all functions take *only positive values*, then $f_1(n) \cdot g_1(n) \in O(f_2 \cdot g_2)$ (same goes for all other notations)

- **Transitivity:** if $f(n) \in O(g(n))$ and $g(n) \in O(h(n))$ then $f(n) \in O(h(n))$ (same goes for all other notations!)

- BUT if $f(n) \in O(h(n))$ and $g(n) \in O(h(n))$ then f and g may not be comparable...



$\log_b n = k \quad b^k = n$

Logarithm Review (CLRS : p56)

For any $b > 1$ and $n > 0$ we define

- Definition of $\log_b(n)$: $b^{\log_b n} = n$
- $\log_b n$ as a function in n: increasing, one-to-one
- $\ln n = \log_e n$ (natural logarithm)
- $\lg n = \log_2 n$ (base 2, binary)

- $\log_b 1 = 0$
- For any x and any p, $\log_b x^p = p \log_b x$
- For any x and any y, $\log_b(xy) = \log_b x + \log_b y$
- For any x and any y, $x^{\log_b y} = y^{\log_b x}$
- For any x and any $c > 1$, $\log_b x = (\log_b c)(\log_c x)$
- For any $b > 1$ we have $\Theta(\log_b n) = \Theta(\log n)$
- $(\log n)^k \in o(n^\epsilon)$, for any fixed positives k and ϵ

Handy 'big O' tips:

- $h(n) \in O(f(n))$ if and only if $f(n) \in \Omega(h(n))$
- $h(n) \in o(f(n))$ if and only if $f(n) \in \omega(h(n))$
- limit rules: if $\lim_{n\to\infty} \frac{h(n)}{f(n)}$ exists then
  - limit $= \infty$, then $h \in \Omega(f), \omega(f)$
  - limit $= k$ for some $0 < k < \infty$, then $h \in \Theta(f)$
  - limit $= 0$, then $h \in O(f), o(f)$
- L'Hôpital's rules: if $\lim_{n\to\infty} h(n) = \infty$, $\lim_{n\to\infty} f(n) = \infty$, and $h'(n)$, $f'(n)$ exist, then

  $$\lim_{n\to\infty} \frac{h(n)}{f(n)} = \lim_{n\to\infty} \frac{h'(n)}{f'(n)}$$

  e.g., $\lim_{n\to\infty} \frac{\ln n}{n} = \lim_{n\to\infty} \frac{1}{n} = 0$
- Cannot always use L'Hôpital's rules. *e.g.*,
  - $h(n) = \begin{cases} 1, & \text{if } n \text{ even} \\ n^2, & \text{if } n \text{ odd} \end{cases}$
  - $\lim_{n\to\infty} \frac{h(n)}{n^2}$ does NOT exist (but $\lim_{n\to\infty} \frac{h(n)}{n^3}$ does)
  - Still, we have $h(n) \in O(n^2)$, $h(n) \in \Omega(1)$, etc.

Handy 'big O' tips:

- If $f, g : \mathbb{N} \to \mathbb{R}$ are both positive functions then $f(n) \geq g(n)$ iff $2^{f(n)} \geq 2^{g(n)}$.
  - Hence, because $\forall n, n \leq 2^n$ then $\forall n \geq 1$, $\log(n) \leq n$. So $\log(n) \in O(n)$.
- It is often useful to write $f(n) = 2^{\log(f(n))}$.
- Another trick: if $f(n) \geq g(n)$ for all n, then for any function h, $f(h(n)) \geq g(h(n))$
  - Since $n \geq \log(n)$, then $\sqrt{n} \geq \log(\sqrt{n}) = \frac{1}{2}\log(n)$ so $\log(n) \in O(\sqrt{n})$
  - Similarly, we can show that for any fixed $\epsilon > 0$, $\log(n) \in O(n^{\epsilon})$.
  - Moreover, for any fixed $\epsilon > 0$, we can show $\log(n) \in O(n^{\frac{\epsilon}{2}})$. Since $n^{\frac{\epsilon}{2}} \in o(n^{\epsilon})$ we get $\log(n) \in o(n^{\epsilon})$.
- And if h is a monotone non-decreasing function then we also have $h(f(n)) \geq h(g(n))$.
  - So since $\forall n, n \leq n^2$, then $\forall n, \sqrt{n} \leq n$, then $\forall n \geq 1$, $\sqrt{\log(n)} \leq \log(n)$, then $\forall n \geq 1, 2^{\sqrt{\log(n)}} \leq 2^{\log(n)} = n$ for every n, so $2^{\sqrt{\log(n)}} \in O(n)$.
- $O(\cdot), \Omega(\cdot), \Theta(\cdot), o(\cdot), \omega(\cdot)$ – JUST useful asymptotic notations

Tower of Exponents

- Define $f(n) = 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}} \}_{n \text{ times}}$
- So $f(1) = 2$, $f(2) = 2^2 = 4$, $f(3) = 2^{2^2} = 2^4 = 16$, $f(4) = 2^{16} = 65,536$, $f(5)$ has more than $19,500$ digits!
- REALLY fast growing function.

$\log^*$ function (iteraive logarithem)

- The inverse of the tower of exponent.
- Formally: $\log^*(n) = \min\{k : 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}} \}_{k \text{ times}} \geq n\}$. E.g. $\log^* 2 = 1$, $\log^* 2^2 = 2$, $\log^* 2^{2^2} = \log^* 16 = 3$, $\log^* 2^{2^{2^2}} = \log^* 65536 = 4$, ... REALLY slow growing function.
- Alternatively: $\min\left\{k : \underbrace{\lg\lg\lg\ldots\lg}_{k}(n) \leq 1\right\}$: Intuitively, the smallest k s.t. applying log function k times yields a value 1 or under.

Another useful formula is Stirling's Approximation: $n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$.

Example: The following functions are ordered in increasing order of growth (each is in big-Oh of next one). Those in the same group are in big-Theta of each other.

$$\{n^{1/\log n},\quad 1\},\ \log^*(n),\quad \{\log\log n,\quad \ln\ln n\},\quad \sqrt{\log n},\quad \ln n,\quad \log^2 n,$$

$$2^{\sqrt{\log n}},\quad (\sqrt{2})^{\log n},\quad 2^{\log n},\quad \{n\log n,\quad \log(n!)\},\quad n^2,\quad \{n^3,\quad 8^{\log(n)}\}$$

$$(\log n)!,\quad \{(\log n)^{\log n},\quad n^{\log\log n}\},\quad \left(\frac{3}{2}\right)^n,$$

$$2^n,\quad n\cdot 2^n,\quad e^n,\quad n!,\quad (n!)^2,\quad (n^2)!,\quad 2^{2^n},\quad 2^{2^{\cdot^{\cdot^{\cdot^2}}}}\Big\}{\scriptstyle n \text{ times}}$$