# COL 702 Lecture 6

Potential function of a (data structure) state

$\phi : S \rightarrow \mathbb{R}$

Initial state is $S_0$

$\phi(S_0) : \phi_0$

$S_0 \rightarrow S_1 \rightarrow S_2 \ldots$

$\phi_0 \quad \phi_1 \quad \phi_2 \ldots$

let the work done be denoted by $w_i$ when we move from $S_i$ to $S_{i+1}$ which is the actual work done

Amortized work : $A_i = w_i + \underbrace{\phi(S_{i+1}) - \phi(S_i)}_{\text{change in potential}}$

Total work done over a sequence of $n$ updates

$$W = \sum_{i=0}^{n-1} w_i$$

Total Amortized work : $\sum_i (w_i + \phi_{i+1} - \phi_i)$

$= W + [\phi_1 - \phi_0 + \phi_2 - \phi_1 \ldots \phi_n - \phi_{n-1}]$

$\phi_n - \phi_0$

Total amortized work = Actual work + $\phi_f - \phi_I$

$\phi_f$ : final $\phi_I$ : initial

If $\phi_f - \phi_I \geq 0$ $\Rightarrow$ Total work $\leq$ Amortized work

( In general Total work = Amortized work + $\phi_I - \phi_f$ )

Example : Stack operations push, pop, Empty stack

$\phi$ (Stack) : # of elements in stack

$\phi_I$ (Empty stack) = 0 $\phi_f \geq \phi_I$

Push : Actual work : 1 Amortized work
1 + 1 (change in potential)
= 2

(Scaling by factor c doesn't change the calcul.)

Pop : Actual work : 1 Amortized work
1 − 1 = 0

Empty stack : Actual work #elements
Amortized work : #elements − #elem'
= 0

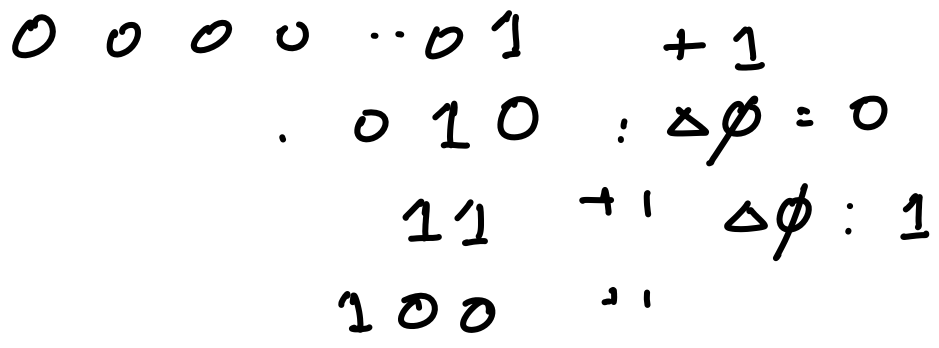
Cost of m stack operation $\leq$ amortized cost of m opns
$\leq 2 \cdot m$

## Example 2: Counter Problem

Potential function: # bits equal to 1

Amortized cost of increment:
(worst case)

# bits that flip to 1 − # bits that flip to 0




0111111
1000000

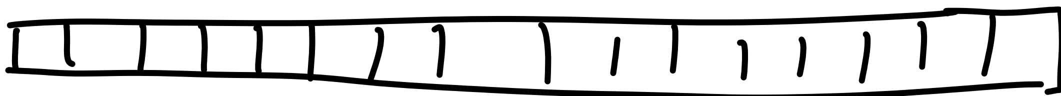
Amortized cost $\leq 1$

Application to Binomial Heaps:

Suppose we keep inserting elements to a Binomial Heap: $O(1)$ from analogy with the counter

# Semi dynamic dictionary problem

Supports insertions and search
(Not deletion)
We will prefer using arrays



<u>Goal</u>: Support search and insertion in $O(\log n)$ time $\begin{cases} O(\log^2 n) \\ O(\log^3 n) \end{cases}$

<u>Idea</u>: Suppose we store the elements in multiple sorted arrays

Cost of search : $\sum_i \log(n_i)$

$n_i$ is the size of array $i$

For example with $\log n$ arrays, we can bound by $O(\log n \times \log n) = O(\log^2 n)$

<u>Insertion</u>: If an existing array has extra space, insert it in the array or else create a new array "of twice the size"

Consider arrays of size $2^i$, $i \geq 0$

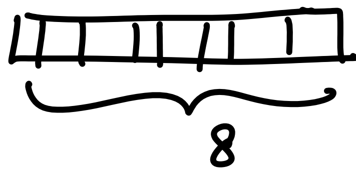
Like in Binomial heaps, we can repr any number $n$ using at most $\log n$ such arrays $\Rightarrow$ Search-time $O(\log^2 n)$

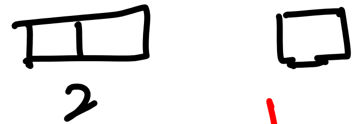
(At most one array of size $2^i$) ($\log n$ binary search)

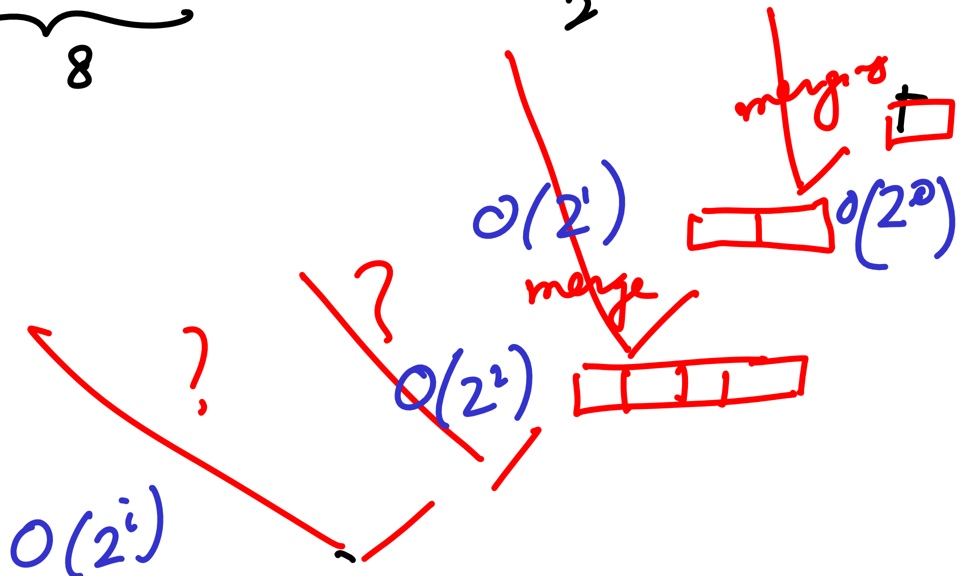
**Insertion**: Create an array of size 1 and "merge" with the existing arrays

Until there are no array of size $2^i$ starting with $i = 0$
we merge two arrays of sizes $2^i$

11 elements



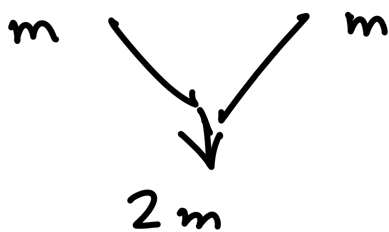
Worst case cost: $\sum_{1 \le j \le k} O(2^j)$

$$\le O(2^{k+1}) \sim O(2^k)$$

**Amortized cost?**

Given $O(\log n)$ amortized cost

When we merge two arrays of size $m$




Cost is $O(2m)$

$O(\#\text{ elements involved})$

Let us pretend that we have a counter with each element that stores the cost of merging

$\sum c_i$ : cost of operations = running time

Define amortized cost on the charging and analyse the algorithm

Define potential function for each element