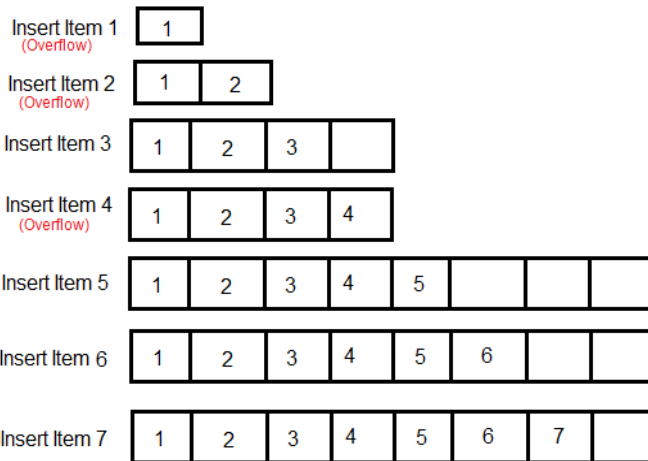


Analysis of Algorithm | Set 6 (Amortized Analysis Introduction) - By GeeksForGeeks

Amortized Analysis is used for algorithms where an occasional operation is very slow, but most of the other operations are faster. In Amortized Analysis, we analyze a sequence of operations and guarantee a worst case average time which is lower than the worst case time of a particular expensive operation. The example data structures whose operations are analyzed using Amortized Analysis are Hash Tables, Disjoint Sets and Splay Trees.

Let us consider an example of a simple hash table insertions. How do we decide table size? There is a trade-off between space and time, if we make hash-table size big, search time becomes fast, but space required becomes high.

Initially table is empty and size is 0



Next overflow would happen when we insert 9, table size would become 16

The solution to this trade-off problem is to use **Dynamic Table (or Arrays)**. The idea is to increase size of table whenever it becomes full. Following are the steps to follow when table becomes full.

- 1) Allocate memory for a larger table of size, typically twice the old table.
- 2) Copy the contents of old table to new table.
- 3) Free the old table.

If the table has space available, we simply insert new item in available space.

What is the time complexity of n insertions using the above scheme?

If we use simple analysis, the worst case cost of an insertion is $O(n)$. Therefore, worst case cost of n inserts is $n * O(n)$ which is $O(n^2)$. This analysis gives an upper bound, but not a tight upper bound for n insertions as all insertions don't take $\Theta(n)$ time.

Item No.	1	2	3	4	5	6	7	8	9	10
Table Size	1	2	4	4	8	8	8	8	16	16
Cost	1	2	3	1	5	1	1	1	9	1

$$\text{Amortized Cost} = \frac{(1 + 2 + 3 + 5 + 1 + 1 + 9 + 1 \dots)}{n}$$

We can simplify the above series by breaking terms 2, 3, 5, 9.. into two as (1+1), (1+2), (1+4), (1+8)

$$\begin{aligned} \text{Amortized Cost} &= \frac{[\underbrace{(1 + 1 + 1 + 1 \dots)}_{n \text{ terms}}] + [\underbrace{(1 + 2 + 4 + \dots)}_{\lfloor \log_2(n-1) \rfloor + 1 \text{ terms}}]}{n} \\ &\leq \frac{[n + 2n]}{n} \\ &\leq 3 \end{aligned}$$

$$\text{Amortized Cost} = O(1)$$

So using Amortized Analysis, we could prove that the Dynamic Table scheme has $O(1)$ insertion time which is a great result used in hashing. Also, the concept of dynamic table is used in **vectors in C++, ArrayList in Java**.

Following are few important notes.

- 1) Amortized cost of a sequence of operations can be seen as expenses of a salaried person. The average monthly expense of the person is less than or equal to the salary, but the person can spend more money in a particular month by buying a car or something. In other months, he or she saves money for the expensive month.
- 2) The above Amortized Analysis done for Dynamic Array example is called **Aggregate Method**. There are two more powerful ways to do Amortized analysis called **Accounting Method** and **Potential Method**. We will be discussing the other two methods in separate posts.
- 3) The amortized analysis doesn't involve probability. There is also another different notion of average case running time where algorithms use randomization to make them faster and expected running time is faster than the worst case running time. These algorithms are analyzed using Randomized Analysis. Examples of these algorithms are Randomized Quick Sort, Quick Select and Hashing. We will soon be covering Randomized analysis in a different post.

Sources:

Berkeley Lecture 35: Amortized Analysis

MIT Lecture 13: Amortized Algorithms, Table Doubling, Potential Method

<http://www.cs.cornell.edu/courses/cs3110/2011sp/lectures/lec20-amortized/amortized.htm>