Discussion related to ArrayList in Java

References: https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/ArrayList.html

ArrayList is a collection. It uses an array as the underlying structure to hold the elements. The size of this array is the capacity of the ArrayList object. If the array becomes full it will grow automatically (it will have a bigger capacity). The *add* operation is required to have *constant amortized cost*.

An ArrayList object has a *capacity* (the maximum number of items it can hold) and a *size* (the actual number of items it holds)

There are two constructors for ArrayList. One creates an array list with a default capacity of 10 and another allows the user to specify the capacity.

Assume you are in charge of implementing the *add* method for the ArrayList and meet the requirement of it having a constant amortized cost.

1. How would you grow the array? Give the answer as a function of the old capacity (that is now full)
2. What is the cost of growing the array? Note that at least in some of the cases you will need to copy all of the elements from the filled out array to the new location where there is enough memory to hold your new array.
3. What is the amortized cost of an add operation?

1. There are at least 2 options for growing the array:

A) by a constant amount. E.g. new_capacity = old_capacity + 100

B) by doubling it. E.g. new_capacity = 2*old_capacity

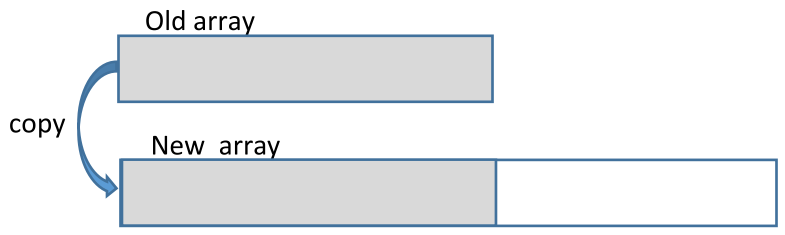
Which one do you think will be more efficient (would require less time to be spent on copying data from the old location to the new one)?

2. The cost to grow the array is the cost of copying every element from the full array to the new location (and any other operations you do as part of resizing). The cost to copy every element is Θ(capacity).

3. The amortized cost is the averaged cost of all operations performed. That is, if N items are inserted in the ArrayList object, the average cost of one add operation is the sum of the cost of each one of the N add operations, divided by N.

$$amortized\ cost\ of\ N\ add\ operations = \frac{\sum_{i=1}^{N}(cost\ to\ add\ item\ i)}{N}$$

Whenever there is space in the array, the add operation takes constant time (it will only need to put the element in the array and update the current size). When the array is full and an add is needed, the array must first grow and then the add can be performed thus this add operation will incur the cost of growing the array (and that is the cost of copying the elements from it, from the old location to the new one).



Since the add performed when there is space available is constant (and so it meets the requirement of constant cost) we will focus on the cost of all the copying and see if that averaged over N will be constant or not for the 2 methods of growing the array listed above.

We will need to keep increasing the capacity (once the array becomes full) for as long as the capacity is less than N. The code relevant to this part alone is:

For method A (new_capacity = old_capacity + 100)

```
for(capacity = 10; capacity<N; capacity=capacity+100){
     ….
     copy_array(capacity,….);  // Θ(capacity)
     ….
}
```

TC_{1iter}(capacity) = Θ(capacity)   dependent

Change of var: capacity: 10, 110,210, 310,…, 10+100x, …, 10+100p=N => p = (N-10)/100

$$\sum_{capacity}\Theta(capacity) = \sum_{x=0}^{p}(10+100x) = 10(p+1) + 100\sum_{x=0}^{p} x = 10p + 10 + 100\frac{p(p+1)}{2} = \Theta(p^2) = \Theta\left(\left(\frac{N-10}{100}\right)^2\right) = \Theta(N^2) \Rightarrow$$

$$amortized\ cost\ for\ N\ add\ operations = \frac{\Theta(N^2)}{N} = \Theta(N)$$

$$which\ is\ NOT\ a\ constant\ therefore\ this\ method\ does\ not\ meet\ the\ requirements$$

For method B (new_capacity = old_capacity * 2)

```
for(capacity = 10; capacity<N; capacity=capacity * 2){
     ….
     copy_array(capacity,….);  // Θ(capacity)
     ….
}
```

TC_{1iter}(capacity) = Θ(capacity)   dependent

Change of var: capacity: 10, 20, 40, 80, 160,…, $10*2^x$, …, $10*2^p=N$ $2^p=N/10 \Rightarrow p = \log_2(N/10)$

$$\sum_{capacity} \Theta(capacity) = \sum_{x=0}^{p}(10 * 2^x) = 10\sum_{x=0}^{p} 2^x = 10\,\frac{2^{p+1}-1}{2-1} = 10(2 * 2^p - 1) = 10(2N/10 - 1) = \Theta(N)$$

To add N elements, for every element (once there is enough space) : N*Θ(1)

Because of all the reallocation and copying we also have cost/time: Θ(N)

From the above 2 lines: N*Θ(1) + Θ(N) = Θ(N). meaning that adding N items has cost/time Θ(N)

$amortized\ cost = \frac{\Theta(N)}{N} = \Theta(1)$ which meets the requirement.

Lessons learnt:

1. Closer look into the behavior of a library collection and understanding of the description given in the documentation for it.
2. Understanding the cost of incrementing by a constant versus doubling. Note that we got Θ(N) because we doubled up to N itself. If we were doubling up to 2^N, we would have $\Theta(2^N)$. Knowing these summations will help you guide your search/solution for a problem that requires a specific time or space complexity.

Getting started with this analysis:

Assume the starting array is created as follows:

int* arr = malloc(10*sizeof(int));

Operation add(e) will be performed N times (to add N elements). We assume no element will be removed. At start, the capacity is 10:

Adding the 1st ,2nd ,…, 10th elements will be Θ(1)   (arr[sz]=e; sz++;)

When out of space, reallocate for an array of size old_size+10 (e.g. 10 + 100)

10 => 110

Reallocate:

arr = realloc(arr, 110); or narr = malloc(110*sizeof(int)) and copy narr[j]=arr[j] (e.g. j:0->9), arr=narr

What is the cost to reallocate and copy the data? Assume we copy the data: $\Theta(N)$ where N is the old capacity

11-th element, must reallocate for 110 (and copy 10)

111-th element, must reallocate for 210 (and copy 110)

211-th element, must reallocate for 230 (and copy 210)

….

How much work do we do for all the above copying?