Data Structures (Algorithms)

Algorithms — run once — worst-case running time

Data Structures — build once

run operations many times

worst-case time for each operation

aggregate/total time for seq. of operations

AMORTIZED TIME

---

\( F(n) = O(g(n)) \) means

There are constants \( N \) and \( c \)

s.t. For all \( n \geq N \)

we have \( F(n) \leq c \cdot g(n) \)

\[
\liminf_{n \to \infty} \frac{F(n)}{g(n)} < \infty
\]

\( F(n) = \Omega(g(n)) \quad \iff \quad F(n) = \Theta(g(n)) \quad \text{O}(g(n)) \text{ and } \Omega(g(n)) \)

---

Amortized time

Suppose data structure has 3 operations \( A, B, C \)

" \( A \) has \( \text{am.-time } O(a(n)) \)

\( B \) runs in \( \text{am. time } O(b(n)) \)

\( C \) runs in \( \text{am. time } O(c(n)) \) "

\( \iff \)

Any sequence of \( N_A \) \( A \)'s + \( N_B \) \( B \)'s + \( N_C \) \( C \)'s

runs in \( O(N_A \cdot a(n) + N_B \cdot b(n) + N_C \cdot c(n) ) \) time
Suppose AL has no items at start of phase

| no full | no empty |

when we double data array, it's full

\[ \text{num} = 2\text{no} \]

\[ \Rightarrow \text{we did no inserts} \]

Time to double the array = \( O(2\text{no}) = O(n) \)
Charging argument

Each cheap/fast operation pays in advance for future expensive/slow operations.

Expensive ops charge earlier cheap ops