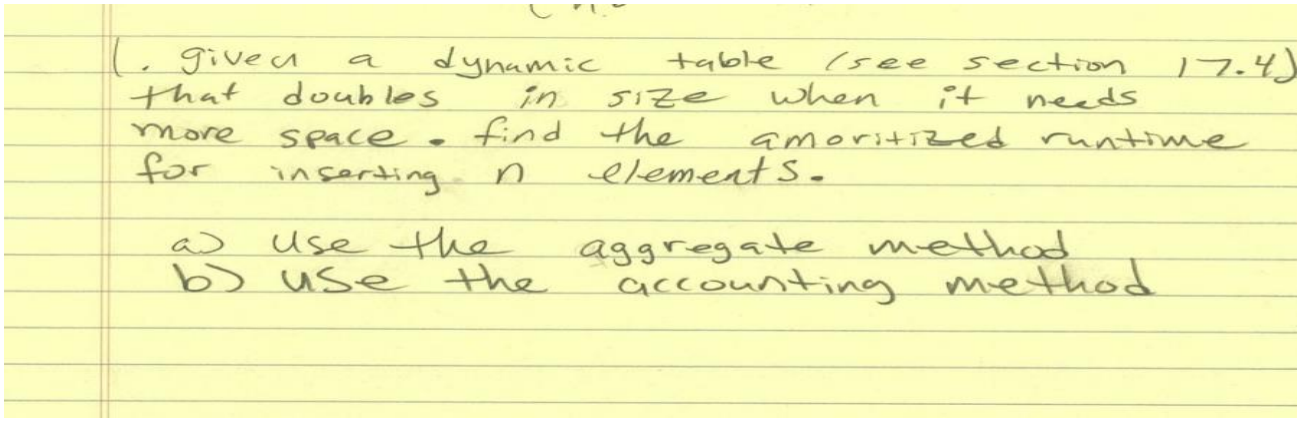


⇨ The **amortized runtime for inserting *n* elements into a dynamic table that doubles in size when full**, using both the **aggregate method** and the **accounting method**:

---

a) Aggregate Method:

Let's say we insert **n** elements into the table, and it starts empty. When the array is full, it doubles in size and copies the elements over.

Each insertion normally costs **1**, but when we double, we have to copy all existing elements.

⇨ Let's count the **total cost** over **n** insertions:

- Insert 1st element: cost = 1 (plus copy 0 items)
- Insert 2nd: copy 1 item (cost = 2)
- Insert 3rd: cost = 1
- Insert 4th: copy 2 items (cost = 3)
- Insert 5th to 7th: cost = 1 each
- Insert 8th: copy 4 items (cost = 5)
- ...

⇨ Every doubling at size 2^K causes 2^K copies. This happens at sizes:

- 1 → cost 1
- 2 → cost 2
- 4 → cost 4

- 8 → cost 8
- ... up to ≤n

⇨ So total copy cost is:

$$\sum_{k=0}^{\log n} 2^k = 2^{\log n+1} - 1 = 2n - 1$$

⇨ Add in the **n** simple insertions:

$$Total\ cost = n + (2n - 1) = 3n - 1$$

⇨ $\boldsymbol{Amortized\ cost\ per\ insertion\ =\ (3n\ -\ 1)/n\ =\ O(1)}$

(b) Accounting Method:

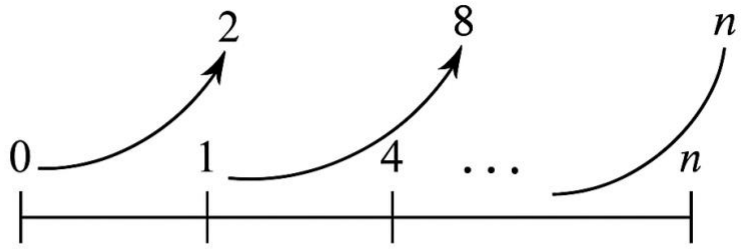
⇨ We assign an **amortized cost** of 3 to each insertion:

- 1 to pay for the insertion itself.
- 2 are saved ("prepaid") to pay for future copying.

⇨ Every time an element is copied during resizing, it has 2 saved units of cost from its original insertion. So, we can afford the cost of copying.
⇨ Thus, with each insert costing 3 units (but only using 1 unless there's a resize), we never run out of "credits" to pay for copying.

⇨ $\boldsymbol{Amortized\ cost\ per\ insert\ =\ 3\ \rightarrow\ O(1)}$

⇨ **The visual representation of both method is follow:**



(a) Aggregate method

(b) Accounting method

Amortized runtime for inserting n elements