

Runtime Complexity

Describes the performance
of an algorithm

How much more processing power/time is required
to run your algorithm if we double the inputs?

String Reverse

abc → cba

abcdefghijklmnopqrstuvwxyz → zyxwvutsrqponmlkjihgfedcba

Each additional
character = 1 step
through 1 loop

This would be 'N', or
'linear' runtime.

Steps Algorithm

steps = 2

#	-
#	#

Had to do
4 things

steps = 3

#	-	-
#	#	-
#	#	#

Had to do
9 things

steps = 4

#	-	-	-
#	#	-	-
#	#	#	-
#	#	#	#

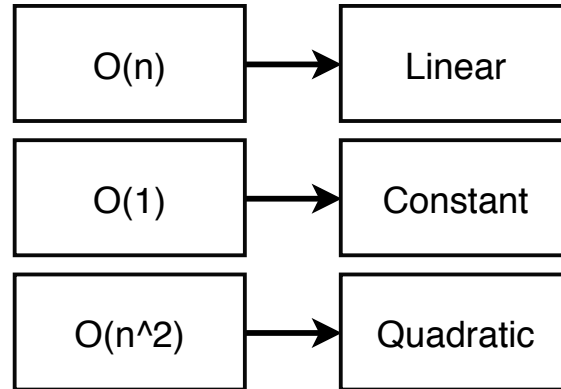
Had to do
16 things

As 'n' increased by one,
we had to do way, way
more stuff, or $(n*n)$ things
total

This would be N^2 , or
quadratic runtime

<i>Constant Time</i>	1	→	No matter how many elements we're working with, the algorithm/operation/whatever will always take the same amount of time
<i>Logarithmic Time</i>	$\log(n)$	→	You have this if doubling the number of elements you are iterating over doesn't double the amount of work. Always assume that searching operations are $\log(n)$
<i>Linear Time</i>	n	→	Iterating through all elements in a collection of data. If you see a for loop spanning from '0' to 'array.length', you probably have 'n', or linear runtime
<i>Quasilinear Time</i>	$n * \log(n)$	→	You have this if doubling the number of elements you are iterating over doesn't double the amount of work. Always assume that any sorting operation is $n * \log(n)$
<i>Quadratic Time</i>	n^2	→	Every element in a collection has to be compared to every other element. 'The handshake problem'
<i>Exponential Time</i>	2^n	→	If you add a <i>single</i> element to a collection, the processing power required doubles

Big 'O' Notation



Identifying Runtime Complexity

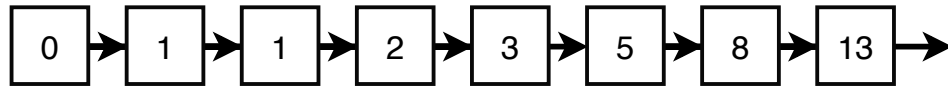
Iterating with a simple for loop through a single collection?	→	Probably $O(n)$
Iterating through half a collection?	→	Still $O(n)$. There are no constants in runtime.
Iterating through two *different* collections with separate for loops?	→	$O(n + m)$
Two nested for loops iterating over the same collection?	→	$O(n^2)$
Two nested for loops iterating over different collections?	→	$O(n*m)$
Sorting?	→	$O(n*\log(n))$
Searching a sorted array?	→	$O(\log(n))$

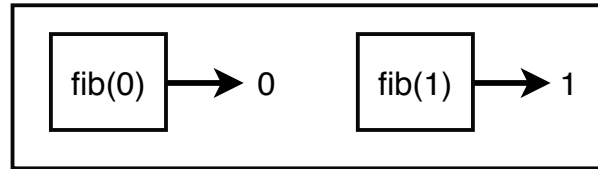
Last note...

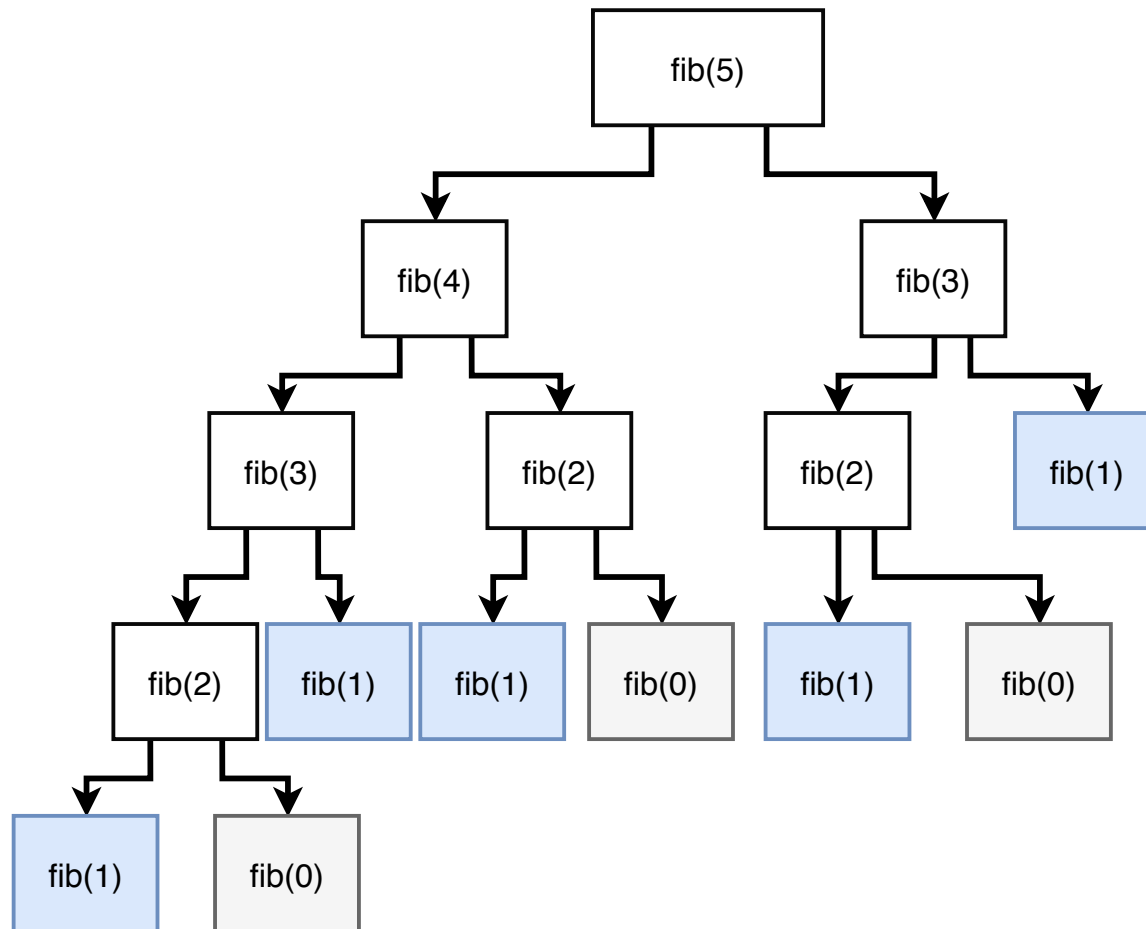
Space Complexity is a thing too

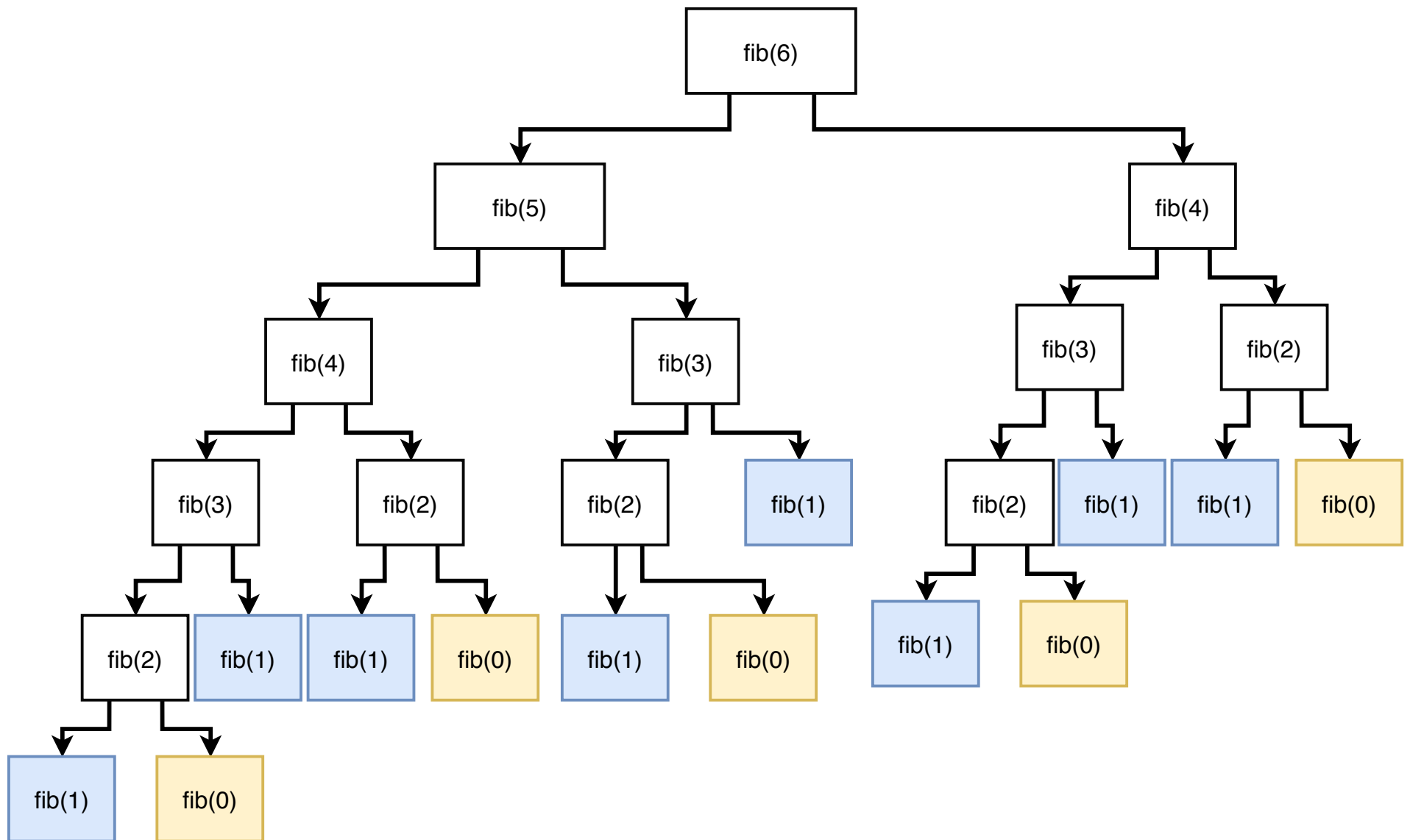
How much more memory is
required by doubling the
problem set?

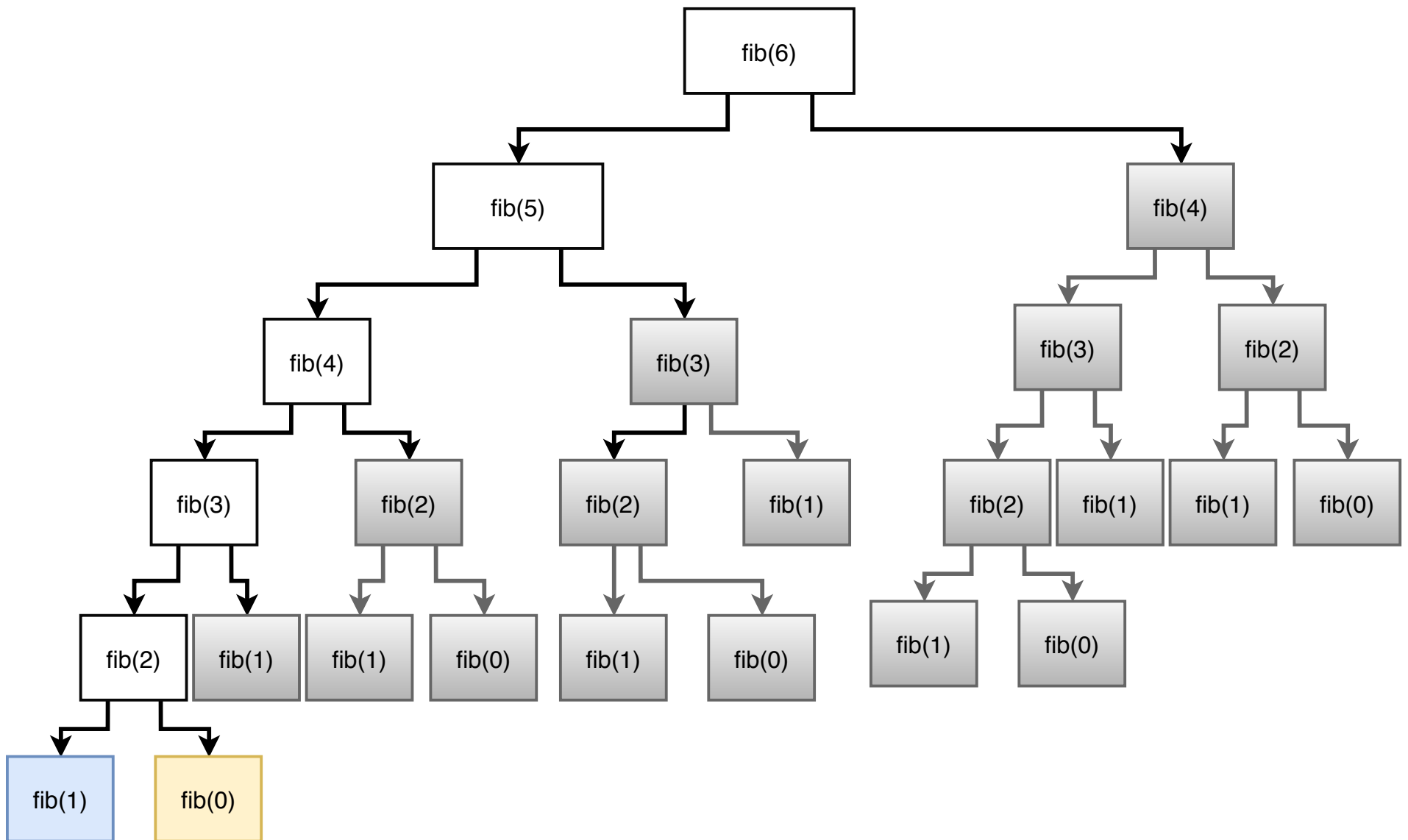
Fibonacci Series











memoization

Store the arguments of each function call along with the result. If the function is called again with the *same arguments*, return the precomputed result, rather than running the function again

