CSE 250: Induction

Lecture 12

Sept 23, 2024

Reminders

- WA2 due Sun, Sept 29 at 11:59 PM
- Midterm 1 in class on Fri, Oct 04.
  - Covers: Asymptotics, Sequences/Lists, Arrays, Linked Lists, Recursion
  - Bounds: Tight Upper/Lower, Unqualified vs Amortized

Sorted Lists

Finding an item

- **Regular List**: Sequential Scan
- **Sorted List**: Binary Search

Sorted Lists

Finding an item

- **Regular List**: Sequential Scan $O(N)$
- **Sorted List**: Binary Search

Sorted Lists

Finding an item

- **Regular List**: Sequential Scan — $O(N)$
- **Sorted List**: Binary Search — $\theta(log(N))$ calls to `get()`

Sorted Lists

Finding an item

- **Regular List**: Sequential Scan $O(N)$
- **Sorted List**: Binary Search $\theta(log(N))$ calls to `get()`

So how do we get a sorted list?

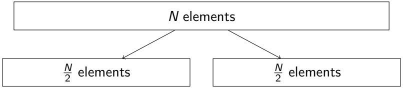
Bubble Sort

$O(N^2)$

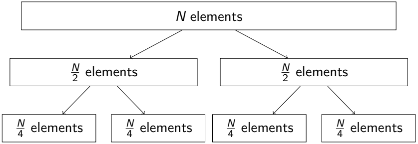
Merge Sort

N elements

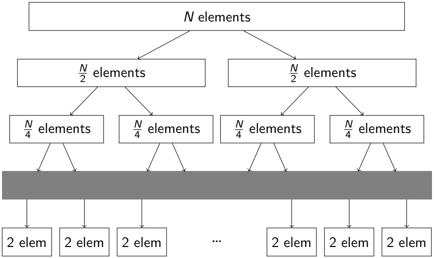
Merge Sort



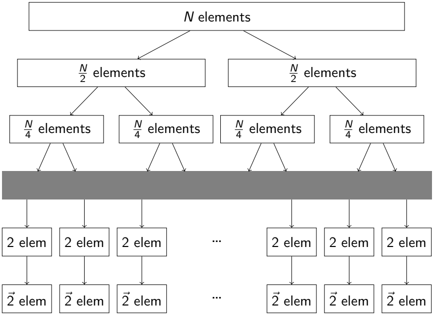
Merge Sort


N elements
$\frac{N}{2}$ elements
$\frac{N}{2}$ elements
$\frac{N}{4}$ elements
$\frac{N}{4}$ elements
$\frac{N}{4}$ elements
$\frac{N}{4}$ elements

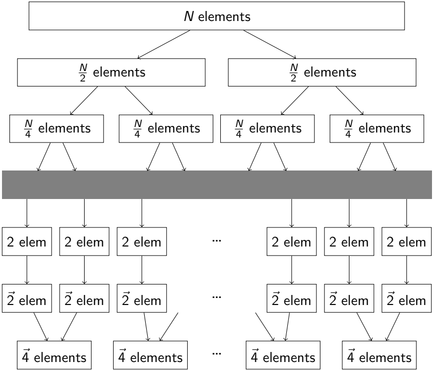
Merge Sort


N elements
$\frac{N}{2}$ elements
$\frac{N}{2}$ elements
$\frac{N}{4}$ elements
$\frac{N}{4}$ elements
$\frac{N}{4}$ elements
$\frac{N}{4}$ elements
2 elem
2 elem
2 elem
...
2 elem
2 elem
2 elem

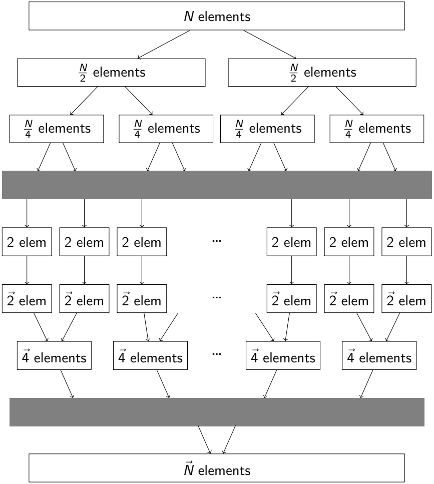
Merge Sort


N elements
N/2 elements
N/2 elements
N/4 elements
N/4 elements
N/4 elements
N/4 elements
2 elem
2 elem
2 elem
...
2 elem
2 elem
2 elem
2⃗ elem
2⃗ elem
2⃗ elem
...
2⃗ elem
2⃗ elem
2⃗ elem

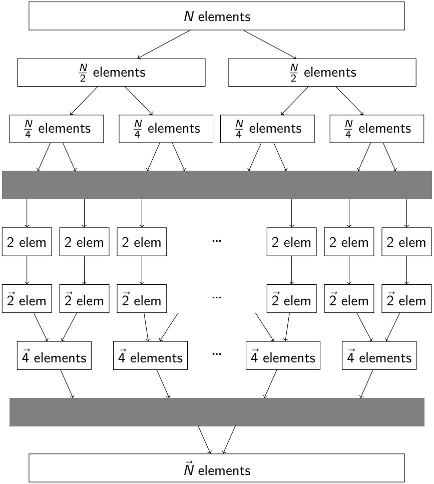
Merge Sort


N elements
N/2 elements
N/2 elements
N/4 elements
N/4 elements
N/4 elements
N/4 elements
2 elem
2 elem
2 elem
...
2 elem
2 elem
2 elem
2⃗ elem
2⃗ elem
2⃗ elem
...
2⃗ elem
2⃗ elem
2⃗ elem
4⃗ elements
4⃗ elements
...
4⃗ elements
4⃗ elements

Merge Sort

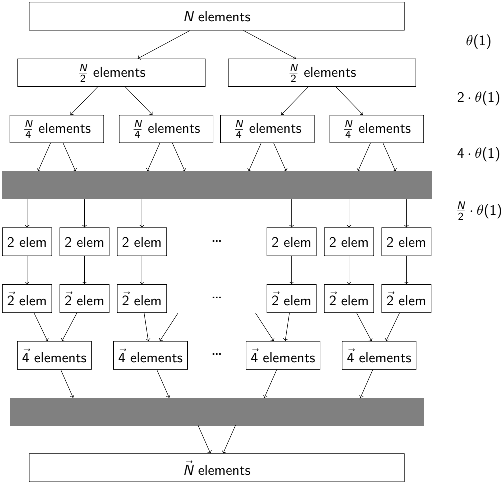

N elements
N/2 elements
N/2 elements
N/4 elements
N/4 elements
N/4 elements
N/4 elements
2 elem
2 elem
2 elem
...
2 elem
2 elem
2 elem
2⃗ elem
2⃗ elem
2⃗ elem
...
2⃗ elem
2⃗ elem
2⃗ elem
4⃗ elements
4⃗ elements
...
4⃗ elements
4⃗ elements
N⃗ elements

Merge Sort

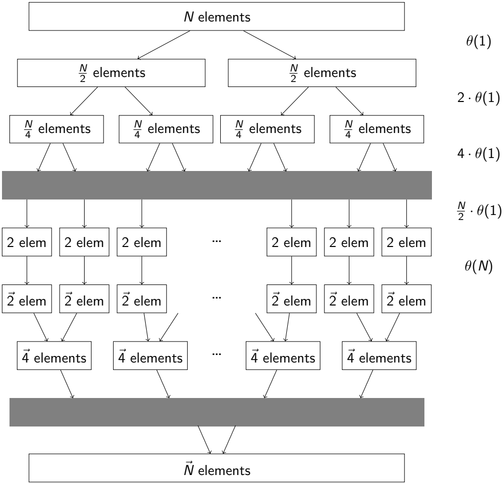


$\theta(1)$

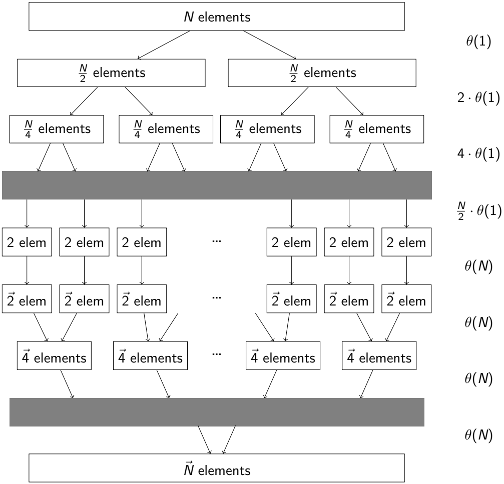
Merge Sort


N elements
N/2 elements
N/2 elements
N/4 elements
N/4 elements
N/4 elements
N/4 elements
2 elem
2 elem
2 elem
...
2 elem
2 elem
2 elem
2⃗ elem
2⃗ elem
2⃗ elem
...
2⃗ elem
2⃗ elem
2⃗ elem
4⃗ elements
4⃗ elements
...
4⃗ elements
4⃗ elements
N⃗ elements
θ(1)
2 · θ(1)
4 · θ(1)
N/2 · θ(1)

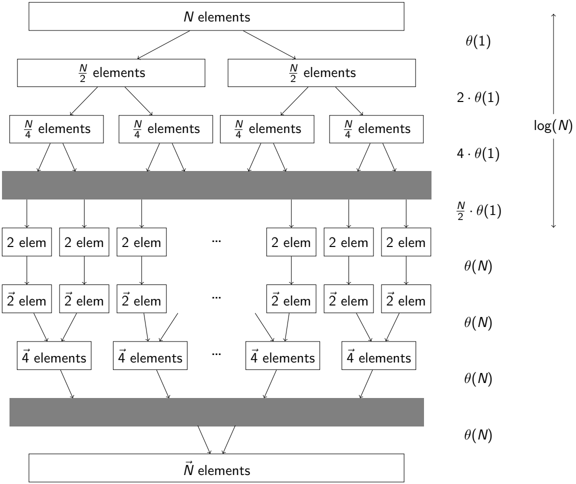
Merge Sort


N elements
N/2 elements
N/2 elements
N/4 elements
N/4 elements
N/4 elements
N/4 elements
2 elem
2 elem
2 elem
...
2 elem
2 elem
2 elem
2⃗ elem
2⃗ elem
2⃗ elem
...
2⃗ elem
2⃗ elem
2⃗ elem
4⃗ elements
4⃗ elements
...
4⃗ elements
4⃗ elements
N⃗ elements
θ(1)
2 · θ(1)
4 · θ(1)
N/2 · θ(1)
θ(N)

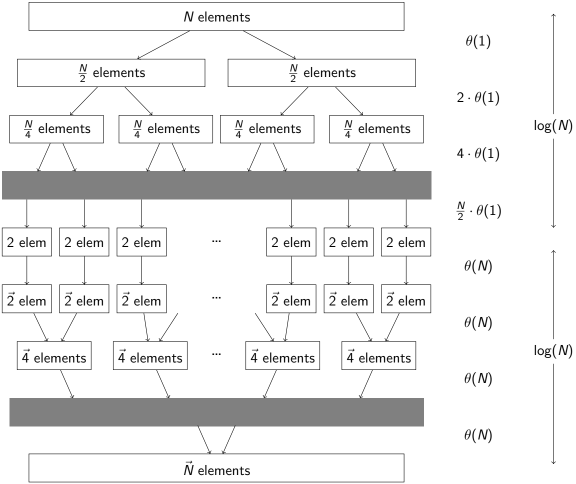
Merge Sort


N elements
N/2 elements
N/2 elements
N/4 elements
N/4 elements
N/4 elements
N/4 elements
2 elem
2 elem
2 elem
...
2 elem
2 elem
2 elem
2⃗ elem
2⃗ elem
2⃗ elem
...
2⃗ elem
2⃗ elem
2⃗ elem
4⃗ elements
4⃗ elements
...
4⃗ elements
4⃗ elements
N⃗ elements
θ(1)
2 · θ(1)
4 · θ(1)
N/2 · θ(1)
θ(N)
θ(N)
θ(N)
θ(N)

Merge Sort


N elements
$\frac{N}{2}$ elements
$\frac{N}{2}$ elements
$\frac{N}{4}$ elements
$\frac{N}{4}$ elements
$\frac{N}{4}$ elements
$\frac{N}{4}$ elements
2 elem
2 elem
2 elem
...
2 elem
2 elem
2 elem
$\vec{2}$ elem
$\vec{2}$ elem
$\vec{2}$ elem
...
$\vec{2}$ elem
$\vec{2}$ elem
$\vec{2}$ elem
$\vec{4}$ elements
$\vec{4}$ elements
...
$\vec{4}$ elements
$\vec{4}$ elements
$\vec{N}$ elements
$\theta(1)$
$2 \cdot \theta(1)$
$4 \cdot \theta(1)$
$\frac{N}{2} \cdot \theta(1)$
$\theta(N)$
$\theta(N)$
$\theta(N)$
$\theta(N)$
$\log(N)$

Merge Sort


N elements
N/2 elements
N/2 elements
N/4 elements
N/4 elements
N/4 elements
N/4 elements
2 elem
2 elem
2 elem
...
2 elem
2 elem
2 elem
$\vec{2}$ elem
$\vec{2}$ elem
$\vec{2}$ elem
...
$\vec{2}$ elem
$\vec{2}$ elem
$\vec{2}$ elem
$\vec{4}$ elements
$\vec{4}$ elements
...
$\vec{4}$ elements
$\vec{4}$ elements
$\vec{N}$ elements
$\theta(1)$
$2 \cdot \theta(1)$
$4 \cdot \theta(1)$
$\frac{N}{2} \cdot \theta(1)$
$\theta(N)$
$\theta(N)$
$\theta(N)$
$\theta(N)$
$\log(N)$
$\log(N)$

Merge Sort

$$\left(\sum_{i=0}^{\log(N)-1} 2^i \cdot \theta(1)\right) + \left(\sum_{i=1}^{\log(N)-1} \theta(N)\right)$$

Merge Sort

$$\left(\sum_{i=0}^{\log(N)-1} 2^i \cdot \theta(1)\right) + \left(\sum_{i=1}^{\log(N)-1} \theta(N)\right)$$

$$\left(2^{\log(N)}\theta(1)\right) + (\log(N)\theta(N))$$

Merge Sort

$$\left(\sum_{i=0}^{\log(N)-1} 2^i \cdot \theta(1)\right) + \left(\sum_{i=1}^{\log(N)-1} \theta(N)\right)$$

$$\left(2^{\log(N)}\theta(1)\right) + (\log(N)\theta(N))$$

$$\theta(N) + \theta(N\log(N))$$

Merge Sort

$$\left(\sum_{i=0}^{\log(N)-1} 2^i \cdot \theta(1)\right) + \left(\sum_{i=1}^{\log(N)-1} \theta(N)\right)$$

$$\left(2^{\log(N)}\theta(1)\right) + (\log(N)\theta(N))$$

$$\theta(N) + \theta(N\log(N))$$

Merge Sort: $\theta(N\log(N))$

Merge Sort

$$\left(\sum_{i=0}^{\log(N)-1} 2^i \cdot \theta(1)\right) + \left(\sum_{i=1}^{\log(N)-1} \theta(N)\right)$$

$$\left(2^{\log(N)}\theta(1)\right) + (\log(N)\theta(N))$$

$$\theta(N) + \theta(N\log(N))$$

Merge Sort: $\theta(N\log(N))$

Bubble Sort: $\theta(N^2)$

Induction

Can we frame this proof in terms of the code?

Merge Sort

```
public ArrayList<E> mergeSort<E>(ArrayList<E> list)
{
  if(list.size() > 2){
    int splitIndex = input.size()/2;
    ArrayList<E> left =
      mergeSort(list.subList(0, splitIndex));
    ArrayList<E> right =
      mergeSort(list.subList(splitIndex, list.size));
    return merge(left, right);
  } else {
    if((list.size() == 2) && (list.get(0) > list.get(1))){
      E tmp = list.get(0);
      list.set(0, list.get(1));
      list.set(1, tmp);
    }
    return list;
  }
}
```

Merge Sort

- If $N > 2$
  - Split
  - $2\times$ `mergeSort(`$\frac{N}{2}$`)`
  - `merge(`N`)`
- Otherwise
  - Sort 2 elements

Merge Sort

- If $N > 2$
  - Split $O(1)$
  - $2\times$ `mergeSort(`$\frac{N}{2}$`)`
  - `merge(`N`)`
- Otherwise
  - Sort 2 elements

Merge Sort

- If $N > 2$
  - Split $O(1)$
  - $2\times$ `mergeSort(`$\frac{N}{2}$`)` ???
  - `merge(`N`)`
- Otherwise
  - Sort 2 elements

Merge Sort

- If $N > 2$
  - Split $O(1)$
  - $2\times$ `mergeSort`$(\frac{N}{2})$???
  - `merge`(N) $O(N)$
- Otherwise
  - Sort 2 elements

Merge Sort

- If $N > 2$
  - Split — $O(1)$
  - $2\times$ `mergeSort(`$\frac{N}{2}$`)` — ???
  - `merge(`N`)` — $O(N)$
- Otherwise
  - Sort 2 elements — $O(1)$

Merge Sort

- If $N > 2$
  - Split $O(1)$
  - $2\times$ `mergeSort(`$\frac{N}{2}$`)` ???
  - `merge(`N`)` $O(N)$
- Otherwise
  - Sort 2 elements $O(1)$

$$T_{merge}(N) = \begin{cases} O(N) + 2 \cdot ??? & \textbf{if } N > 2 \\ O(1) & \textbf{otherwise} \end{cases}$$

Merge Sort

- If $N > 2$
  - Split — $O(1)$
  - $2\times$ `mergeSort`$(\frac{N}{2})$ — ???
  - `merge`(N) — $O(N)$
- Otherwise
  - Sort 2 elements — $O(1)$

$$T_{merge}(N) = \begin{cases} O(N) + 2 \cdot T_{merge}(\frac{N}{2}) & \textbf{if } N > 2 \\ O(1) & \textbf{otherwise} \end{cases}$$

Merge Sort

- If $N > 2$
  - Split — $O(1)$
  - $2\times$ `mergeSort(`$\frac{N}{2}$`)` — ???
  - `merge(`N`)` — $O(N)$
- Otherwise
  - Sort 2 elements — $O(1)$

$$T_{merge}(N) = \begin{cases} O(N) + 2 \cdot T_{merge}(\frac{N}{2}) & \textbf{if } N > 2 \\ O(1) & \textbf{otherwise} \end{cases}$$

How do we solve for $T_{merge}(N)$?

Recursive Complexity

Let's start with a simpler problem.

Factorial

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

Factorial

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

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

Factorial

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

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

Factorial

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

Factorial

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

Factorial

- $1! = 1$ Base Case
- $N! = N \cdot (N-1)!$ Recursive Case

Factorial

- $1! = 1$ — Base Case
- $N! = N \cdot (N-1)!$ — Recursive Case

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

Factorial

What is the complexity of Factorial?

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

Factorial

What is the complexity of Factorial?

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

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

Factorial

What is the complexity of Factorial?

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

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

Factorial

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

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

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

Generate a Hypothesis

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

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

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

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

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

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

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

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

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

Factorial

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

Factorial

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

Factorial

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

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

Factorial

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

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

Factorial

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

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

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

Factorial

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

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

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

$$c_1 \overset{?}{\leq} 1 \cdot c$$

Factorial

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

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

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

$$c_1 \overset{?}{\leq} 1 \cdot c \checkmark$$

Factorial

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

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

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

Factorial

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

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

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

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

Factorial

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

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

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

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

$$c_2 + c_1 \stackrel{?}{\leq} 2 \cdot c$$

Factorial

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

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

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

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

$$c_2 + c_1 \stackrel{?}{\leq} 2 \cdot c \checkmark$$

Factorial

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

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

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

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

Factorial

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

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

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

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

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

Factorial

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

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

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

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

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

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

Factorial

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

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

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

Factorial

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

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

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

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

Factorial

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

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

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

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

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

Factorial

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

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

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

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

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

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

Factorial

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

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

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

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

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

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

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. $T(N) \in O(N)$
2. Prove the hypothesis for the base case. $\exists c : T(N) \leq c \cdot N$
3. Prove the hypothesis inductively.

Induction

1. Generate a hypothesis. $T(N) \in O(N)$
2. Prove the hypothesis for the base case. $\exists c : T(N) \leq c \cdot N$
3. Prove the hypothesis inductively. $\ldots$

Induction

1. Generate a hypothesis. $T(N) \in O(N)$
2. Prove the hypothesis for the base case. $\exists c : T(N) \leq c \cdot N$
3. Prove the hypothesis inductively.
   - Prove that it holds for some specific case N
   - You can assume that you've already proved it for $N - 1$

Factorial

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

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

Factorial

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

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

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

Factorial

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

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

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

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

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

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

Factorial

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

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

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

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

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

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

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

Factorial

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

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

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

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

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

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

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

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. $\ldots$

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?

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

Answer: $O(N)$

Factorial

What is the complexity of Factorial?

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

Answer: $O(N)$[1]

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

Factorial

What is the complexity of Factorial?

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

Answer: $O(N)$[1]

How much memory does it use?

[1]Technically 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**.

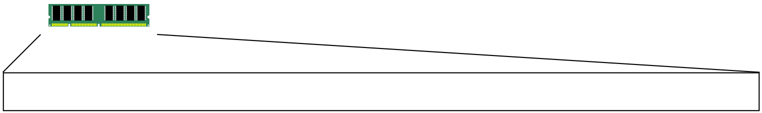
Stack Frames

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

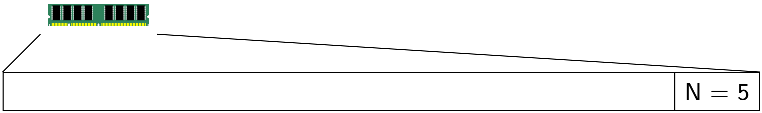
This chunk of memory is called a **Stack Frame**.

This is where the term `StackOverflowError` comes from.

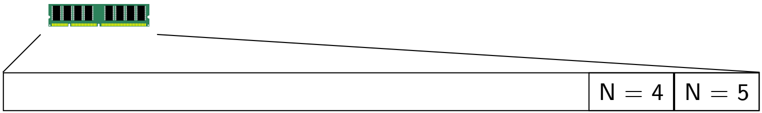
Stack Frames



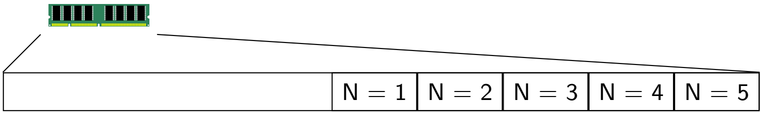
Stack Frames



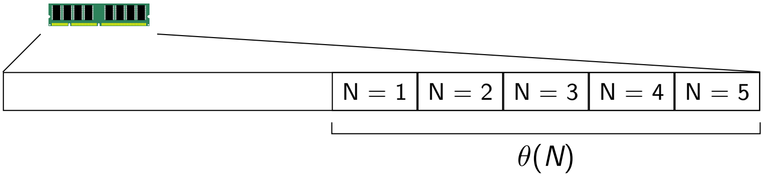
Stack Frames



Stack Frames



Stack Frames



Factorial (as a loop)

```
public long factorial(long N)
{
  long total = 1;
  for(long i = N; i > 0; i--)
  {
    total *= i
  }
  return total
}
```

Factorial

Why does this work?

Factorial (as a loop)

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

Factorial (as a loop)

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

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

Factorial (as a loop)

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

Factorial (as a loop)

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

Factorial (as a loop)

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

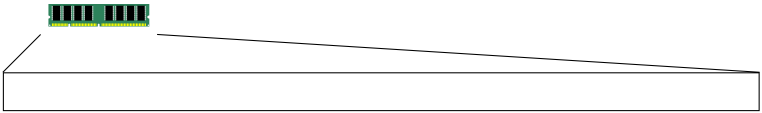
Factorial (as a loop)

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

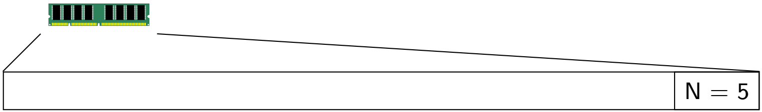
Factorial (as a loop)

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

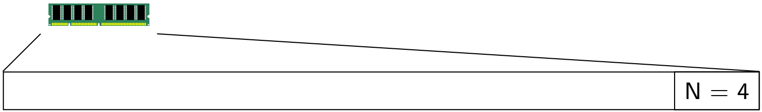
Stack Frames



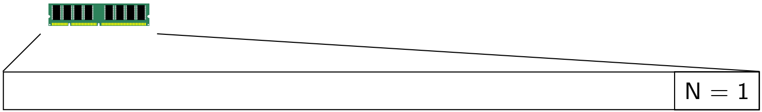
Stack Frames



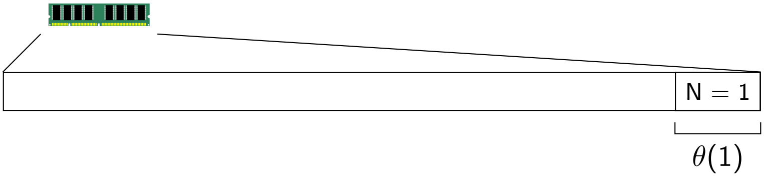
Stack Frames



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 away[2].

[2]... but not Java

Tail Recursion

If the recursive call is the last operation before the return, most languages optimize the recursion away[2].

This is called **Tail Recursion**

[2]... but not Java

Fibonacci

Time permitting...

Fibonacci

What's the complexity:

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