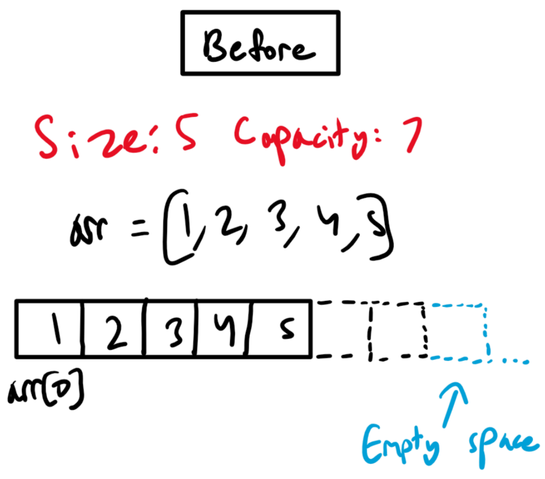
Exercise 2:

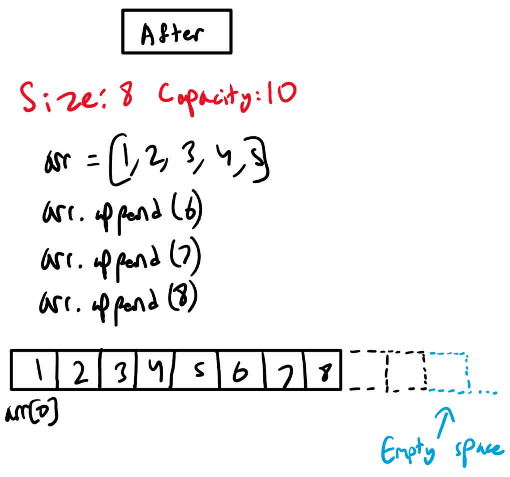
Question 1:

The difference between array size and capacity is that array size represents the actual number of elements that an array can hold. For example, when initializing an array in python with for example: arr[5]. The “arr” size would be 5 elements because that is the number of elements the array can actually hold but array capacity is the amount of space Python would allocate in memory for this array: “arr.” The array capacity is always bigger than array size and is usually calculated by multiplying array size by a constant that is bigger than one. The array capacity is needed so that when a person appends an element to an array, the computer does not need to allocate an entirely new spot in memory and have to copy all the old elements into the new space. It makes the process of appending elements easier and much faster.

Question 2:

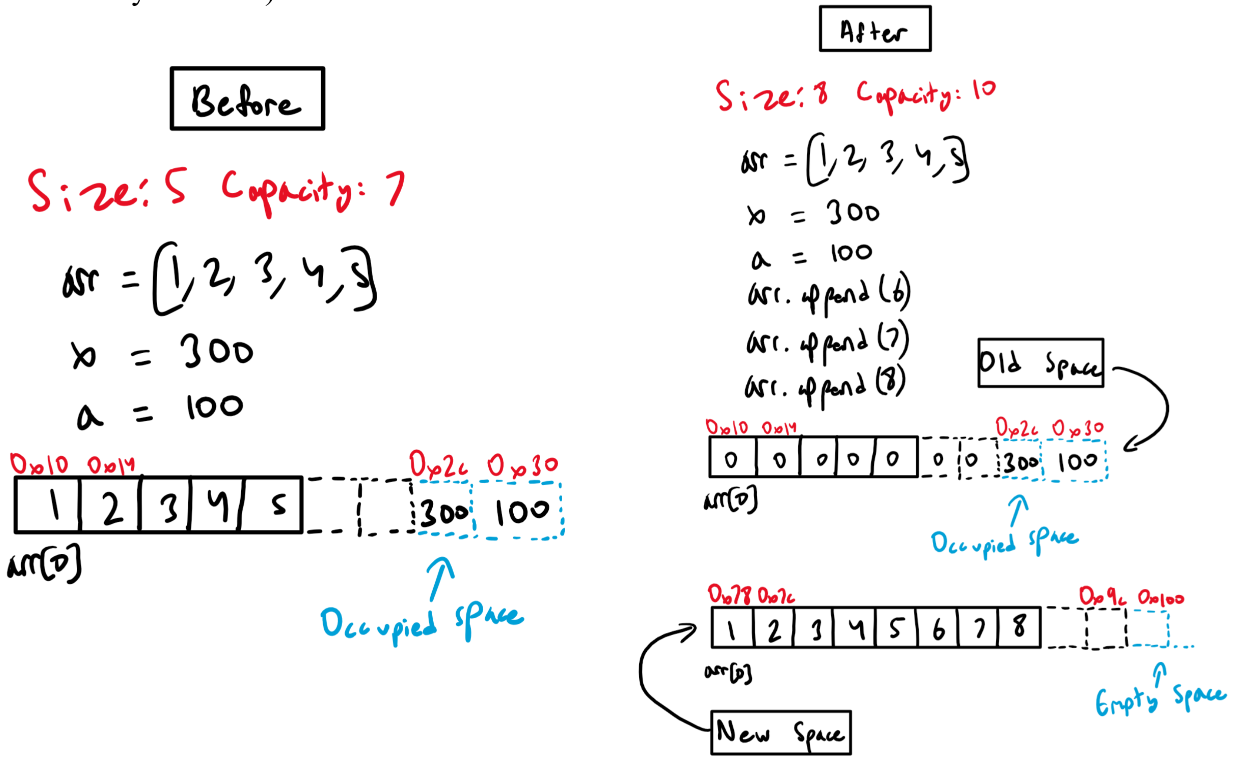
1) When an array needs to grow beyond its current capacity, it will first look at the memory right after the capacity of the array. If it is not occupied it will store the next element in that spot and resize the current array. These diagrams below represent this situation. As seen on the diagram on the left, there is extra space after capacity and so when three elements are added, the extra element is added right next to the array as seen in the “After” diagram. Furthermore, the array is resized to have the size of 8 elements and the capacity is also recalculated to have 10 elements. This represents the case where there is space in memory after the end of the array. (Note: the values for capacity are just an example, in reality, the value might be very different)





2) In the case where there is no memory after the end of the array, the interpreter needs to completely move the entire array to a new spot in memory. Then, it needs to copy all the elements in the array into this new space along with the new element. It then needs to resize the array and calculate its new capacity in the new memory space allocated for it. As seen in the “Before” diagram, the space after the end of the array is being used to store variables such as “x” and “a.” Thus, we will not be able to simply

resize the array as the space after is already occupied. In this case, Python will find and allocate an entirely new area in memory for this larger array as seen in the "After" diagram. It will find an appropriate space to move the entire array, after which Python's garbage collector will delete the contents of the old memory space as shown by how the address "0x10" and its neighbors have the value of "0" or NULL. Again, the capacity will be recalculated by multiplying the new size by a constant bigger than one. (Note: the values for capacity are just an example, in reality, the value might be very different)



Question 3:

One method is to resize your array incrementally, when the array is initialized it first starts with a small initial capacity. When the array needs to be resized, you make the interpreter only increase the space in memory by a fixed, small increment. This ensures that the user only uses as minimal space as needed. This is because if the memory space were to be double the size of the array, then a lot of useless memory might be unusable because of the array. This extra memory space also may be almost entirely filled with NULL values and so it would be useless for this array and other variables. By using incremental resizing, the cost of resizing is spread out over multiple operations. Although resizing might happen a lot, this technique will ensure that each time it happens, it is an efficient process that does not take that much time. This ultimately leads to a more predictable and efficient memory usage process.