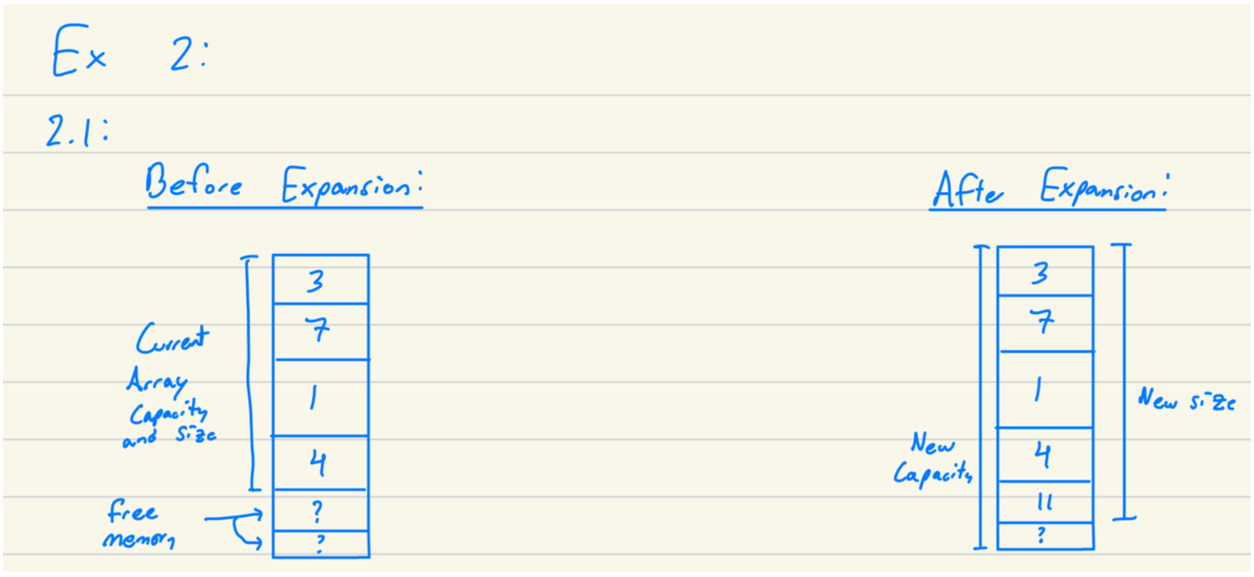
Ex 2.1:

The difference between array size and capacity is that the size of an array describes the current number of elements stored in the array while the capacity describes the maximum number of elements that could be stored in the allocated memory for this array.

Ex 2.2:

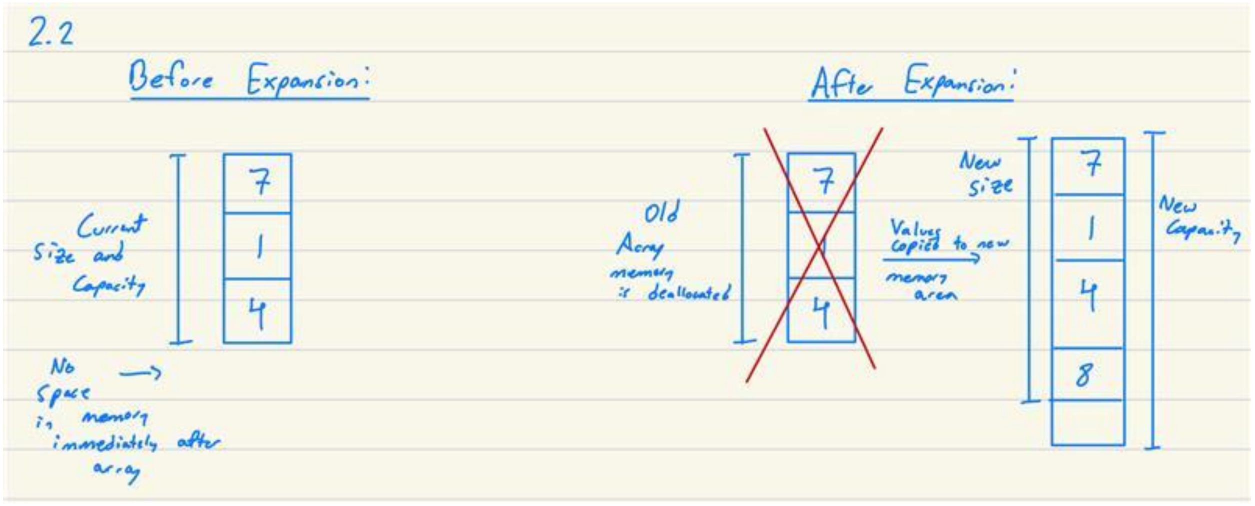
In the case where there is space in memory after the end of the array, the subsequent memory can be allocated to the existing array, and the capacity can be increased. Then, the new element is inserted, and the size will also be increased.

Diagram:



In the case where there is not enough space in memory after the end of the array, the program can find a space in memory that is large enough to accommodate the new capacity and it can allocate that memory to a new array. Then, the program would copy over all of the existing values from the previous array and increase the capacity. Once it has been copied over, it can also insert the new element and increase the size.

Diagram:


2.2
Before Expansion:
Current Size and Capacity
7
1
4
No space in memory immediately after array
After Expansion:
Old Array memory is deallocated
7
1
4
New size
Values copied to new memory area
7
1
4
8
New Capacity

Ex 2.3:

One technique that real world arrays use to amortize the cost is to use geometric resizing. Geometric resizing refers to expanding the capacity by a constant factor such as 1.125 times the original size in Python. This means that the number of copy operations per insertion (when you have to copy over the old values since you are creating a whole new array when you insert a new element) decreases over time. It decreases over time since in the beginning if you have an array of size 1 and you add an element, you have to expand it and copy everything over after 1 insertion. However, if you have 2000 elements and you say double the size, you would then need to insert 2000 more elements before having to incur a O(n) complexity penalty of copying the data.