Constant Amortized Time

What is meant by "Constant Amortized Time" when talking about time complexity of an algorithm?

algorithm complexity-theory big-o

edited Oct 14 '08 at 9:52
Hank Gay
48.4k ● 22 ● 130 ● 204

asked Oct 14 '08 at 8:32
VarunGupta
2,162 ● 3 ● 22 ● 28

mortoray.com/2014/08/11/what-is-amortized-time – edA-qa mort-ora-y Jan 14 '15 at 7:20

5 Answers

Amortised time explained in simple terms:

If you do an operation say a million times, you don't really care about the worst-case or the best-case of that operation - what you care about is how much time is taken in total when you repeat the operation a million times.

So it doesn't matter if the operation is very slow once in a while, as long as "once in a while" is rare enough for the slowness to be diluted away. Essentially amortised time means "average time taken per operation, if you do many operations". Amortised time doesn't have to be constant; you can have linear and logarithmic amortised time or whatever else.

Let's take mats' example of a dynamic array, to which you repeatedly add new items. Normally adding an item takes constant time (that is, `O(1)`). But each time the array is full, you allocate twice as much space, copy your data into the new region, and free the old space. Assuming allocates and frees run in constant time, this enlargement process takes `O(n)` time where n is the current size of the array.

So each time you enlarge, you take about twice as much time as the last enlarge. But you've also waited twice as long before doing it! The cost of each enlargement can thus be "spread out" among the insertions. This means that in the long term, the total time taken for adding *m* items to the array is `O(m)`, and so the amortised time (i.e. time per insertion) is `O(1)`.

edited May 5 '09 at 18:09
Motti
61.5k ● 28 ● 145 ● 216

answered Oct 30 '08 at 9:48
Artelius
38.3k ● 7 ● 70 ● 90

27 Just a note in terms of notation: An amortized constant execution time of O(n) is often written as O(n)+, as opposed to just O(n). The addition of the plus sign indicates that that execution time is not guaranteed to be O(n) and can actually exceed that execution time. – Jeffpowrs Apr 8 '14 at 20:55

In terms of allocating space, is that from the heap? – committedandroider Dec 29 '14 at 6:26

It means that over time, the worst case scenario will default to O(1), or constant time. A common example is the dynamic array. If we have already allocated memory for a new entry, adding it will be O(1). If we haven't allocated it we will do so by allocating, say, twice the current amount. This particular insertion will *not* be O(1), but rather something else.

What is important is that the algorithm guarantees that after a sequence of operations the expensive operations will be amortised and thereby rendering the entire operation O(1).

Or in more strict terms,

> There is a constant c, such that for *every* sequence of operations (also one ending with a costly operation) of length L, the time is not greater than c*L (Thanks Rafał Dowgird)

edited May 23 at 10:31

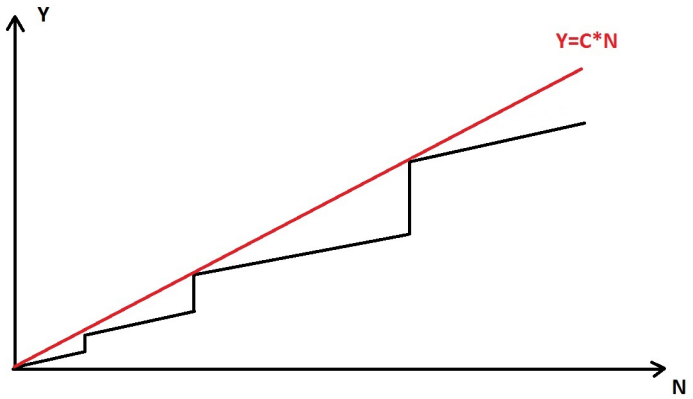
Community ♦
1 ● 1

answered Oct 14 '08 at 8:48

Mats Fredriksson
13.4k ● 6 ● 30 ● 54

9 "after a large enough amount of operations" - constant amortized time doesn't need this condition. There is a constant c, such that for *every* sequence of operations (also one ending with a costly operation) of length L, the time is not greater than c*L. – Rafał Dowgird Oct 14 '08 at 9:35

To develop an intuitive way of thinking about it, consider insertion of elements in dynamic array (for example `std::vector` in C++). Let's plot a graph, that shows dependency of number of operations (Y) needed to insert N elements in array:



Vertical parts of black graph corresponds to reallocations of memory in order to expand an array. Here we can see that this dependency can be roughly represented as a line. And this line equation is `Y=C*N + b` (`C` is constant, `b` = 0 in our case). Therefore we can say that we need to spend `C*N` operations on average to add N elements to array, or `C*1` operations to add one element (amortized constant time).

answered Jan 19 at 10:17

Megamozg
409 ● 4 ● 10

I found below Wikipedia explanation useful, after repeat reading for 3 times:

Source: https://en.wikipedia.org/wiki/Amortized_analysis#Dynamic_Array

> "Dynamic Array
>
> Amortized Analysis of the Push operation for a Dynamic Array
>
> Consider a dynamic array that grows in size as more elements are added to it such as an ArrayList in Java. If we started out with a dynamic array of size 4, it would take constant time to push four elements onto it. Yet pushing a fifth element onto that array would take longer as the array would have to create a new array of double the current size (8), copy the old elements onto the new array, and then add the new element. The next three push operations would similarly take constant time, and then the subsequent addition would require another slow doubling of the array size.
>
> In general if we consider an arbitrary number of pushes n to an array of size n, we notice that push operations take constant time except for the last one which takes O(n) time to perform the size doubling operation. Since there were n operations total we can take the average of this and find that for pushing elements onto the dynamic array takes: O(n/n)=O(1), constant time."

edited Aug 29 '16 at 6:49

answered Jul 8 '16 at 7:42

Manohar Reddy Poreddy
1,796 ● 17 ● 21

Ah, so it's O(worst case / # of operations). I like this answer the best. – Jason Tu Oct 27 '16 at 3:17

The explanations above apply to Aggregate Analysis, the idea of taking "an average" over multiple operations. I am not sure how they apply to Bankers-method or the Physicists Methods of Amortized analysis.

Now. I am not exactly sure of the correct answer. But it would have to do with the principle condition of the both Physicists+Banker's methods:

(Sum of amortized-cost of operations) >= (Sum of actual-cost of operations).

The chief difficulty that I face is that given that Amortized-asymptotic costs of operations differ from the normal-asymptotic-cost, I am not sure how to rate the significance of amortized-costs.

That is when somebody gives my an amortized-cost, I know its not the same as normal-asymptotic cost What conclusions am I to draw from the amortized-cost then?

Since we have the case of some operations being overcharged while other operations are undercharged, one hypothesis could be that quoting amortized-costs of individual operations would be meaningless.

For eg: For a fibonacci heap, quoting amortized cost of just Decreasing-Key to be O(1) is meaningless since costs are reduced by "work done by earlier operations in increasing potential of the heap."

OR

We could have another hypothesis that reasons about the amortized-costs as follows:

1. I know that the expensive operation is going to preceded by MULTIPLE LOW-COST operations.
2. For the sake of analysis, I am going to overcharge some low-cost operations, SUCH THAT THEIR ASYMPTOTIC-COST DOES NOT CHANGE.
3. With these increased low-cost operations, I can PROVE THAT EXPENSIVE OPERATION has a SMALLER ASYMPTOTIC COST.
4. Thus I have improved/decreased the ASYMPTOTIC-BOUND of the cost of n operations.

Thus amortized-cost analysis + amortized-cost-bounds are now applicable to only the expensive operations. The cheap operations have the same asymptotic-amortized-cost as their normal-asymptotic-cost.

edited May 10 '13 at 22:45

answered May 10 '13 at 20:38

Misraji

21 ● 2