

CSE 250: Recursion

Lecture 11

Sept 20, 2024

Reminders

- PA1 Implementation due Sun, Sept 22 at 11:59 PM
 - Implement a Sorted Linked List
- WA2 to be released this weekend

ArrayList

`ArrayList`: An array with empty space at the end.

`add(v)`

- Use an empty slot if one is available
- When you run out of space, double the size.

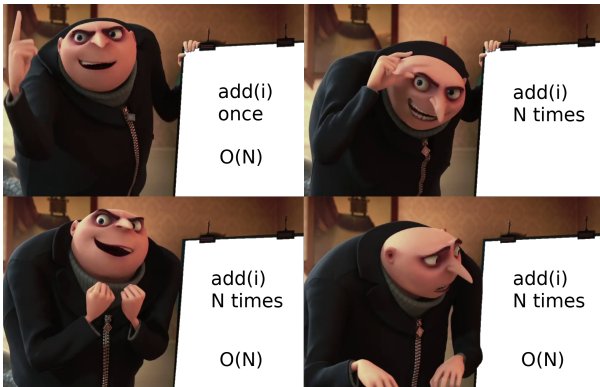
`add(idx, v)`

- As `add`, but shift elements $\geq \text{idx}$ right one space first.

`remove(idx)`

- Shift elements $> \text{idx}$ left one spot.

Huh?



Despicable Me; ©2010 Universal Pictures

Amortized Runtimes

$$T_{add}(N) = \begin{cases} \theta(1) & \text{if } capacity > size \\ \theta(N) & \text{otherwise} \end{cases}$$

Amortized Runtimes

$$T_{add}(N) = \begin{cases} \theta(1) & \text{if } capacity > size \\ \theta(N) & \text{otherwise} \end{cases}$$

$$T_{add}(N) \in O(N)$$

Amortized Runtimes

$$T_{add}(N) = \begin{cases} \theta(1) & \text{if } capacity > size \\ \theta(N) & \text{otherwise} \end{cases}$$

$$T_{add}(N) \in O(N)$$

- Any **one** call could be $O(N)$
- But the $O(N)$ case happens rarely.

Amortized Runtimes

$$T_{add}(N) = \begin{cases} \theta(1) & \text{if } capacity > size \\ \theta(N) & \text{otherwise} \end{cases}$$

$$T_{add}(N) \in O(N)$$

- Any **one** call could be $O(N)$
- But the $O(N)$ case happens rarely.
 - ... rarely enough (with doubling) that the expensive call amortizes over the cheap calls.

ArrayList

- Double the size
 - Copy 2^i elements
 - Get $2^i - 1$ $\theta(1)$ freebie inserts

ArrayList

- Double the size
 - Copy 2^i elements
 - Get $2^i - 1$ $\theta(1)$ freebie inserts
- Contrast with always adding k slots
 - Copy $k \cdot i$ elements
 - Get k $\theta(1)$ freebie inserts

ArrayList

- Double the size
 - Copy 2^i elements
 - Get $2^i - 1$ $\theta(1)$ freebie inserts
 - Contrast with always adding k slots
 - Copy $k \cdot i$ elements
 - Get k $\theta(1)$ freebie inserts
- (Put $\theta(2^i)$ on a credit card)

ArrayList

- Double the size
 - Copy 2^i elements (Put $\theta(2^i)$ on a credit card)
 - Get $2^i - 1$ $\theta(1)$ freebie inserts (Pay down $\theta(1)$ over $\theta(2^i)$ calls)
- Contrast with always adding k slots
 - Copy $k \cdot i$ elements
 - Get k $\theta(1)$ freebie inserts

ArrayList

- Double the size
 - Copy 2^i elements (Put $\theta(2^i)$ on a credit card)
 - Get $2^i - 1$ $\theta(1)$ freebie inserts (Pay down $\theta(1)$ over $\theta(2^i)$ calls)
- Contrast with always adding k slots
 - Copy $k \cdot i$ elements (Put $k \cdot i$ on a credit card)
 - Get k $\theta(1)$ freebie inserts

ArrayList

- Double the size
 - Copy 2^i elements (Put $\theta(2^i)$ on a credit card)
 - Get $2^i - 1$ $\theta(1)$ freebie inserts (Pay down $\theta(1)$ over $\theta(2^i)$ calls)
- Contrast with always adding k slots
 - Copy $k \cdot i$ elements (Put $k \cdot i$ on a credit card)
 - Get k $\theta(1)$ freebie inserts (Pay down $\theta(1)$ over $\theta(k)$ calls)

ArrayList

- Double the size
 - Copy 2^i elements (Put $\theta(2^i)$ on a credit card)
 - Get $2^i - 1$ $\theta(1)$ freebie inserts (Pay down $\theta(1)$ over $\theta(2^i)$ calls)
- Contrast with always adding k slots
 - Copy $k \cdot i$ elements (Put $k \cdot i$ on a credit card)
 - Get k $\theta(1)$ freebie inserts (Pay down $\theta(1)$ over $\theta(k)$ calls)

ArrayList

- Double the size
 - Copy 2^i elements (Put $\theta(2^i)$ on a credit card)
 - Get $2^i - 1$ $\theta(1)$ freebie inserts (Pay down $\theta(1)$ over $\theta(2^i)$ calls)
- Contrast with always adding k slots
 - Copy $k \cdot i$ elements (Put $k \cdot i$ on a credit card)
 - Get k $\theta(1)$ freebie inserts (Pay down $\theta(1)$ over $\theta(k)$ calls)

When doubling, the time you have to 'pay off' your card grows with the amount that goes on the card.

Amortized Runtime

- The tight unqualified upper bound on $\text{add}(i)$ is $O(N)$
Any one call to $\text{add}(i)$ could take up to $O(N)$.
- The tight amortized upper bound on $\text{add}(i)$ is $O(1)$
 N calls to $\text{add}(i)$ average out to $O(1)$ each.
($O(N)$ for all N calls)

Amortized Runtime

- The tight unqualified upper bound on $\text{add}(i)$ is $O(N)$
Any one call to $\text{add}(i)$ could take up to $O(N)$.
- The tight amortized upper bound on $\text{add}(i)$ is $O(1)$
 N calls to $\text{add}(i)$ average out to $O(1)$ each.
($O(N)$ for all N calls)
(Amortized lets you use a credit card, as long as you pay it off)

Recursion

Recursion



<https://www.etsy.com/listing/916447505/ukrainian-nesting-doll-nesting-dolls>

Recursion



<https://www.etsy.com/listing/916447505/ukrainian-nesting-doll-nesting-dolls>

Recursion



<https://www.etsy.com/listing/916447505/ukrainian-nesting-doll-nesting-dolls>

Recursion



<https://www.etsy.com/listing/916447505/ukrainian-nesting-doll-nesting-dolls>

Recursion



<https://www.etsy.com/listing/916447505/ukrainian-nesting-doll-nesting-dolls>

Recursion



<https://www.etsy.com/listing/916447505/ukrainian-nesting-doll-nesting-dolls>

Recursion



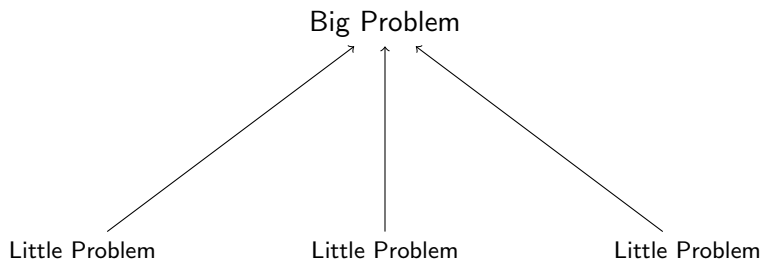
<https://www.etsy.com/listing/916447505/ukrainian-nesting-doll-nesting-dolls>

Recursion

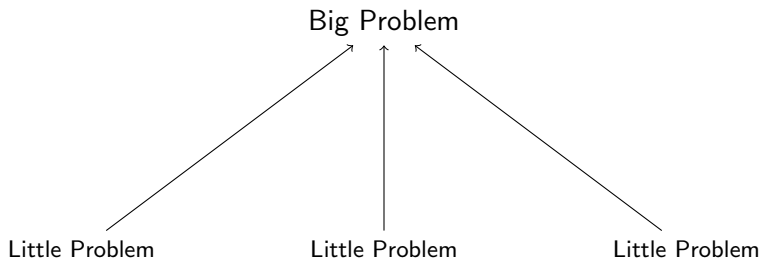


<https://www.etsy.com/listing/916447505/ukrainian-nesting-doll-nesting-dolls>

Algorithms



Algorithms



Recursive Algorithm: When the little problem is the same as the big problem, just smaller.

Factorial

$$439! = 439 \cdot 438 \cdot 437 \cdot 436 \cdot 435 \cdot 434 \cdot 433 \cdot 432 \cdot \dots$$

Factorial

$$439! = 439 \cdot 438 \cdot 437 \cdot 436 \cdot 435 \cdot 434 \cdot 433 \cdot 432 \cdot \dots$$

$$438! = 438 \cdot 437 \cdot 436 \cdot 435 \cdot 434 \cdot 433 \cdot 432 \cdot \dots$$

Factorial

$$\begin{aligned}439! &= 439 \cdot 438 \cdot 437 \cdot 436 \cdot 435 \cdot 434 \cdot 433 \cdot 432 \cdot \dots \\ &= 439 \cdot 438!\end{aligned}$$

$$438! = 438 \cdot 437 \cdot 436 \cdot 435 \cdot 434 \cdot 433 \cdot 432 \cdot \dots$$

Factorial

$$N! = N \cdot (N - 1)!$$

Factorial

$$\underbrace{N!}_{\text{big problem}} = N \cdot \underbrace{(N-1)!}_{\text{smaller (same) problem}}$$

Factorial

$$\underbrace{N!}_{\text{big problem}} = N \cdot \underbrace{(N-1)!}_{\text{smaller (same) problem}}$$

```
1 public long factorial(long N)
2 {
3     return N * factorial(N-1);
4 }
```

Factorial

$$\underbrace{N!}_{\text{big problem}} = N \cdot \underbrace{(N-1)!}_{\text{smaller (same) problem}}$$

```
1 public long factorial(long N)
2 {
3     return N * factorial(N-1);
4 }
```

StackOverflowError

Factorial

- $1! = 1$
- $N! = N \cdot (N - 1)!$

Factorial

- $1! = 1$
- $N! = N \cdot (N - 1)!$

Base Case

Factorial

- $1! = 1$
- $N! = N \cdot (N - 1)!$

Base Case

Recursive Case

Factorial

- $1! = 1$

Base Case

- $N! = N \cdot (N - 1)!$

Recursive Case

```
1 public long factorial(long N)
2 {
3     if(N <= 1){ return 1; }
4     else { return N * factorial(N-1); }
5 }
```


Fibonacci

1, 1,

Fibonacci

1, 1, 2,

Fibonacci

1, 1, 2, 3,

Fibonacci

1, 1, 2, 3, 5,

Fibonacci

1, 1, 2, 3, 5, 8,

Fibonacci

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

Fibonacci

- $\text{Fib}(0) = 1$
- $\text{Fib}(1) = 1$
- $\text{Fib}(N) = \text{Fib}(N - 1) + \text{Fib}(N - 2)$

Fibonacci

- $\text{Fib}(0) = 1$ (Base Case)
- $\text{Fib}(1) = 1$ (Base Case)
- $\text{Fib}(N) = \text{Fib}(N - 1) + \text{Fib}(N - 2)$ (Recursive Case)

Fibonacci

- $\text{Fib}(0) = 1$ (Base Case)
- $\text{Fib}(1) = 1$ (Base Case)
- $\text{Fib}(N) = \text{Fib}(N - 1) + \text{Fib}(N - 2)$ (Recursive Case)

```
1 public long fib(long N)
2 {
3     if(n <= 1){ return 1; }
4     else { fib(n-1) + fib(n-2) }
5 }
```

Towers of Hanoi

Live Demo!

Towers of Hanoi

Task: Move n blocks from **A** to **C**

Base Case ($n = 1$)

- 1 Move the Block from **A** to **C**

Base Case ($n \geq 2$)

- 1 Move $n - 1$ blocks from **A** to **B**
- 2 Move the n 'th block from **A** to **C**
- 3 Move $n - 1$ blocks from **B** to **C**

Towers of Hanoi

```
1  public void move(Tower from, Tower to, Tower other, int n)
2  {
3      if(n == 1)
4      {
5          from.moveOneTo(to);
6      }
7      else
8      {
9          move(from, other, to, n-1);
10         from.moveOneTo(to);
11         move(other, to, from, n-1);
12     }
13 }
```

How do we compute the complexity of recursive algorithms?

Factorial

What is the complexity of Factorial?

```
1  public long factorial(long N)
2  {
3      if(N <= 1){ return 1; }
4      else { return N * factorial(N-1); }
5  }
```

Factorial

What is the complexity of Factorial?

```
1  public long factorial(long N)
2  {
3      if(N <= 1){ return 1; }
4      else { return N * factorial(N-1); }
5  }
```

$$T_{factorial}(N) = \begin{cases} \theta(1) & \text{if } N \leq 1 \\ \theta(1) + ??? & \text{otherwise} \end{cases}$$

Factorial

What is the complexity of Factorial?

```
1 public long factorial(long N)
2 {
3     if(N <= 1){ return 1; }
4     else { return N * factorial(N-1); }
5 }
```

$$T_{factorial}(N) = \begin{cases} \theta(1) & \text{if } N \leq 1 \\ \theta(1) + T_{factorial}(N - 1) & \text{otherwise} \end{cases}$$

Factorial

What is the complexity of Factorial?

```
1  public long factorial(long N)
2  {
3      if(N <= 1){ return 1; }
4      else { return N * factorial(N-1); }
5  }
```

$$T_{factorial}(N) = \begin{cases} \theta(1) & \text{if } N \leq 1 \\ \theta(1) + T_{factorial}(N - 1) & \text{otherwise} \end{cases}$$

Runtime growth functions have base and recursive cases too.

Factorial

$$T_{\text{factorial}}(N) = \begin{cases} \theta(1) & \text{if } N \leq 1 \\ \theta(1) + T_{\text{factorial}}(N - 1) & \text{otherwise} \end{cases}$$

Solve for $T_{\text{factorial}}(N)$.

Induction

Solve for $T_{factorial}(N)$.

Induction

- 1 Generate a hypothesis.
- 2 Prove the hypothesis for the base case.
- 3 Prove the hypothesis inductively.

Generate a Hypothesis

Hypothesis: Solve for increasing values of N

| N | 1 | 2 | 3 | 4 | 5 | 6 |
|--------|---|---|---|---|---|---|
| $T(N)$ | | | | | | |

Generate a Hypothesis

Hypothesis: Solve for increasing values of N

| N | 1 | 2 | 3 | 4 | 5 | 6 |
|--------|---------------------|---|---|---|---|---|
| $T(N)$ | $1 \cdot \theta(1)$ | | | | | |

Generate a Hypothesis

Hypothesis: Solve for increasing values of N

| N | 1 | 2 | 3 | 4 | 5 | 6 |
|--------|---------------------|---------------------|---|---|---|---|
| $T(N)$ | $1 \cdot \theta(1)$ | $2 \cdot \theta(1)$ | | | | |

Generate a Hypothesis

Hypothesis: Solve for increasing values of N

| N | 1 | 2 | 3 | 4 | 5 | 6 |
|--------|---------------------|---------------------|---------------------|---|---|---|
| $T(N)$ | $1 \cdot \theta(1)$ | $2 \cdot \theta(1)$ | $3 \cdot \theta(1)$ | | | |

Generate a Hypothesis

Hypothesis: Solve for increasing values of N

| N | 1 | 2 | 3 | 4 | 5 | 6 |
|--------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|
| $T(N)$ | $1 \cdot \theta(1)$ | $2 \cdot \theta(1)$ | $3 \cdot \theta(1)$ | $4 \cdot \theta(1)$ | $5 \cdot \theta(1)$ | $6 \cdot \theta(1)$ |

Generate a Hypothesis

Hypothesis: Solve for increasing values of N

| N | 1 | 2 | 3 | 4 | 5 | 6 |
|--------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|
| $T(N)$ | $1 \cdot \theta(1)$ | $2 \cdot \theta(1)$ | $3 \cdot \theta(1)$ | $4 \cdot \theta(1)$ | $5 \cdot \theta(1)$ | $6 \cdot \theta(1)$ |

What's the pattern?

Generate a Hypothesis

Hypothesis: Solve for increasing values of N

| N | 1 | 2 | 3 | 4 | 5 | 6 |
|--------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|
| $T(N)$ | $1 \cdot \theta(1)$ | $2 \cdot \theta(1)$ | $3 \cdot \theta(1)$ | $4 \cdot \theta(1)$ | $5 \cdot \theta(1)$ | $6 \cdot \theta(1)$ |

What's the pattern? ($N \cdot \theta(1)$)

Generate a Hypothesis

Hypothesis: Solve for increasing values of N

| N | 1 | 2 | 3 | 4 | 5 | 6 |
|--------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|
| $T(N)$ | $1 \cdot \theta(1)$ | $2 \cdot \theta(1)$ | $3 \cdot \theta(1)$ | $4 \cdot \theta(1)$ | $5 \cdot \theta(1)$ | $6 \cdot \theta(1)$ |

What's the pattern? ($N \cdot \theta(1)$)

Hypothesis: $T(N) \in \theta(N)$

Generate a Hypothesis

Hypothesis: Solve for increasing values of N

| N | 1 | 2 | 3 | 4 | 5 | 6 |
|--------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|
| $T(N)$ | $1 \cdot \theta(1)$ | $2 \cdot \theta(1)$ | $3 \cdot \theta(1)$ | $4 \cdot \theta(1)$ | $5 \cdot \theta(1)$ | $6 \cdot \theta(1)$ |

What's the pattern? ($N \cdot \theta(1)$)

Hypothesis: $T(N) \in \theta(N)$

- There is some $c_{high} > 0$ such that $T(n) \leq c_{high} \cdot n$
- There is some $c_{low} > 0$ such that $T(n) \geq c_{low} \cdot n$

Generate a Hypothesis

Hypothesis: Solve for increasing values of N

| N | 1 | 2 | 3 | 4 | 5 | 6 |
|--------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|
| $T(N)$ | $1 \cdot \theta(1)$ | $2 \cdot \theta(1)$ | $3 \cdot \theta(1)$ | $4 \cdot \theta(1)$ | $5 \cdot \theta(1)$ | $6 \cdot \theta(1)$ |

What's the pattern? ($N \cdot \theta(1)$)

Hypothesis: $T(N) \in \theta(N)$

- There is some $c_{high} > 0$ such that $T(n) \leq c_{high} \cdot n$ ←
- There is some $c_{low} > 0$ such that $T(n) \geq c_{low} \cdot n$

Algebra with θ

Remember, $\theta(N)$ used in a math equation is shorthand for:
 $f(N)$ where $f(N) \in \theta(N)$

Algebra with θ

Remember, $\theta(N)$ used in a math equation is shorthand for:
 $f(N)$ where $f(N) \in \theta(N)$

So $\theta(1)$ is shorthand for some constant c .

Factorial

$$T_{\text{factorial}}(N) = \begin{cases} \theta(1) & \text{if } N \leq 1 \\ \theta(1) + T_{\text{factorial}}(N - 1) & \text{otherwise} \end{cases}$$

Factorial

$$T_{\text{factorial}}(N) = \begin{cases} c_1 & \text{if } N \leq 1 \\ c_2 + T_{\text{factorial}}(N - 1) & \text{otherwise} \end{cases}$$

Factorial

$$T_{\text{factorial}}(N) = \begin{cases} c_1 & \text{if } N \leq 1 \\ c_2 + T_{\text{factorial}}(N - 1) & \text{otherwise} \end{cases}$$

Goal: Show that $T_{\text{factorial}}(N) \leq c \cdot N$ for some $c > 0$

Factorial

$$T_{\text{factorial}}(N) = \begin{cases} c_1 & \text{if } N \leq 1 \\ c_2 + T_{\text{factorial}}(N - 1) & \text{otherwise} \end{cases}$$

Goal: Show that $T_{\text{factorial}}(N) \leq c \cdot N$ for some $c > 0$ ($N > N_0$)

Factorial

$$T_{\text{factorial}}(N) = \begin{cases} c_1 & \text{if } N \leq 1 \\ c_2 + T_{\text{factorial}}(N - 1) & \text{otherwise} \end{cases}$$

Goal: Show that $T_{\text{factorial}}(N) \leq c \cdot N$ for some $c > 0$ ($N > N_0$)

$$T_{\text{factorial}}(1) \stackrel{?}{\leq} 1 \cdot c$$

Factorial

$$T_{\text{factorial}}(N) = \begin{cases} c_1 & \text{if } N \leq 1 \\ c_2 + T_{\text{factorial}}(N - 1) & \text{otherwise} \end{cases}$$

Goal: Show that $T_{\text{factorial}}(N) \leq c \cdot N$ for some $c > 0$ ($N > N_0$)

$$T_{\text{factorial}}(1) \stackrel{?}{\leq} 1 \cdot c$$
$$c_1 \stackrel{?}{\leq} 1 \cdot c$$

Factorial

$$T_{\text{factorial}}(N) = \begin{cases} c_1 & \text{if } N \leq 1 \\ c_2 + T_{\text{factorial}}(N - 1) & \text{otherwise} \end{cases}$$

Goal: Show that $T_{\text{factorial}}(N) \leq c \cdot N$ for some $c > 0$ ($N > N_0$)

$$T_{\text{factorial}}(1) \stackrel{?}{\leq} 1 \cdot c$$
$$c_1 \stackrel{?}{\leq} 1 \cdot c \checkmark$$

Factorial

$$T_{\text{factorial}}(N) = \begin{cases} c_1 & \text{if } N \leq 1 \\ c_2 + T_{\text{factorial}}(N - 1) & \text{otherwise} \end{cases}$$

Goal: Show that $T_{\text{factorial}}(N) \leq c \cdot N$ for some $c > 0$ ($N > N_0$)

$$T_{\text{factorial}}(2) \stackrel{?}{\leq} 2 \cdot c$$

Factorial

$$T_{\text{factorial}}(N) = \begin{cases} c_1 & \text{if } N \leq 1 \\ c_2 + T_{\text{factorial}}(N - 1) & \text{otherwise} \end{cases}$$

Goal: Show that $T_{\text{factorial}}(N) \leq c \cdot N$ for some $c > 0$ ($N > N_0$)

$$T_{\text{factorial}}(2) \stackrel{?}{\leq} 2 \cdot c$$

$$c_2 + T_{\text{factorial}}(1) \stackrel{?}{\leq} 2 \cdot c$$

Factorial

$$T_{\text{factorial}}(N) = \begin{cases} c_1 & \text{if } N \leq 1 \\ c_2 + T_{\text{factorial}}(N - 1) & \text{otherwise} \end{cases}$$

Goal: Show that $T_{\text{factorial}}(N) \leq c \cdot N$ for some $c > 0$ ($N > N_0$)

$$\begin{aligned} T_{\text{factorial}}(2) &\stackrel{?}{\leq} 2 \cdot c \\ c_2 + T_{\text{factorial}}(1) &\stackrel{?}{\leq} 2 \cdot c \\ c_2 + c_1 &\stackrel{?}{\leq} 2 \cdot c \end{aligned}$$

Factorial

$$T_{\text{factorial}}(N) = \begin{cases} c_1 & \text{if } N \leq 1 \\ c_2 + T_{\text{factorial}}(N - 1) & \text{otherwise} \end{cases}$$

Goal: Show that $T_{\text{factorial}}(N) \leq c \cdot N$ for some $c > 0$ ($N > N_0$)

$$\begin{aligned} T_{\text{factorial}}(2) &\stackrel{?}{\leq} 2 \cdot c \\ c_2 + T_{\text{factorial}}(1) &\stackrel{?}{\leq} 2 \cdot c \\ c_2 + c_1 &\stackrel{?}{\leq} 2 \cdot c \checkmark \end{aligned}$$

Factorial

$$T_{\text{factorial}}(N) = \begin{cases} c_1 & \text{if } N \leq 1 \\ c_2 + T_{\text{factorial}}(N - 1) & \text{otherwise} \end{cases}$$

Goal: Show that $T_{\text{factorial}}(N) \leq c \cdot N$ for some $c > 0$ ($N > N_0$)

$$T_{\text{factorial}}(2) \stackrel{?}{\leq} 2 \cdot c$$

$$c_2 + T_{\text{factorial}}(1) \stackrel{?}{\leq} 2 \cdot c$$

Factorial

$$T_{\text{factorial}}(N) = \begin{cases} c_1 & \text{if } N \leq 1 \\ c_2 + T_{\text{factorial}}(N - 1) & \text{otherwise} \end{cases}$$

Goal: Show that $T_{\text{factorial}}(N) \leq c \cdot N$ for some $c > 0$ ($N > N_0$)

$$T_{\text{factorial}}(2) \stackrel{?}{\leq} 2 \cdot c$$

$$c_2 + T_{\text{factorial}}(1) \stackrel{?}{\leq} 2 \cdot c$$

$$c_2 + T_{\text{factorial}}(1) \stackrel{?}{\leq} c + c$$

Factorial

$$T_{\text{factorial}}(N) = \begin{cases} c_1 & \text{if } N \leq 1 \\ c_2 + T_{\text{factorial}}(N-1) & \text{otherwise} \end{cases}$$

Goal: Show that $T_{\text{factorial}}(N) \leq c \cdot N$ for some $c > 0$ ($N > N_0$)

$$\begin{aligned} T_{\text{factorial}}(2) &\stackrel{?}{\leq} 2 \cdot c \\ c_2 + T_{\text{factorial}}(1) &\stackrel{?}{\leq} 2 \cdot c \\ c_2 + T_{\text{factorial}}(1) &\stackrel{?}{\leq} c + c \\ &\stackrel{?}{\leq} c \end{aligned}$$

Factorial

$$T_{\text{factorial}}(N) = \begin{cases} c_1 & \text{if } N \leq 1 \\ c_2 + T_{\text{factorial}}(N-1) & \text{otherwise} \end{cases}$$

Goal: Show that $T_{\text{factorial}}(N) \leq c \cdot N$ for some $c > 0$ ($N > N_0$)

$$\begin{aligned} T_{\text{factorial}}(2) &\stackrel{?}{\leq} 2 \cdot c \\ c_2 + T_{\text{factorial}}(1) &\stackrel{?}{\leq} 2 \cdot c \\ c_2 + T_{\text{factorial}}(1) &\stackrel{?}{\leq} c + c \\ c_2 &\stackrel{?}{\leq} c \checkmark \end{aligned}$$

Factorial

$$T_{\text{factorial}}(N) = \begin{cases} c_1 & \text{if } N \leq 1 \\ c_2 + T_{\text{factorial}}(N - 1) & \text{otherwise} \end{cases}$$

Goal: Show that $T_{\text{factorial}}(N) \leq c \cdot N$ for some $c > 0$ ($N > N_0$)

$$T_{\text{factorial}}(3) \stackrel{?}{\leq} 3 \cdot c$$

$$c_2 + T_{\text{factorial}}(2) \stackrel{?}{\leq} 3 \cdot c$$

Factorial

$$T_{\text{factorial}}(N) = \begin{cases} c_1 & \text{if } N \leq 1 \\ c_2 + T_{\text{factorial}}(N - 1) & \text{otherwise} \end{cases}$$

Goal: Show that $T_{\text{factorial}}(N) \leq c \cdot N$ for some $c > 0$ ($N > N_0$)

$$T_{\text{factorial}}(3) \stackrel{?}{\leq} 3 \cdot c$$

$$c_2 + T_{\text{factorial}}(2) \stackrel{?}{\leq} 3 \cdot c$$

$$c_2 + T_{\text{factorial}}(2) \stackrel{?}{\leq} c + 2c$$

Factorial

$$T_{\text{factorial}}(N) = \begin{cases} c_1 & \text{if } N \leq 1 \\ c_2 + T_{\text{factorial}}(N - 1) & \text{otherwise} \end{cases}$$

Goal: Show that $T_{\text{factorial}}(N) \leq c \cdot N$ for some $c > 0$ ($N > N_0$)

$$T_{\text{factorial}}(3) \stackrel{?}{\leq} 3 \cdot c$$

$$c_2 + T_{\text{factorial}}(2) \stackrel{?}{\leq} 3 \cdot c$$

$$c_2 + T_{\text{factorial}}(2) \stackrel{?}{\leq} c + 2c$$

$$c_2 \stackrel{?}{\leq} c$$

Factorial

$$T_{\text{factorial}}(N) = \begin{cases} c_1 & \text{if } N \leq 1 \\ c_2 + T_{\text{factorial}}(N - 1) & \text{otherwise} \end{cases}$$

Goal: Show that $T_{\text{factorial}}(N) \leq c \cdot N$ for some $c > 0$ ($N > N_0$)

$$T_{\text{factorial}}(3) \stackrel{?}{\leq} 3 \cdot c$$

$$c_2 + T_{\text{factorial}}(2) \stackrel{?}{\leq} 3 \cdot c$$

$$c_2 + T_{\text{factorial}}(2) \stackrel{?}{\leq} c + 2c$$

$$c_2 \stackrel{?}{\leq} c \checkmark$$

Factorial

This is boring!

Factorial

This is boring!

Can't I automate the proof?

Induction

- 1 Generate a hypothesis.
- 2 Prove the hypothesis for the base case.
- 3 Prove the hypothesis inductively.

Induction

- 1 Generate a hypothesis.
- 2 Prove the hypothesis for the base case.
- 3 Prove the hypothesis inductively.

$$T(N) \in O(N)$$

Induction

- 1 Generate a hypothesis.
- 2 Prove the hypothesis for the base case.
- 3 Prove the hypothesis inductively.

$$T(N) \in O(N)$$

$$\exists c : T(N) \leq c \cdot N$$

Induction

- 1 Generate a hypothesis.
- 2 Prove the hypothesis for the base case.
- 3 Prove the hypothesis inductively.

$$T(N) \in O(N)$$

$$\exists c : T(N) \leq c \cdot N$$

...

Induction

- 1 Generate a hypothesis.
- 2 Prove the hypothesis for the base case.
- 3 Prove the hypothesis inductively.
 - Assume you've proved it for case $N - 1$
 - Prove that it holds for case N

$$T(N) \in O(N)$$

$$\exists c : T(N) \leq c \cdot N$$

Factorial

$$T_{\text{factorial}}(N) = \begin{cases} c_1 & \text{if } N \leq 1 \\ c_2 + T_{\text{factorial}}(N - 1) & \text{otherwise} \end{cases}$$

Goal: Show that $T_{\text{factorial}}(N) \leq c \cdot N$ for some $c > 0$ ($N > N_0$)

Factorial

$$T_{\text{factorial}}(N) = \begin{cases} c_1 & \text{if } N \leq 1 \\ c_2 + T_{\text{factorial}}(N - 1) & \text{otherwise} \end{cases}$$

Goal: Show that $T_{\text{factorial}}(N) \leq c \cdot N$ for some $c > 0$ ($N > N_0$)

Assume: $T_{\text{factorial}}(N - 1) \leq c \cdot N - 1$

$$T_{\text{factorial}}(N) \stackrel{?}{\leq} N \cdot c$$

$$c_2 + T_{\text{factorial}}(N - 1) \stackrel{?}{\leq} N \cdot c$$

Factorial

$$T_{\text{factorial}}(N) = \begin{cases} c_1 & \text{if } N \leq 1 \\ c_2 + T_{\text{factorial}}(N - 1) & \text{otherwise} \end{cases}$$

Goal: Show that $T_{\text{factorial}}(N) \leq c \cdot N$ for some $c > 0$ ($N > N_0$)

Assume: $T_{\text{factorial}}(N - 1) \leq c \cdot N - 1$

$$T_{\text{factorial}}(N) \stackrel{?}{\leq} N \cdot c$$

$$c_2 + T_{\text{factorial}}(N - 1) \stackrel{?}{\leq} N \cdot c$$

$$c_2 + T_{\text{factorial}}(N - 1) \stackrel{?}{\leq} c + (N - 1)c$$

Factorial

$$T_{\text{factorial}}(N) = \begin{cases} c_1 & \text{if } N \leq 1 \\ c_2 + T_{\text{factorial}}(N - 1) & \text{otherwise} \end{cases}$$

Goal: Show that $T_{\text{factorial}}(N) \leq c \cdot N$ for some $c > 0$ ($N > N_0$)

Assume: $T_{\text{factorial}}(N - 1) \leq c \cdot N - 1$

$$\begin{aligned} T_{\text{factorial}}(N) &\stackrel{?}{\leq} N \cdot c \\ c_2 + T_{\text{factorial}}(N - 1) &\stackrel{?}{\leq} N \cdot c \\ c_2 + T_{\text{factorial}}(N - 1) &\stackrel{?}{\leq} c + (N - 1)c \\ &\stackrel{?}{\leq} c \end{aligned}$$

Factorial

$$T_{\text{factorial}}(N) = \begin{cases} c_1 & \text{if } N \leq 1 \\ c_2 + T_{\text{factorial}}(N-1) & \text{otherwise} \end{cases}$$

Goal: Show that $T_{\text{factorial}}(N) \leq c \cdot N$ for some $c > 0$ ($N > N_0$)

Assume: $T_{\text{factorial}}(N-1) \leq c \cdot N - 1$

$$\begin{aligned} T_{\text{factorial}}(N) &\stackrel{?}{\leq} N \cdot c \\ c_2 + T_{\text{factorial}}(N-1) &\stackrel{?}{\leq} N \cdot c \\ c_2 + T_{\text{factorial}}(N-1) &\stackrel{?}{\leq} c + (N-1)c \\ &\stackrel{?}{\leq} c \checkmark \end{aligned}$$

Induction

We showed there exists a c such that...

- $T(1) \leq 1 \cdot c$ (Base Case)
- if $T(N - 1) \leq (N - 1) \cdot c$ then $T(N) \leq N \cdot c$ (Inductive Proof)

Induction

We showed there exists a c such that...

- $T(1) \leq 1 \cdot c$ (Base Case)
- if $T(N - 1) \leq (N - 1) \cdot c$ then $T(N) \leq N \cdot c$ (Inductive Proof)

So...

- 1 $T(1) \leq 1 \cdot c$ (Base Case)

Induction

We showed there exists a c such that...

- $T(1) \leq 1 \cdot c$ (Base Case)
- if $T(N - 1) \leq (N - 1) \cdot c$ then $T(N) \leq N \cdot c$ (Inductive Proof)

So...

- 1 $T(1) \leq 1 \cdot c$ (Base Case)
- 2 $T(2) \leq 2 \cdot c$ (#1 + Inductive Proof)

Induction

We showed there exists a c such that...

- $T(1) \leq 1 \cdot c$ (Base Case)
- if $T(N - 1) \leq (N - 1) \cdot c$ then $T(N) \leq N \cdot c$ (Inductive Proof)

So...

- 1 $T(1) \leq 1 \cdot c$ (Base Case)
- 2 $T(2) \leq 2 \cdot c$ (#1 + Inductive Proof)
- 3 $T(3) \leq 3 \cdot c$ (#2 + Inductive Proof)

Induction

We showed there exists a c such that...

- $T(1) \leq 1 \cdot c$ (Base Case)
- if $T(N - 1) \leq (N - 1) \cdot c$ then $T(N) \leq N \cdot c$ (Inductive Proof)

So...

- 1 $T(1) \leq 1 \cdot c$ (Base Case)
- 2 $T(2) \leq 2 \cdot c$ (#1 + Inductive Proof)
- 3 $T(3) \leq 3 \cdot c$ (#2 + Inductive Proof)
- 4 $T(4) \leq 4 \cdot c$ (#3 + Inductive Proof)
- 5 $T(5) \leq 5 \cdot c$ (#4 + Inductive Proof)
- 6 $T(6) \leq 6 \cdot c$ (#5 + Inductive Proof)
- 7 ...

Induction

We showed there exists a c such that...

- $T(1) \leq 1 \cdot c$ (Base Case)
- if $T(N - 1) \leq (N - 1) \cdot c$ then $T(N) \leq N \cdot c$ (Inductive Proof)

So...

- 1 $T(1) \leq 1 \cdot c$ (Base Case)
- 2 $T(2) \leq 2 \cdot c$ (#1 + Inductive Proof)
- 3 $T(3) \leq 3 \cdot c$ (#2 + Inductive Proof)
- 4 $T(4) \leq 4 \cdot c$ (#3 + Inductive Proof)
- 5 $T(5) \leq 5 \cdot c$ (#4 + Inductive Proof)
- 6 $T(6) \leq 6 \cdot c$ (#5 + Inductive Proof)
- 7 ...

The proof holds for any $N \geq 1$

Induction

We showed there exists a c such that...

- $T(1) \leq 1 \cdot c$ (Base Case)
- if $T(N - 1) \leq (N - 1) \cdot c$ then $T(N) \leq N \cdot c$ (Inductive Proof)

So...

- 1 $T(1) \leq 1 \cdot c$ (Base Case)
- 2 $T(2) \leq 2 \cdot c$ (#1 + Inductive Proof)
- 3 $T(3) \leq 3 \cdot c$ (#2 + Inductive Proof)
- 4 $T(4) \leq 4 \cdot c$ (#3 + Inductive Proof)
- 5 $T(5) \leq 5 \cdot c$ (#4 + Inductive Proof)
- 6 $T(6) \leq 6 \cdot c$ (#5 + Inductive Proof)
- 7 ...

The proof holds for any $N \geq 1 \rightarrow T(N) \in O(N)$ ✓

Factorial

What is the complexity of Factorial?

```
1  public long factorial(long N)
2  {
3      if(N <= 1){ return 1; }
4      else { return N * factorial(N-1); }
5  }
```

Answer: $O(N)$

Factorial

What is the complexity of Factorial?

```
1 public long factorial(long N)
2 {
3     if(N <= 1){ return 1; }
4     else { return N * factorial(N-1); }
5 }
```

Answer: $O(N)^1$

¹Technically it's $\theta(N)$, but we haven't proven $T(N) \in \Omega(N)$

Factorial

What is the complexity of Factorial?

```
1  public long factorial(long N)
2  {
3      if(N <= 1){ return 1; }
4      else { return N * factorial(N-1); }
5  }
```

Answer: $O(N)^1$

How much memory does it use?

¹Technically it's $\theta(N)$, but we haven't proven $T(N) \in \Omega(N)$

Stack Frames

Every time you call a function, it allocates some memory for local variables (e.g., N).

This chunk of memory is called a **Stack Frame**.

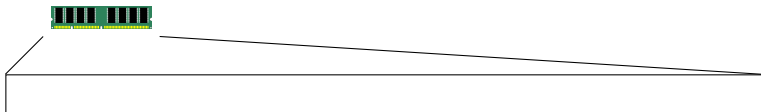
Stack Frames

Every time you call a function, it allocates some memory for local variables (e.g., N).

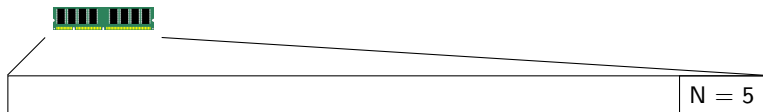
This chunk of memory is called a **Stack Frame**.

This is where the term `StackOverflowError` comes from.

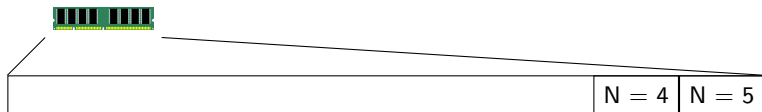
Stack Frames



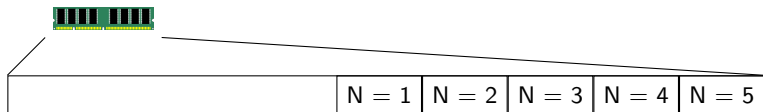
Stack Frames



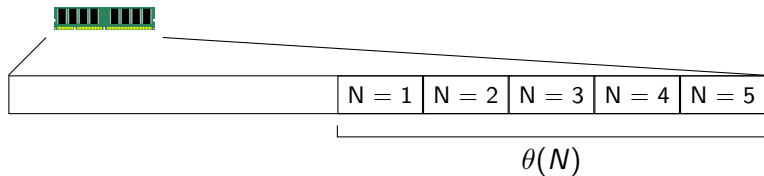
Stack Frames



Stack Frames



Stack Frames



Factorial (as a loop)

```
1  public long factorial(long N)
2  {
3      long total = 1;
4      for(long i = N; i > 0; i--)
5      {
6          total *= i
7      }
8      return total
9  }
```


Factorial

Why does this work?

Factorial (as a loop)

```
1  public long factorial(long N)
2  {
3      if(N <= 1){ return 1; }
4      else { return N * factorial(N-1); }
5  }
```

Factorial (as a loop)

```
1  public long factorial(long N)
2  {
3      if(N <= 1){ return 1; }
4      else { return N * factorial(N-1); }
5  }
```

Each call to factorial only makes **one** recursive call.

Factorial (as a loop)

- Is $N > 1$?
- Compute $\text{arg} = N - 1$
- Call `factorial(arg)`
- Compute $N \times \text{result}$
- Return

Factorial (as a loop)

- Is $N > 1$? ← Requires stack frame
- Compute $\text{arg} = N - 1$ ← Requires stack frame
- Call `factorial(arg)`
- Compute $N \times \text{result}$ ← Requires stack frame
- Return

Factorial (as a loop)

```
1  public long factorial(long N, long total)
2  {
3      if(N <= 1){ return total; }
4      else { return factorial(N-1, N * total); }
5  }
```

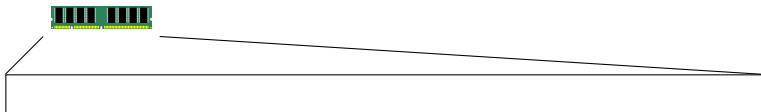
Factorial (as a loop)

- Is $N > 1$?
- Compute $\text{arg1} = N - 1$
- Compute $\text{arg2} = N \times \text{total}$
- Call `factorial(arg1, arg2)`
- Return

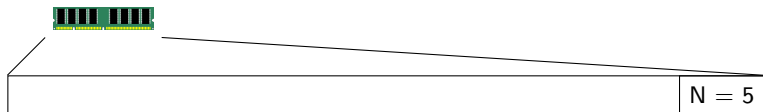
Factorial (as a loop)

- Is $N > 1$? ← Requires stack frame
- Compute $\text{arg1} = N - 1$ ← Requires stack frame
- Compute $\text{arg2} = N \times \text{total}$ ← Requires stack frame
- Call `factorial(arg1, arg2)`
- Return ← Stack frame unnecessary

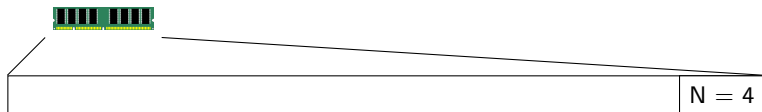
Stack Frames



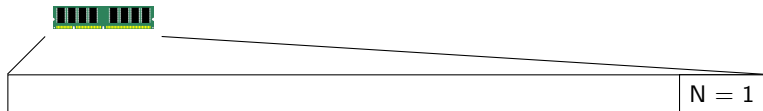
Stack Frames



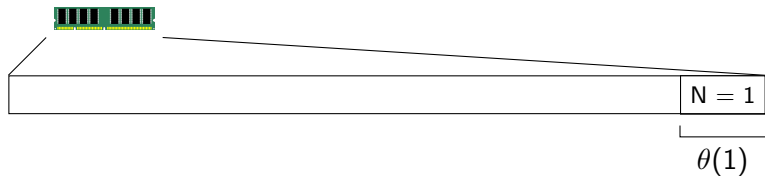
Stack Frames



Stack Frames



Stack Frames



Tail Recursion

If the recursive call is the last operation before the return, most languages optimize the recursion away.

Tail Recursion

If the recursive call is the last operation before the return, most languages optimize the recursion away².

²... but not Java

Tail Recursion

If the recursive call is the last operation before the return, most languages optimize the recursion away².

This is called **Tail Recursion**

²... but not Java

Fibonacci

Time permitting...

Fibonacci

What's the complexity:

```
1  public long fib(long N)
2  {
3      if(n <= 1){ return 1; }
4      else { fib(n-1) + fib(n-2) }
5  }
```