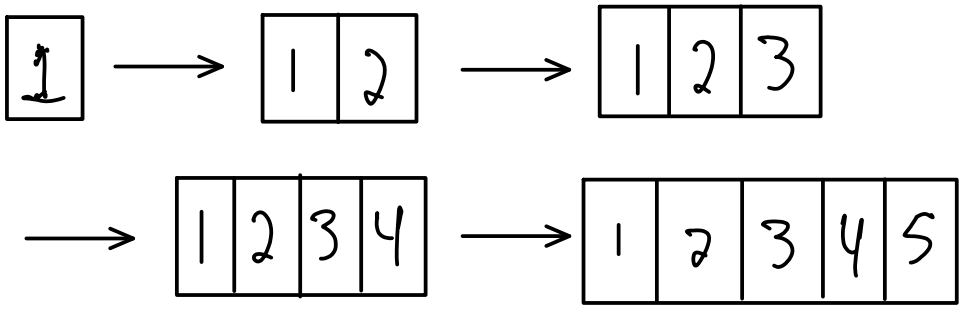
Idea: When out of space, copy to an array 1 larger.

Start with the empty array.



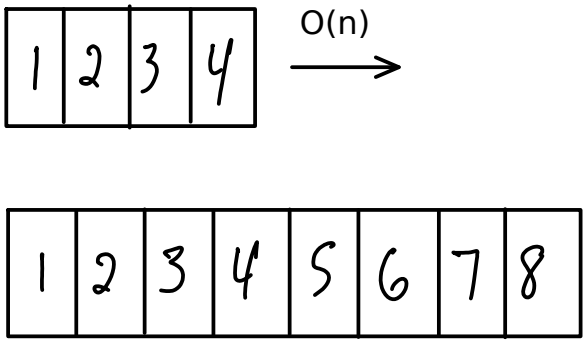
ins	ops
1	1
2	2
3	3
4	4
5	5

Question: What's the runtime for n insertions?

$O(n^2)$

Idea: When full, double.

For the sake of the example, start with an empty array of size 4.



ins	ops	O(?)
1	1	$O(1)$
2	1	↓
3	1	
4	1	
5	5	$O(n)$
6	1	$O(1)$
7	1	↓
8	1	
9	9	$O(n)$

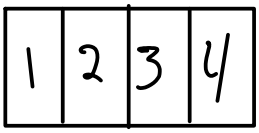
Amortized analysis: accounting analysis--how many tokens?

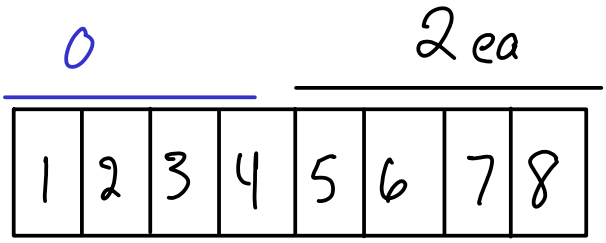
Clearly, 1 won't work. How about 2?
Again, start with an initially empty array of size 4.

	budget	spend	left over (cumulative)
1	2	1	1
2		1	2
3		1	3
4		1	4
5		5	1
6		1	2
7		1	3
8		1	4
9		9	6 - 9 = bad.

Ok, how about 3?

	budget	spend	left over (cumulative)
1	3	1	2
2		1	4
3		1	6
4		1	8 ←
5		5	11 - 5 = 6
6		1	8
7		1	10
8		1	12
9		9	15 - 9 = 6

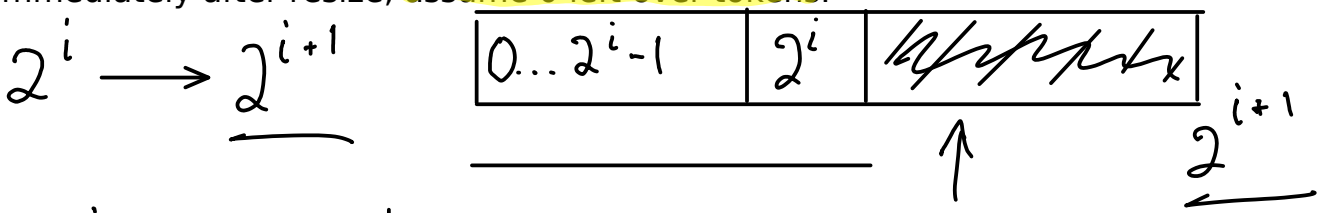




3 Tokens Pay For:

1. Initial insertion
2. Move self
3. Move someone in 1st half.

Ok, but this isn't really interesting yet. How about the general case?
Immediately after resize, assume 0 left over tokens.



$(2^{i+1} - 1) - 2^i$ ins

$2^i - 1$ ins → full

$2(2^i - 1)$ extra tokens

$2(2^i - 1) + 3 = 2^{i+1} + 1$

3 tokens → ins $\in O(1)$

copy 2^{i+1}

insert new

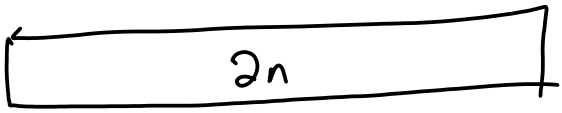
$2^{i+1} + 1$

n ins $O(n)$

When to shrink? How about we halve the array when half full?

ins

$O(n)$

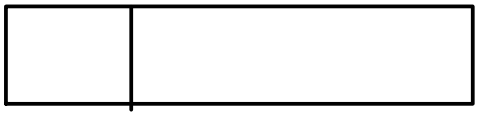


$n+1$

$O(n^2)$

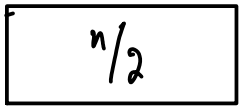
n

Shrink @ 1/4 full



n

del 1/4



$n/4$