CS2102 Analysis and Design of Algorithms

Amortized Analysis

M.Sc. Bryan Gonzales Vega
bgonzales.vega@gmail.com

University of Engineering and Technology

Lecture Content

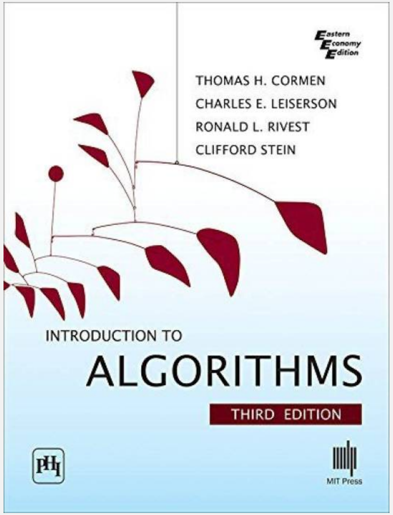
Amortized Analysis

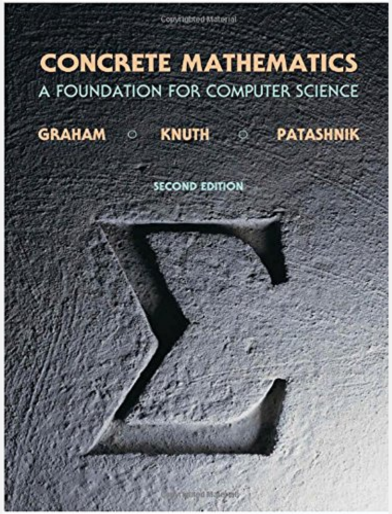
Introduction to Algorithms
[Cormen et al., 2009]

- Chap 17: Amortized Analysis

Concrete Mathematics
[Graham et al., 1989]

- Chap 02: Sums
- Chap 03: Integer Functions

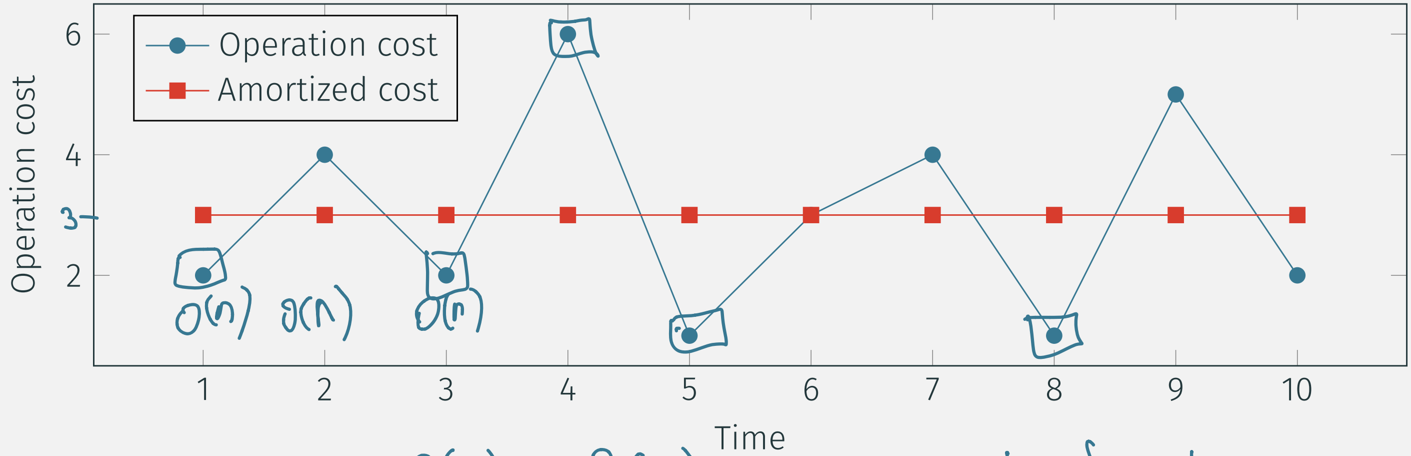




Amortized Analysis is a worst case analysis over a sequence of operations to get the overall cost per operation. There are 3 types of methods: **aggregate** analysis, **accounting** method and **potential** method.

General Idea

If some uncommon and expensive operation occurs at some moment, we should contrast that cost against the others since it may balance the overall performance.



Avg. worst case: $n \cdot O(n) = O(n^2)$ ≠ amortized analysis

This method considers the total running time for a sequence of operations following the steps below:

1. Calculate the total cost of all the n operations as $T(n)$
2. Calculate the average cost of each operation as $\frac{T(n)}{n}$

Aggregate method considers that each operation has the same cost (amortized)

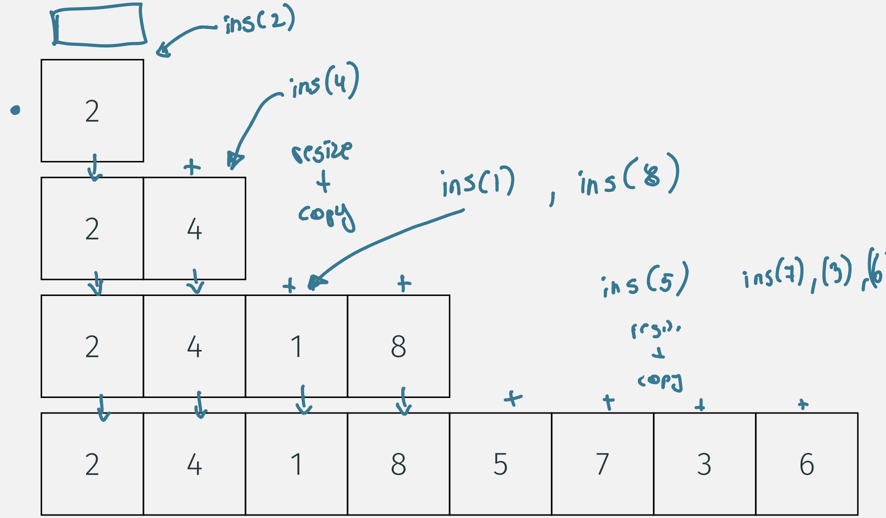
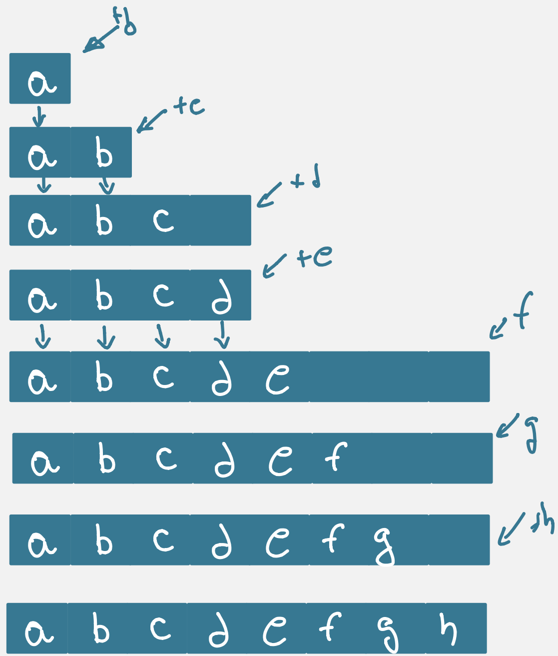


Figure 1: Expansion of the capacity of a dynamic table over 8 insertion operations. Just in some moments the array needs to allocate more space and copy previous elements.

$$\sum \left(\text{cheap ops} + \text{expensive ops} \right) \leq 2n \approx O(n)$$

Consider the aggregate method to calculate the amortized cost of each operation of a binary adder.



i	cost (c_i)
1	1 (insert)
2	2 (resize + insert)
3	3 (resize + insert)
4	1 (insert)
5	5 (resize + insert)
6	1 (insert)
7	1 (insert)
8	1 (insert)

$$c_i = \begin{cases} i, & (i-1) \text{ is power of } 2 \\ 1, & \text{otherwise} \end{cases}$$

$$\text{amortized cost} = \frac{T(n)}{n} = \frac{\sum_{i=1}^{n} c_i}{n} = \frac{15}{8} \approx 2 = \hat{c}_i$$

$$\hat{c}_i = \Theta(1)$$

Account Method

bank account

This method is based in some ideas taken from accounting. It defines an overcharged cost to each operation with the intention that the remaining will contribute to future operations.

Intuition: Low cost and frequent operations are charged more than high cost and less frequent operations.

Given c_i as the actual cost and $\hat{c}_i$ as the charged cost of the ith operation, then for all n we would like to:

invariant : $$\sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} \hat{c}_i$$ balance shouldn't be negative

Where the amortized cost acts as an upper bound to the actual cost of the operation.

Potential Method

Also know as the physicist method, proposes a function $\Phi : \{D_i\} \to \Re$ that defines the state of a data structure D such that:

- $\Phi(D_0) = 0$
- $\Phi(D_i) \geq 0, \forall i$

And the amortized cost $\hat{c}_i$ with respect to Φ as:

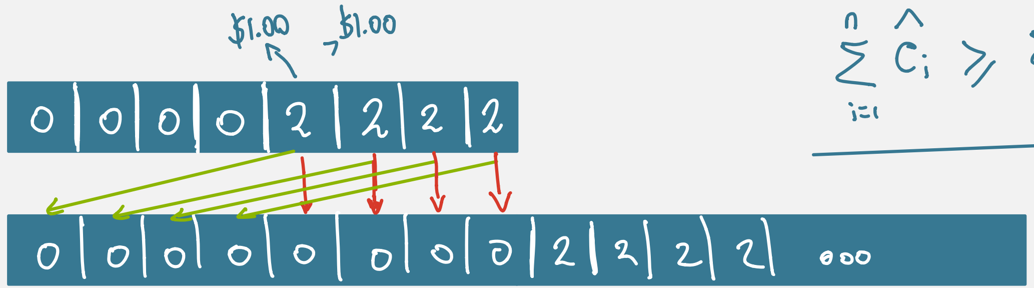
$$\hat{c}_i = c_i + \Delta(\Phi_i)$$
$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$$

So, over n operations the total amortized cost will be represented by:

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} (c_i + \Phi(D_i) - \Phi(D_{i-1}))$$

Account Method Intuition

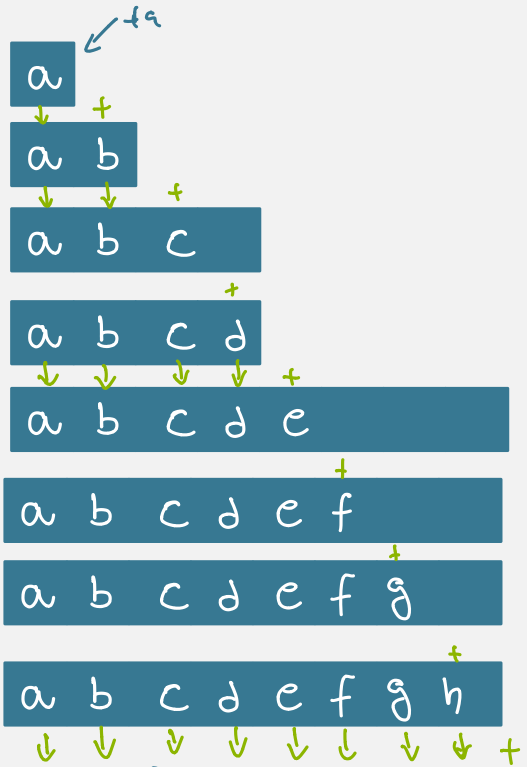
$$\hat{c}_i = \$3.00 \quad \begin{cases} \$1.00 \text{ for insert} \\ \$2.00 \text{ stored for later use (copy old elements)} \end{cases}$$



$$\sum_{i=1}^{n} \hat{c}_i \geq \sum_{i=1}^{n} c_i$$

Given an amortized cost $\hat{c}_i$, we want to check that the bank balance supports the sequence of operations.

Account Method



cost($\hat{c}_i$)	balance(\$)
3	$2 = 3 - 1$
3	$3 = 2 + 3 - 1 - 1$
3	$3 = 3 + 3 - 2 - 1$
3	$5 = 3 + 3 - 1$
3	$3 = 5 + 3 - 4 - 1$
3	$5 = 3 + 3 - 1$
3	$7 = 5 + 3 - 1$
3	$9 = 7 + 3 - 1$
3	$3 = 9 + 3 - 8 - 1$

$\hat{c}_i = 3 \approx \Theta(1)$

↪ move 8 elements are already paid.

Account Method

bank account

This method is based in some ideas taken from accounting. It defines an overcharged cost to each operation with the intention that the remaining will contribute to future operations.

Intuition: Low cost and frequent operations are charged more than high cost and less frequent operations.

Given c_i as the actual cost and $\hat{c}_i$ as the charged cost of the ith operation, then for all n we would like to:

invariant :
$$\sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} \hat{c}_i$$
balance shouldn't be negative

Where the amortized cost acts as an upper bound to the actual cost of the operation.

Potential Method

Also know as the physicist method, proposes a function $\Phi : \{D_i\} \rightarrow \Re$ that defines the state of a data structure D such that:

- $\Phi(D_0) = 0$
- $\Phi(D_i) \geq 0, \forall i$

And the amortized cost $\hat{c}_i$ with respect to Φ as:

$$\hat{c}_i = c_i + \Delta(\Phi_i)$$
$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$$

So, over n operations the total amortized cost will be represented by:

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} (c_i + \Phi(D_i) - \Phi(D_{i-1}))$$

- Bank account → Potential energy

Given a function that maps the bank account we need to find the amortized cost $\hat{c}_i$.

i-th operation transform $D_{i-1} \to D_i$

$$\hat{c}_i = c_i + \Delta(\Phi_i)$$

$$\hat{c}_i = c_i + \underbrace{\Phi(D_i) - \Phi(D_{i-1})}_{\text{potential difference}}$$

$\Delta\phi_i > 0 \Rightarrow \hat{c}_i > c_i$, it charges more than the actual cost, save energy for later

$\Delta\phi_i < 0 \Rightarrow \hat{c}_i < c_i$, release energy to afford operation

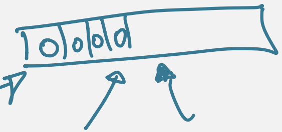
$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n}(c_i + \Phi(D_i) - \Phi(D_{i-1})) = \sum_{i=1}^{n} c_i + \underbrace{\Phi(D_n)}_{\geq 0} - \underbrace{\Phi(D_0)}_{0}$$

$1..n$ $0..n-1$

$$\sum_{i=1}^{n} \hat{c}_i \geq \sum_{i=1}^{n} c_i$$

Cont'd

$\Phi(D_i) = 2i - \text{size}$ $\rightarrow$ capacity i



$\Phi_4 = 0$

$$\boxed{\Phi(D_i) = 2i - 2^{\lceil \log i \rceil}} \checkmark$$

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$$

$$\hat{c}_i = c_i + (2i - \text{size}) - (2(i-1) - \text{size})$$

Case 1: // insertion

$$\hat{c}_i = 1 + (2i - \text{size}) - (2(i-1) - \text{size})$$

$$\hat{c}_i = 1 + \cancel{2i} - \cancel{\text{size}} - \cancel{2i} + 2 + \cancel{\text{size}}$$

$$\hat{c}_i = 3$$

Case 2: // Expension (copy + insertion)

$$\hat{c}_i = \underbrace{(i+1)}_{i\text{ copies} + 1\text{ insert}} + (2i - \overset{\text{size}}{\underbrace{2i}_{\text{after}}}) - (2(i-1) - \overset{\text{size}}{\underbrace{i}_{\text{before}}})$$

$$\hat{c}_i = \cancel{i} + 1 + \cancel{2i} - \cancel{2i} - \cancel{2i} + 2 + \cancel{i}$$

$$\hat{c}_i = 3$$

$$\Theta(1)$$

References i

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009).
Introduction to algorithms.
MIT press.

Graham, R., Graham, R., Knuth, D., Knuth, D., and Patashnik, O. (1989).
Concrete Mathematics: A Foundation for Computer Science.
A foundation for computer science. Addison-Wesley.

Cont’d