Efficiency of Algorithms

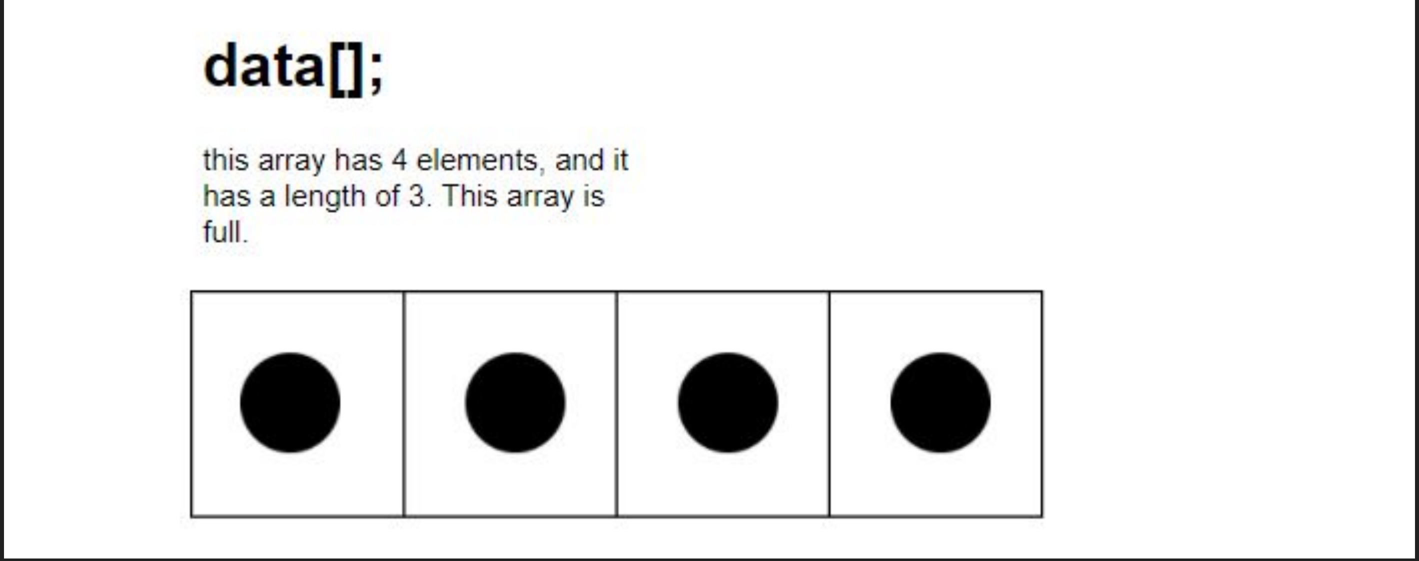
Chris King
Ryan Scott

Process of Resizing Arrays

When instantiating an array, the size stays static for the duration of the program. The array occupies an amount of memory in proportion to its size, and the dimension of the array can’t be changed later on. So how do we deal with these memory constraints when trying to add elements to an array that doesn’t have enough space?

Process of Resizing Arrays

Think of an array as a container. This container, of course, has a set maximum capacity. Once it’s filled, it can no longer store more objects.

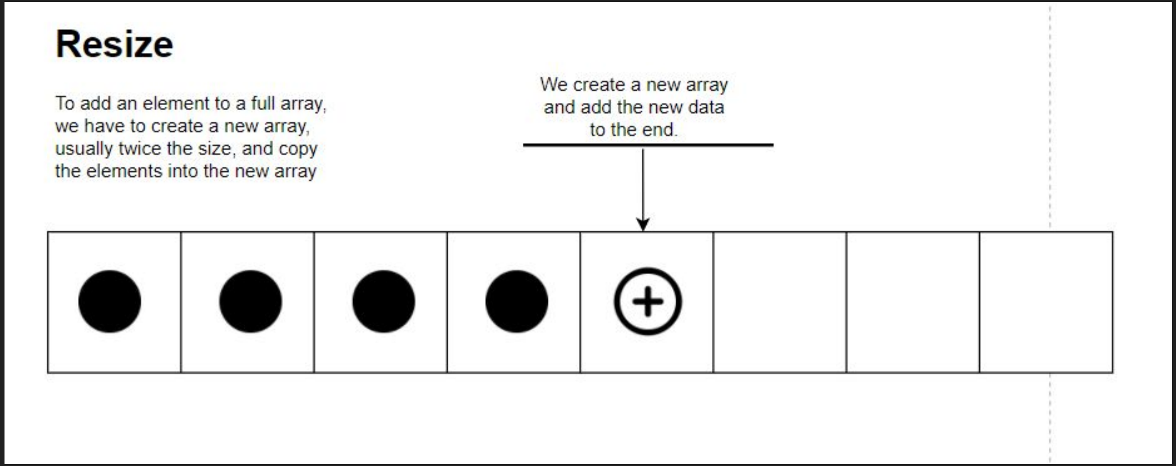


data[];

this array has 4 elements, and it has a length of 3. This array is full.

Process of Resizing Arrays

If we want to add an element to a container that’s full, the primary, simplest way of doing this is making a bigger container, usually twice the size, copying the data from the old one into the new one, and then updating the reference. This process is usually done with a method that is called when needed.



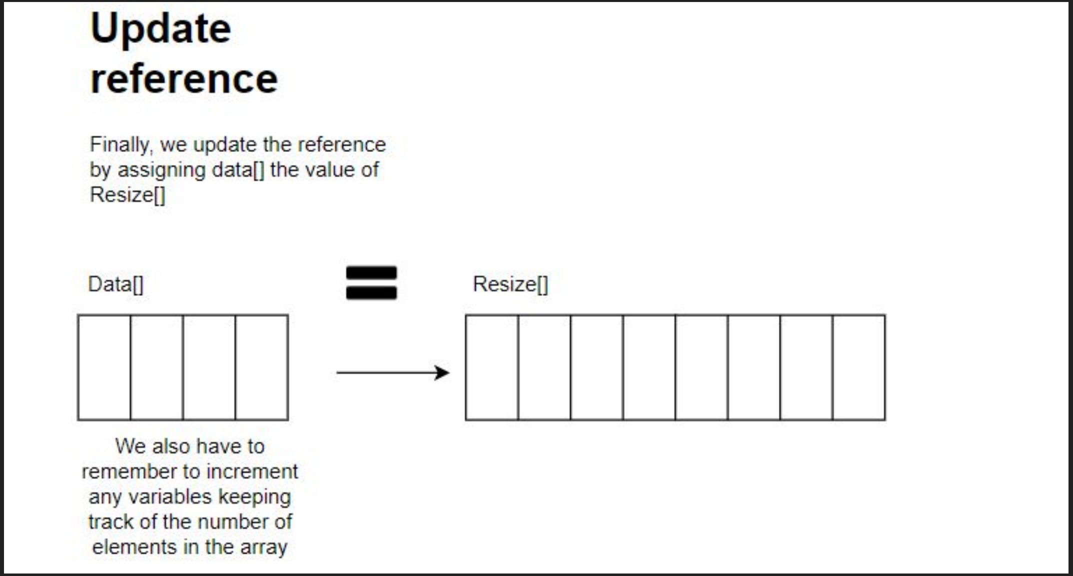
Resize

To add an element to a full array, we have to create a new array, usually twice the size, and copy the elements into the new array

We create a new array and add the new data to the end.

Process of Resizing Arrays

Finally, we update the reference by assigning data[] the value of the new array. When we grab the container we no longer want to grab the smaller one. So we throw the small container out and leave the bigger container on the counter to use.



Update reference

Finally, we update the reference by assigning data[] the value of Resize[]

We also have to remember to increment any variables keeping track of the number of elements in the array

Runtime Analysis

Appending an element to an array n that isn’t full results in a time complexity of O(1), as you simply tack on an extra element.

This changes when the array is full, as a full copy will be required along with doubling the array. The price of doubling is 2n, and copying is n. This will result in a time complexity of O(3n) which simplifies to O(n) time complexity WORST CASE SCENARIO (array is full).

Visuals + References

https://drive.google.com/file/d/1FBrCXwtSD771pcL-OYwgDexsIRs66NXH/view?usp=sharing

Dynamic Arrays 1: https://youtu.be/wXeBVndWA78

Dynamic Arrays 2: https://youtu.be/6ijRCyt28DE

Dynamic Arrays 3: https://youtu.be/XGJeXLlhfdA

https://www.interviewcake.com/concept/java/dynamic-array-amortized-analysis