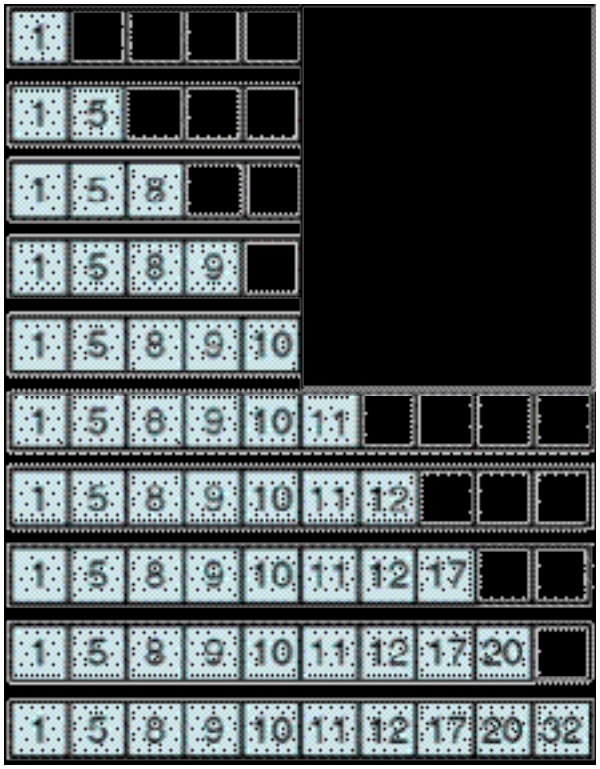
Worksheet 15: Amortized Constant Execution Time

In preparation: Read Chapter 5 to learn more about the basic abstract data types, and the introduction to the dynamic array. If you have not done so already you should complete worksheet 14, which introduces the dynamic array.

In the previous worksheet you analyzed the algorithmic execution time for the **add** operation in a dynamic array. When the size was less than the capacity, the execution time was constant. But when a reallocation became necessary, execution time slowed to O(n).

This might at first seem like a very negative result, since it means that the worst case execution time for addition to a dynamic array is O(n). But the reality is not nearly so bleak. Look again at the picture that described the internal array as new elements were added to the collection.



Notice that the costly reallocation of a new array occurred only once during the time that ten elements were added to the collection. If we compute the *average* cost, rather than the *worst case* cost, we will see that the dynamic array is still a relatively efficient container.

To compute the average, count 1 "unit" of cost each time a value is added to the dynamic array without requiring a reallocation. When the reallocation occurs, count one "unit" of cost for each assignment performed as part of the reallocation process, plus one more for placing the new element into the newly enlarged array. How many "units" are spent in the entire process of inserting these ten elements? What is the average "unit" cost for an insertion?

Total units spent in the process is 15.
The average unit cost is 1.5

When we can bound an "average" cost of an operation in this fashion, but not bound the worst case execution time, we call it *amortized constant* execution time. Amortized constant execution time is often written as O(1)+, the plus sign indicating it is not a guaranteed execution time bound.

Do a similar analysis for 25 consecutive add operations, assuming that the internal array begins with 5 elements (that means, with size =0, capacity=5). What is the cost when averaged over this range?

N = 25

Array doubles when n surpasses 5, 10, and 20 for the first time. (so the 6^{th}, 11^{th} and 21^{st} values cause the array to be doubled)

Total cost = 25(# elements) + 5(1st resize) + 10(2nd resize) + 20(3rd resize) = 60

Average cost is 60/25 = 2.4