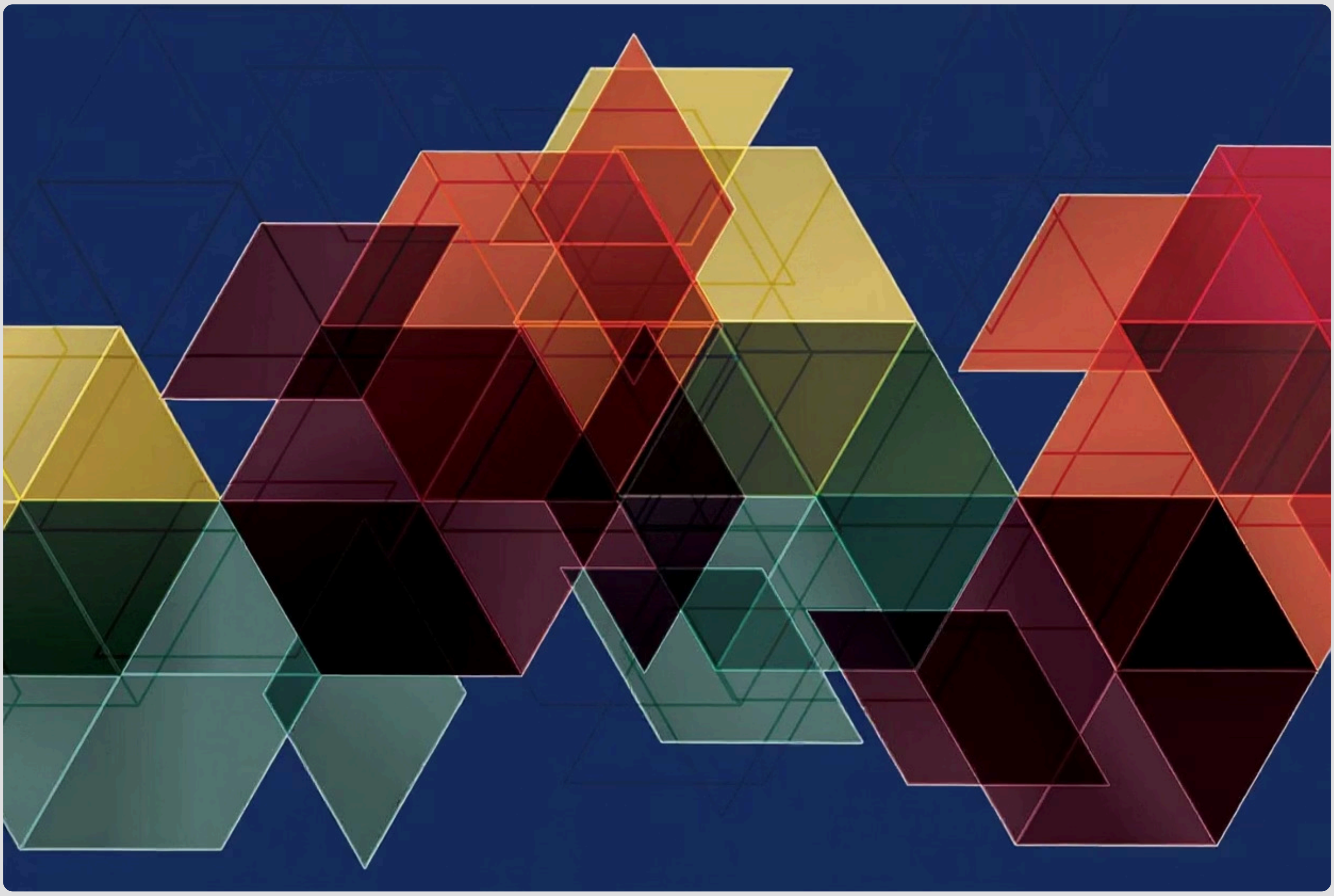
Dynamic Array Resizing: The Power of Geometric Growth

Understanding why geometric growth strategies (like doubling) achieve optimal amortized performance—and why most programming languages use this approach for dynamic arrays.



The Solution: Geometric Growth Strategy

Modern Approach

When a dynamic array runs out of space, we must create a new, larger array and copy all existing elements. The optimal strategy is to multiply the current capacity by a constant factor (typically 2).

This geometric growth approach, where we double the array size each time, provides excellent amortized performance and is used by virtually all modern programming language implementations.

The Mathematical Advantage

If we double the array size each time we resize, the mathematics work in our favor:

- After t resizes, capacity becomes $2^t \cdot C_0$ (where C_0 is initial capacity)
- For n elements, we need $2^t \geq n$
- Therefore: $t \leq \log_2(n)$ resizes required

This logarithmic relationship is key to achieving optimal performance.

When a dynamic array is resized by doubling, the total number of elements copied over time forms a geometric series. Let's trace the work done:

- 1st resize: C_0 elements copied (initial capacity).
- 2nd resize: $2C_0$ elements copied.
- 3rd resize: $4C_0$ elements copied.
- ...
- t-th resize: $2^{t-1}C_0$ elements copied.

The total number of copy operations after t resizes is the sum of a geometric series:

$$\sum_{i=0}^{t-1} 2^i C_0 = C_0 \sum_{i=0}^{t-1} 2^i = C_0 \frac{2^t - 1}{2 - 1} = C_0(2^t - 1)$$

Since the final capacity is $2^t C_0$ and contains n elements, we know that $2^{t-1}C_0 < n \leq 2^t C_0$. This means $2^t C_0 < 2n$, so:

$$\text{Total Copies} < C_0(2n/C_0) = 2n$$

Therefore, the total work done (copy operations) is proportional to n. This leads to a time complexity of $\Theta(n)$ for n insertions, making geometric growth highly efficient even for very large datasets.

Amortised Analysis: Optimal Cost Per Operation

Amortised analysis examines the average cost per operation across a sequence of operations:

$$Amortised\ cost = \frac{Total\ cost}{Number\ of\ operations}$$

From our previous calculation:

- Total cost: $\Theta(n)$
- Number of operations: n
- Amortised cost per operation: $\Theta(1)$

This constant amortised time per operation is optimal—we cannot do better than $O(1)$ amortised cost for dynamic array insertions.

$$O(n)$$

Total Time

For n insertions

$$O(1)$$

Per Operation

Amortised cost

Key Advantages of Geometric Growth

Optimal Strategy

Geometric growth (doubling) provides the best balance of space efficiency and time performance.

Linear Complexity

The total work for n insertions is $\Theta(n)$ due to the geometric series properties.

Constant Amortised Cost

Each operation, on average, takes $\Theta(1)$ time, making it highly practical for datasets of any size.