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 $\boxed{F(n) \leq c \cdot g(n)}$

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

$F(n) = \Omega(g(n))$ ——— $\geq$ ———

$F(n) = \Theta(g(n))$ $O(g(n))$ and $\Omega(g(n))$

↳ Amortized time

Spose data structure has 3 operations A, B, C

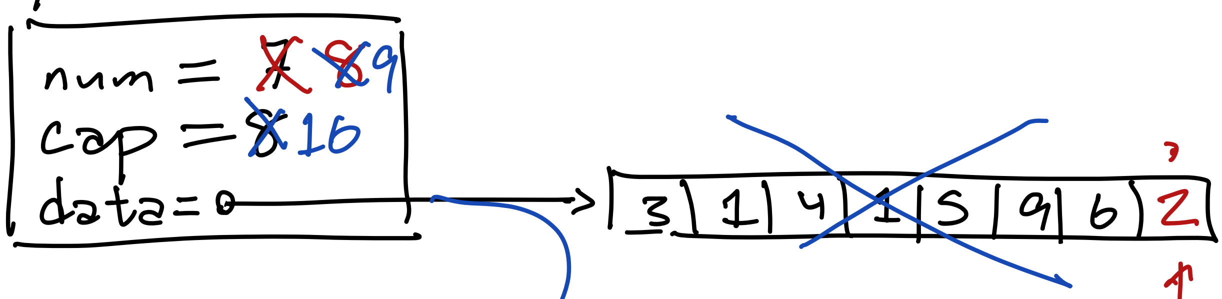
" A has am-time $O(a(n))$
B runs in am time $O(b(n))$
C runs in am time $O(c(n))$ "

$\Longleftrightarrow$

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

Array List

AL.data[5] = 1

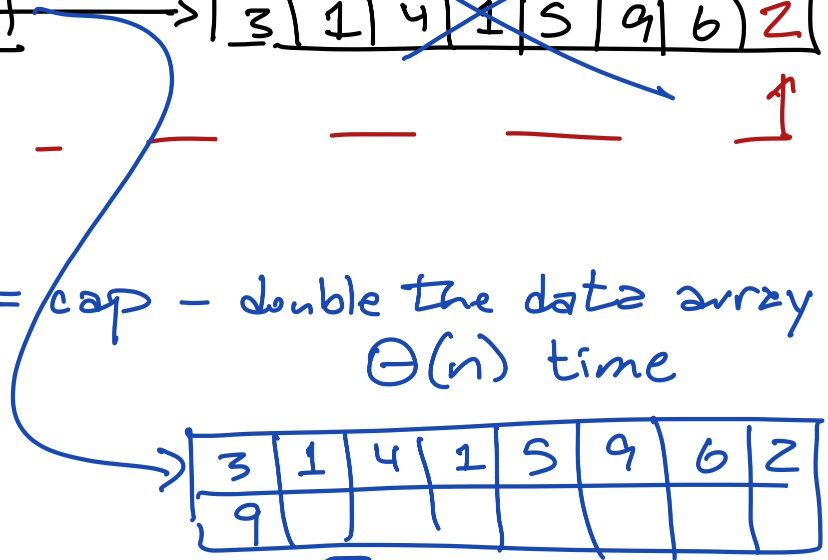


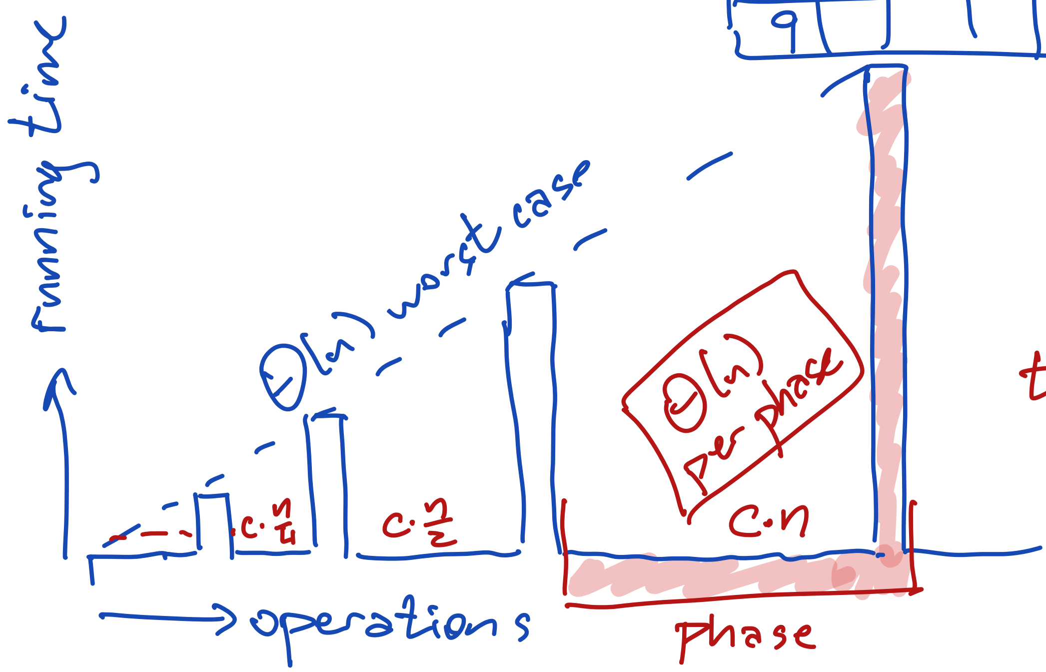
INSERT(2)

INSERT(9)

when num = cap – double the data array

$\Theta(n)$ time

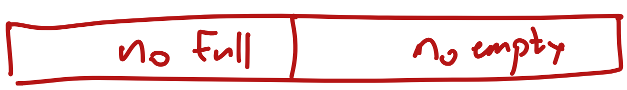




$$\text{total} = \Theta\Big(\underbrace{cn + c\tfrac{n}{2} + c\tfrac{n}{4} \cdots}\Big)$$

$$= \Theta(n)$$

Spose AL has n_0 items at start of phase



when we double data array, it's full $\quad num = 2n_0$

$\Rightarrow$ we did n_0 INSERTS

Time to double the array $= O(2n_0) = O(n_0)$

Charging argument

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

Expensive ops charge earlier cheap ops