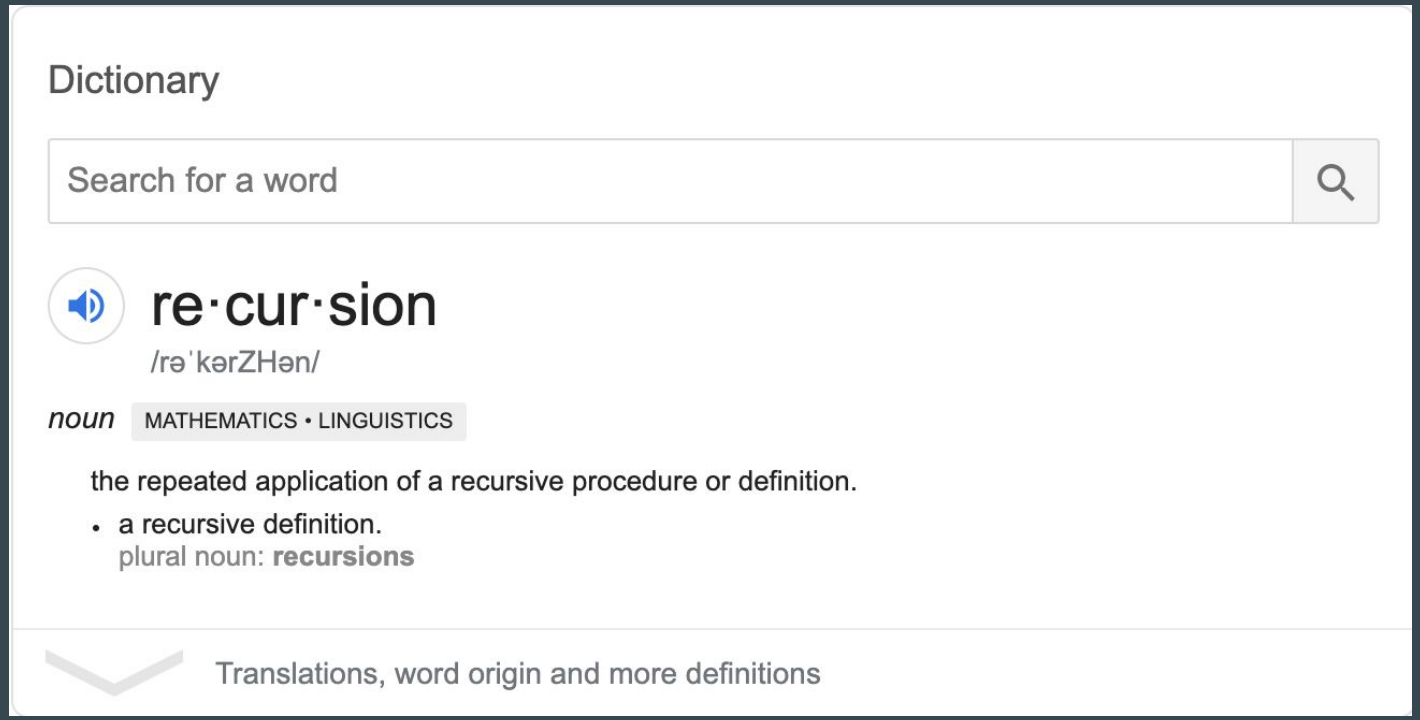
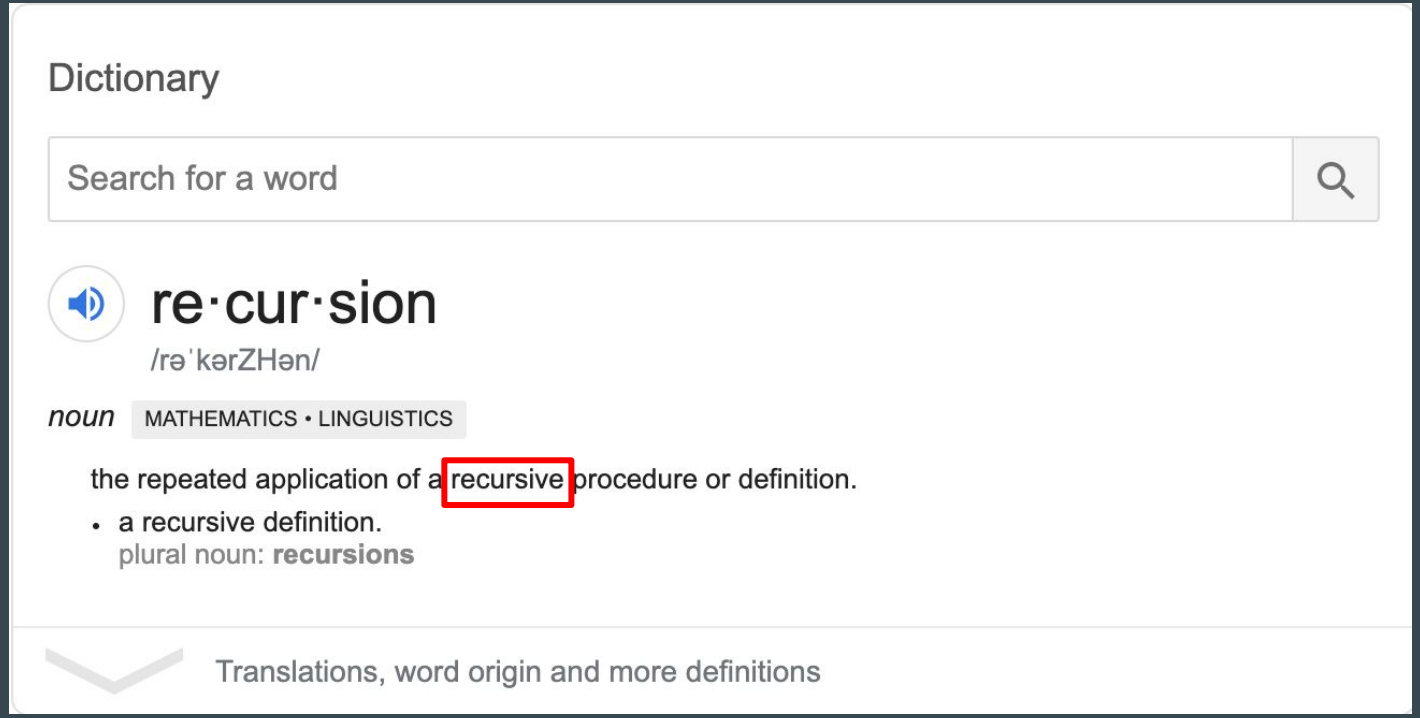
Recursion

No Capital Corp | Poly Team

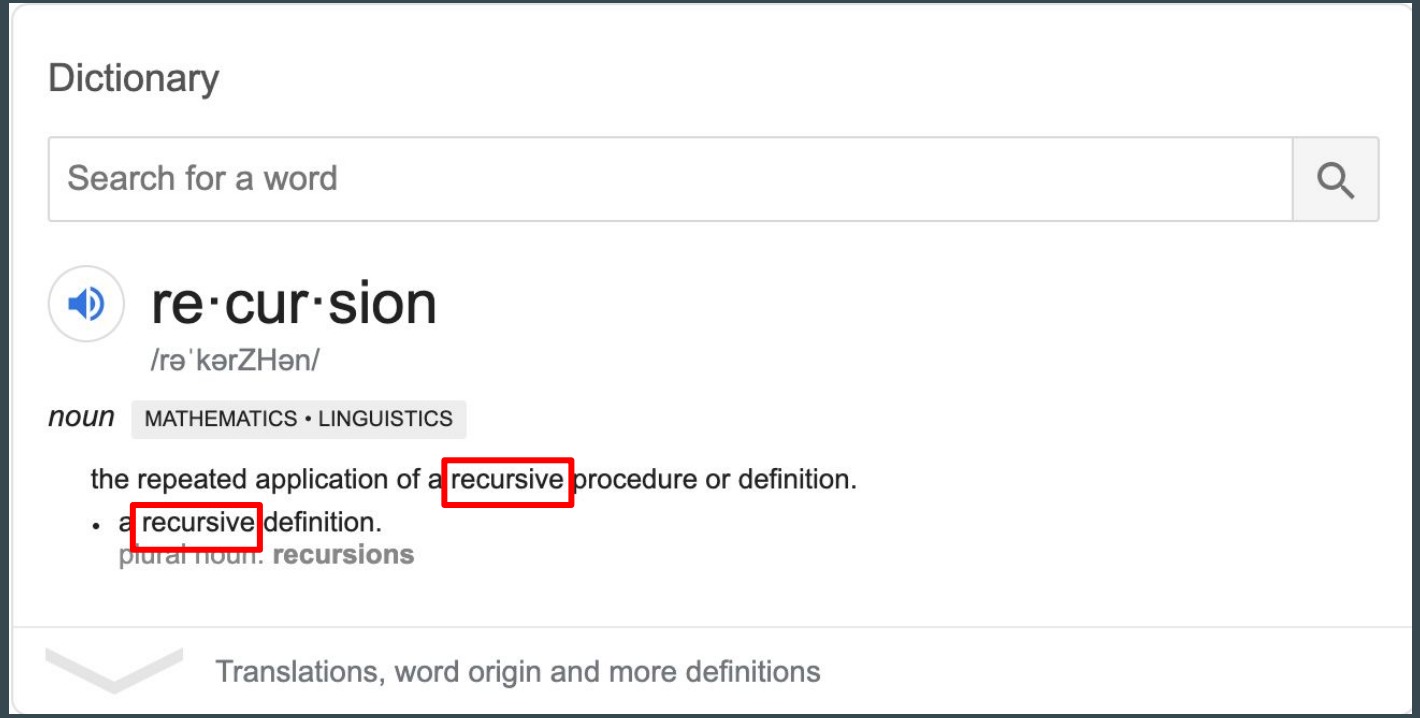
Recursion


Dictionary
Search for a word
re·cur·sion
/rəˈkərZHən/
noun
MATHEMATICS • LINGUISTICS
the repeated application of a recursive procedure or definition.
a recursive definition.
plural noun: recursions
Translations, word origin and more definitions

Recursion


Dictionary
Search for a word
re·cur·sion
/rəˈkərZHən/
noun
MATHEMATICS • LINGUISTICS
the repeated application of a recursive procedure or definition.
a recursive definition.
plural noun: recursions
Translations, word origin and more definitions

Recursion


Dictionary
Search for a word
re·cur·sion
/rəˈkərZHən/
noun
MATHEMATICS • LINGUISTICS
the repeated application of a recursive procedure or definition.
a recursive definition.
plural noun: recursions
Translations, word origin and more definitions

Recursion Example: Fibonacci

1 1

> First two elements are 1

Recursion Example: Fibonacci

1 1 2

> First two elements are 1

> The n-th element is the sum of (n-1)th element and (n-2)th element

Recursion Example: Fibonacci

1 1 2 = 1 + 1

> First two elements are 1

> The n-th element is the sum of (n-1)th element and (n-2)th element

Recursion Example: Fibonacci

1 1 2 3 = 2 + 1

> First two elements are 1

> The n-th element is the sum of (n-1)th element and (n-2)th element

Recursion Example: Fibonacci

1 1 2 3 5 = 3 + 2

> First two elements are 1

> The n-th element is the sum of (n-1)th element and (n-2)th element

Recursion Example: Fibonacci

1 1 2 3 5 … …

> First two elements are 1

> The n-th element is the sum of (n-1)th element and (n-2)th element

Recursion Example: Fibonacci

- Write a function to compute the n-th element in the Fibonacci sequence.

Recursion Example: Fibonacci

- Write a function to compute the n-th element in the Fibonacci sequence.
- **Consider a function that calls itself.**

```
function fib(n) {




}
```

```
function fib(n) {
  if (n == 1 || n == 2) {
    return ???;
  }


}
```

```
function fib(n) {
  if (n == 1 || n == 2) {
    return 1;
  }


}
```

```
function fib(n) {
  if (n == 1 || n == 2) {
    return 1;
  } else {
    return ???;
  }
}
```

```
function fib(n) {
  if (n == 1 || n == 2) {
    return 1;
  } else {
    return fib(n-1) + ???;
  }
}
```

```
function fib(n) {
  if (n == 1 || n == 2) {
    return 1;
  } else {
    return fib(n-1) + fib(n-2);
  }
}
```

```
function fib(n) {
  if (n == 1 || n == 2) {
    return 1;
  } else {
    return fib(n-1) + fib(n-2);
  }
}
```

```
function fib(n) {
  if (n == 1 || n == 2) {
    return 1;
  } else {
    return fib(n-1) + fib(n-2);
  }
}
```

Base Case

```
function fib(n) {
  if (n == 1 || n == 2) {
    return 1;
  } else {
    return fib(n-1) + fib(n-2);
  }
}
```

Base Case

Recursive Step

Recursion Example: Fibonacci

- What is the time complexity of our `fib` function?

Recursion Example: Fibonacci

- What is the time complexity of our `fib` function?
- **O(2^n)**

Recursion Example: Fibonacci

- What is the time complexity of our `fib` function?
- **O(2^n)**
- **Why?**

Runtime of Fibonacci

- Let T(n) be the runtime of `fib` with input `n`

Runtime of Fibonacci

- Let T(n) be the runtime of `fib` with input `n`
- **Consider `fib(1)` and `fib(2)`**

```
function fib(n) {
  if (n == 1 || n == 2) {
    return 1;
  } else {
    return fib(n-1) + fib(n-2);
  }
}
```

n = 1

```
function fib(n) {
  if (n == 1 || n == 2) {
    return 1;
  } else {
    return fib(n-1) + fib(n-2);
  }
}
```

n = 1

O(1)

```
function fib(n) {
  if (n == 1 || n == 2) {
    return 1;
  } else {
    return fib(n-1) + fib(n-2);
  }
}
```

n = 2

O(1)

Runtime of Fibonacci

- Let T(n) be the runtime of `fib` with input `n`
- Consider `fib(1)` and `fib(2)` **- O(1)**

Runtime of Fibonacci

- Let T(n) be the runtime of `fib` with input `n`
- Consider `fib(1)` and `fib(2)` - O(1)
- **For simplicity, let T(1) = T(2) = 1**

Runtime of Fibonacci

- Let T(n) be the runtime of `fib` with input `n`
- Consider `fib(1)` and `fib(2)` - O(1)
- For simplicity, let T(1) = T(2) = 1
- **T(n) = ??? (for n > 2)**

```
function fib(n) {
  if (n == 1 || n == 2) {
    return 1;
  } else {
    return fib(n-1) + fib(n-2);
  }
}
```

n > 2

```
function fib(n) {
  if (n == 1 || n == 2) {
    return 1;
  } else {
    return fib(n-1) + fib(n-2);
  }
}
```

n > 2

T(n-1)

```
function fib(n) {
  if (n == 1 || n == 2) {
    return 1;
  } else {
    return fib(n-1) + fib(n-2);
  }
}
```

n > 2

T(n-1)

T(n-2)

```
function fib(n) {
  if (n == 1 || n == 2) {
    return 1;
  } else {
    return fib(n-1) + fib(n-2);
  }
}
```

n > 2

$$T(n) = T(n-1) + T(n-2) + 1$$

fib(n-1): $T(n-1)$

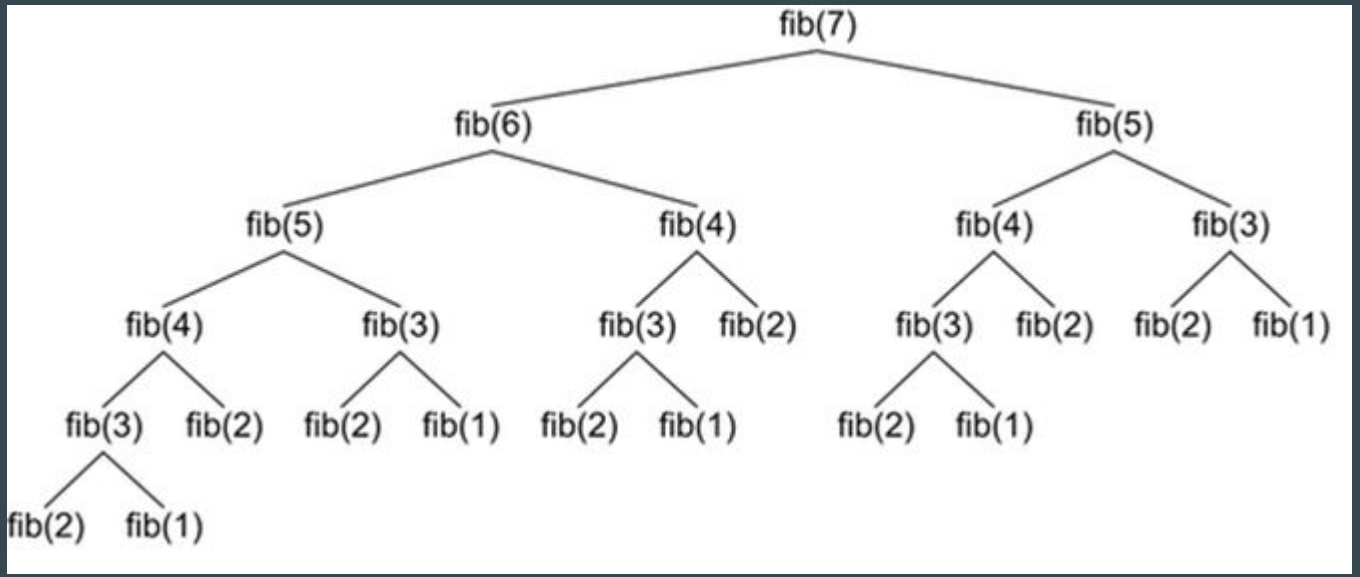
fib(n-2): $T(n-2)$

Runtime of Fibonacci

- Let T(n) be the runtime of `fib` with input `n`
- Consider `fib(1)` and `fib(2)` - O(1)
- For simplicity, let T(1) = T(2) = 1
- **T(n) = T(n-1) + T(n-2) + 1**

https://www.rubyguides.com/2015/08/ruby-recursion-and-memoization/

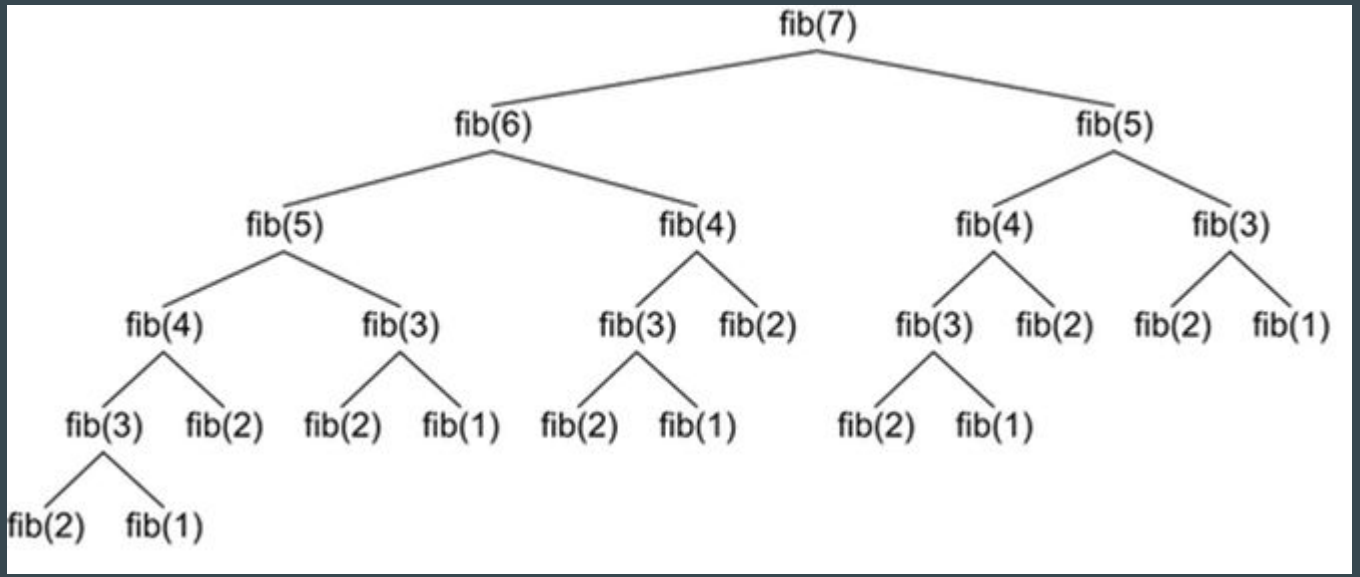
Runtime of Fibonacci



- $T(n) = T(n-1) + T(n-2) + 1$

https://www.rubyguides.com/2015/08/ruby-recursion-and-memoization/

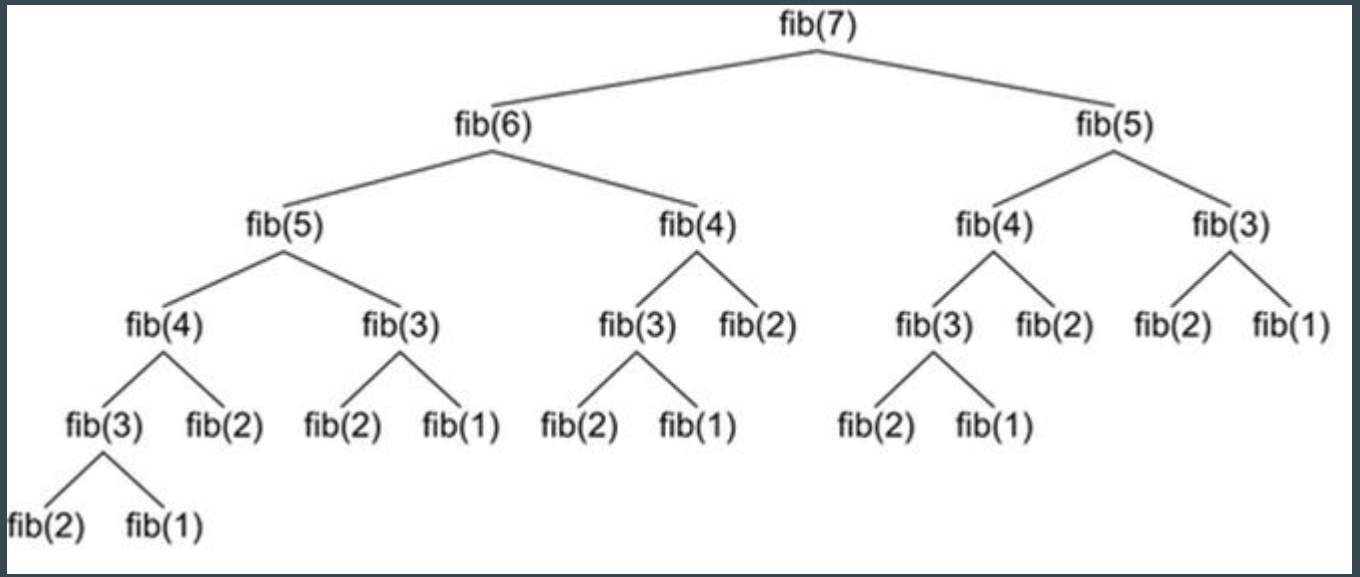
Runtime of Fibonacci



- T(n) is at most # of nodes in recursive tree

https://www.rubyguides.com/2015/08/ruby-recursion-and-memoization/

Runtime of Fibonacci



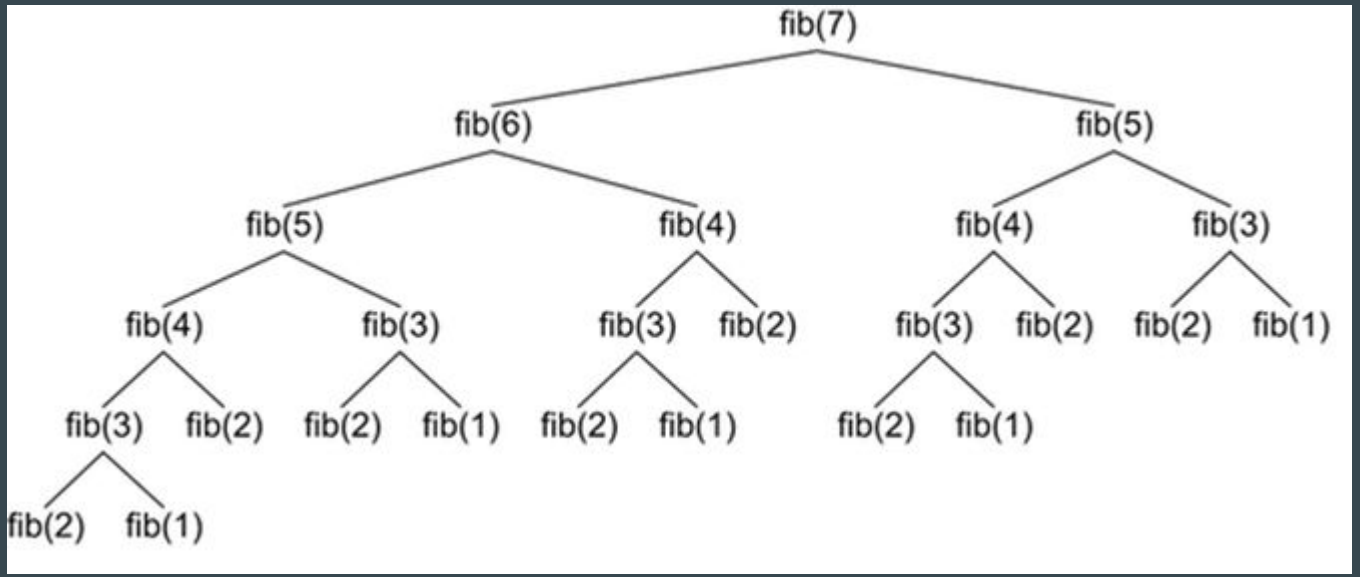
- T(n) is O(2^n)

Memoization

No Capital Corp | Poly Team

https://www.rubyguides.com/2015/08/ruby-recursion-and-memoization/

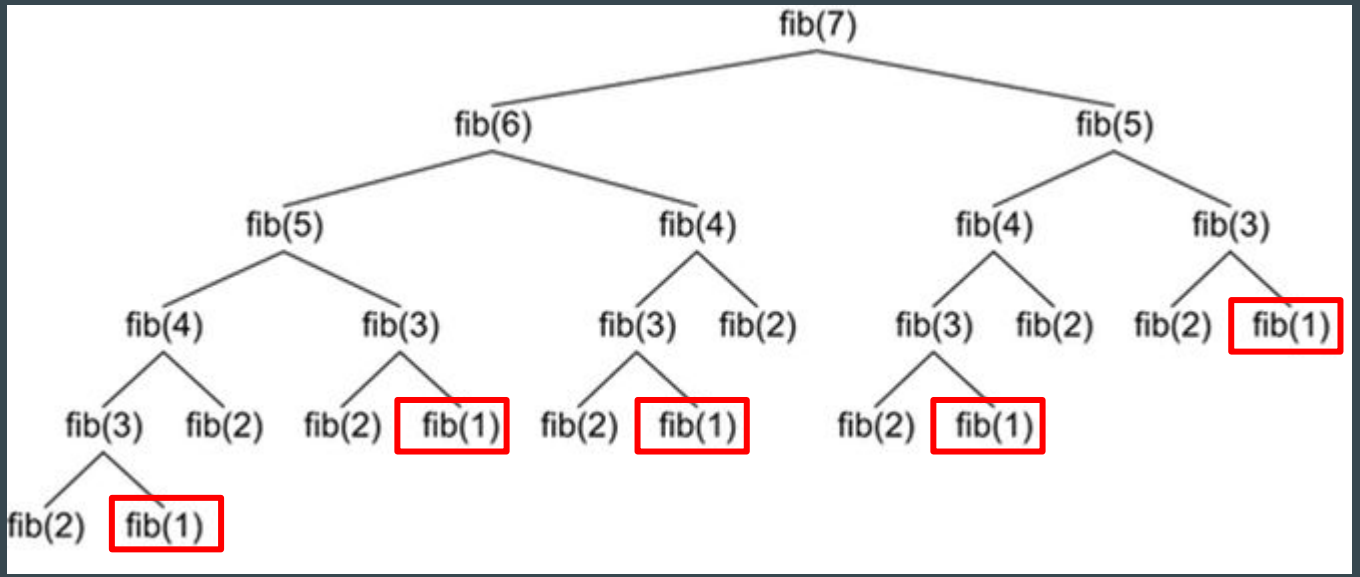
Runtime of Fibonacci



- T(n) is O(2^n)

https://www.rubyguides.com/2015/08/ruby-recursion-and-memoization/

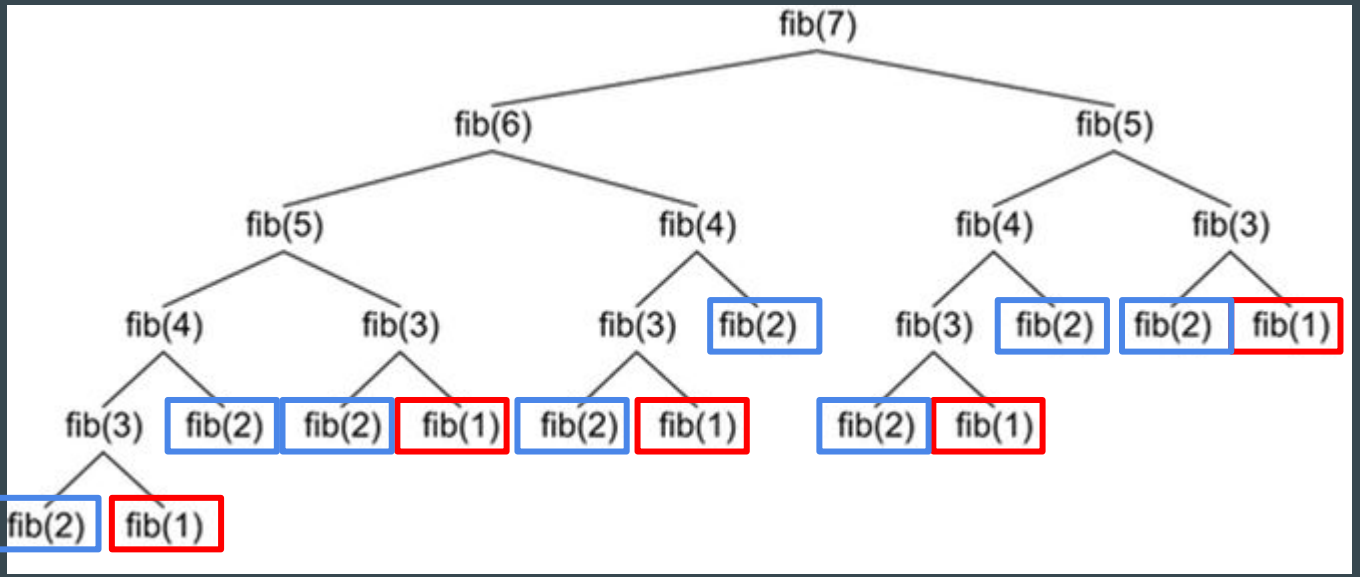
Runtime of Fibonacci



- T(n) is O(2^n)

https://www.rubyguides.com/2015/08/ruby-recursion-and-memoization/

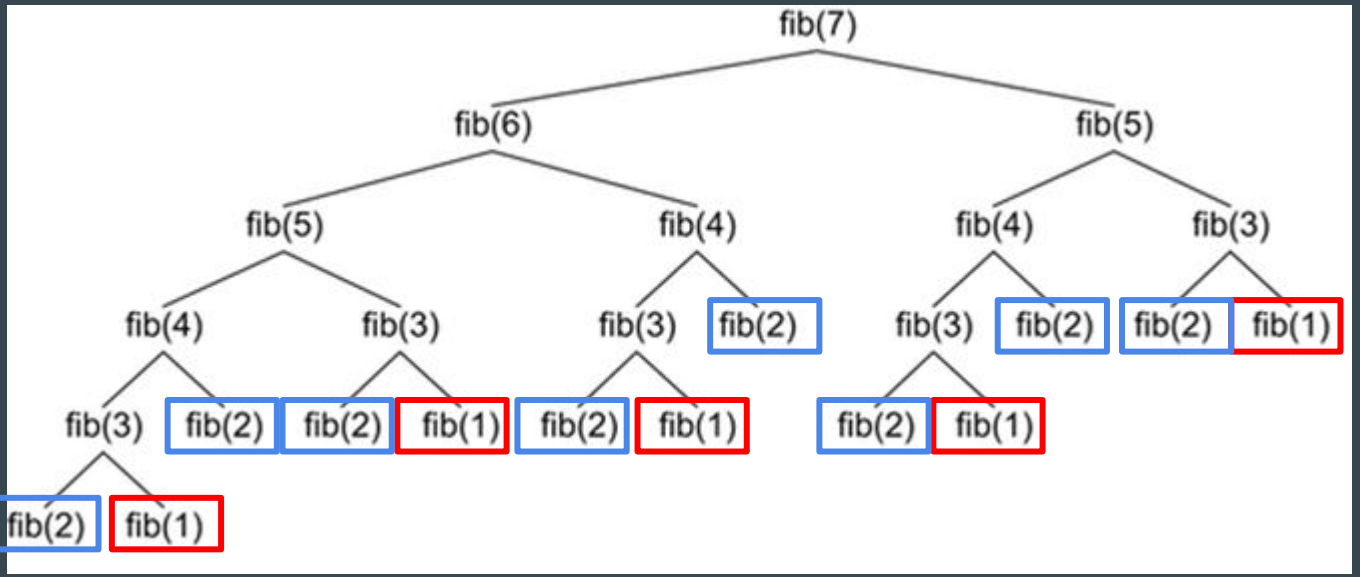
Runtime of Fibonacci



- T(n) is O(2^n)

https://www.rubyguides.com/2015/08/ruby-recursion-and-memoization/

Runtime of Fibonacci

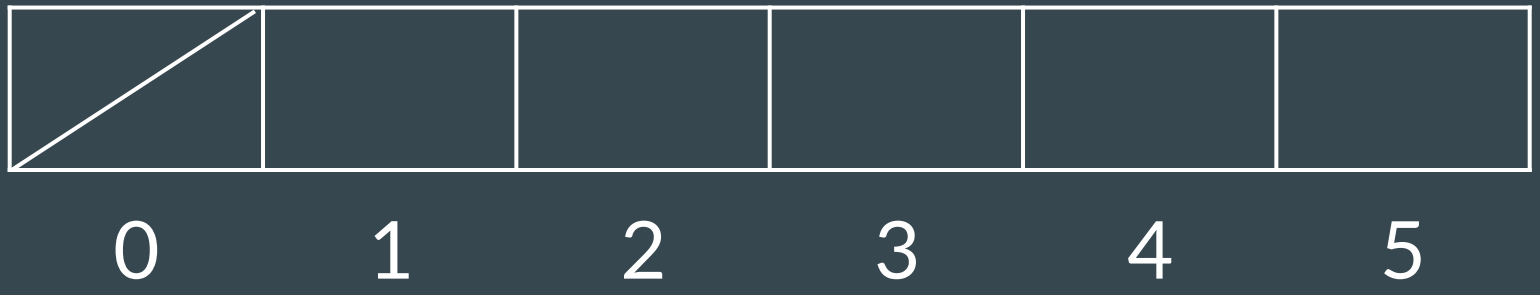


- Lots of repetitions!

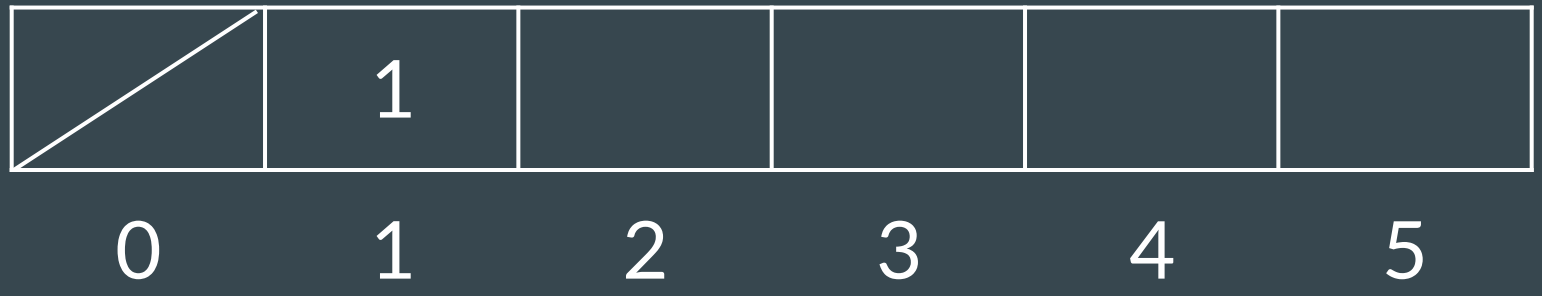
What We Will Do...

Trading space for runtime!

Array

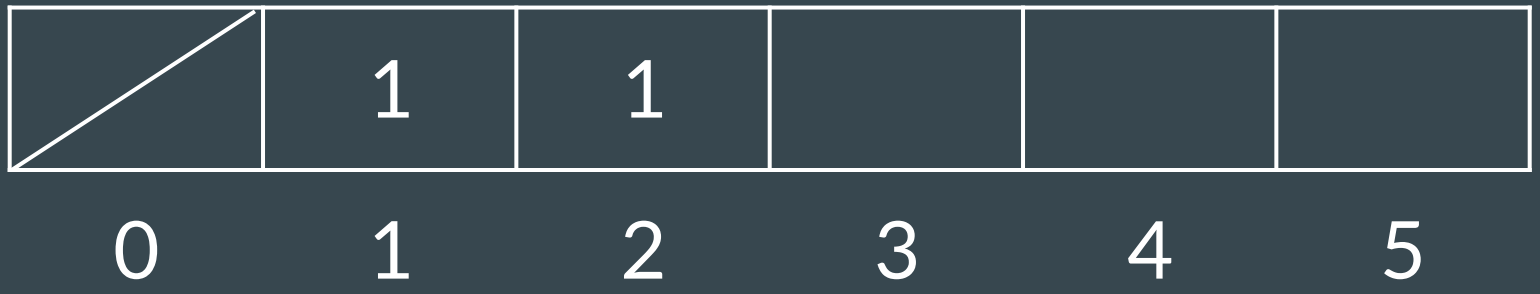


Array



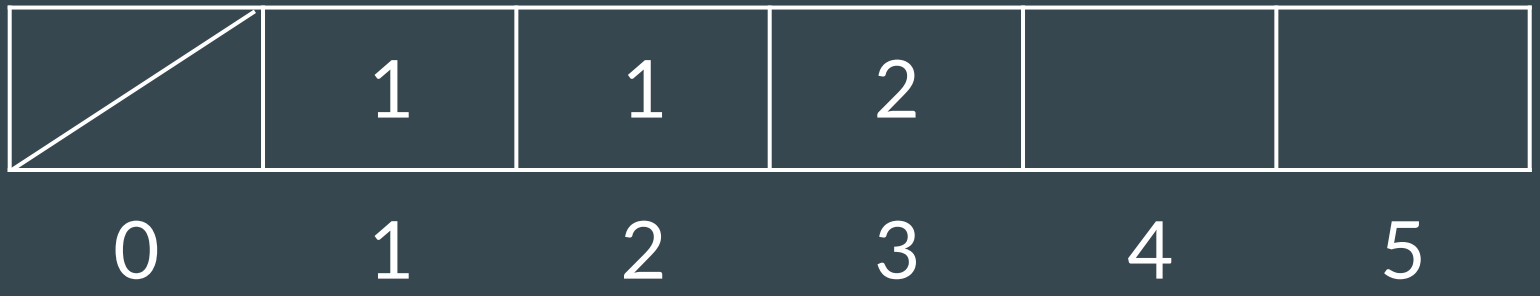
After calculating fib(1)

Array



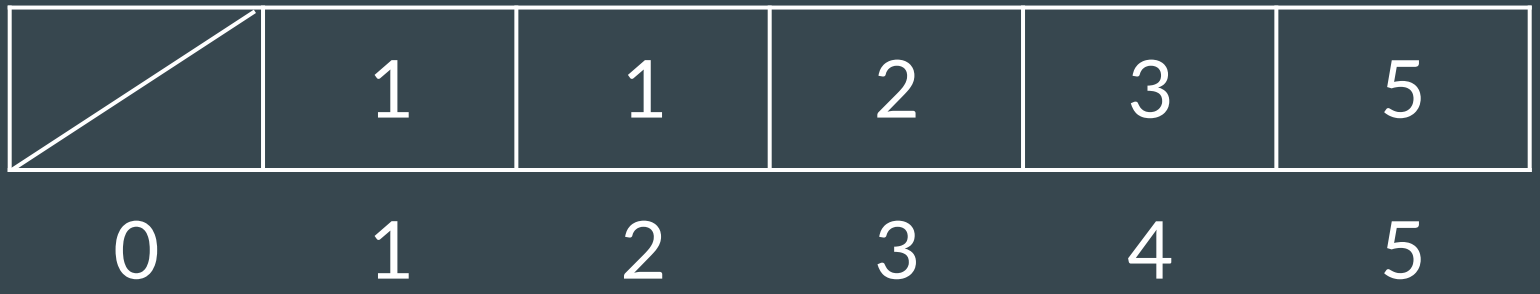
After calculating fib(2)

Array



After calculating fib(3)

Array



After calculating fib(n), store the value at arr[n]

```
function fib(n) {
  if (n == 1 || n == 2) {
    return 1;
  } else {
    return fib(n-1) + fib(n-2);
  }
}
```

```
let fibArr = [1, 1, 1];

function fib(n) {
  if (n == 1 || n == 2) {
    return 1;
  } else {
    return fib(n-1) + fib(n-2);
  }
}
```

```
let fibArr = [1, 1, 1];
```

Placeholder

```
let fibArr = [1, 1, 1];

function fib(n) {
  if (fibArr[n]) {
    return fibArr[n];
  } else {
    return fib(n-1) + fib(n-2);
  }
}
```

```
let fibArr = [1, 1, 1];

function fib(n) {
  if (fibArr[n]) {
    return fibArr[n];
  } else {
    let result = fib(n-1) + fib(n-2);
    fibArr[n] = result;
    return result;
  }
}
```

```
let fibArr = [1, 1, 1];

function fib(n) {
  if (fibArr[n]) {
    return fibArr[n];
  } else {
    let result = fib(n-1) + fib(n-2);
    fibArr[n] = result;
    return result;
  }
}
```

Runtime
O(n)

```
let fibArr = [1, 1, 1];

function fib(n) {
  if (fibArr[n]) {
    return fibArr[n];
  } else {
    let result = fib(n-1) + fib(n-2);
    fibArr[n] = result;
    return result;
  }
}
```

Runtime
O(n)
HW: Why?