$A[\,] = \{5, 6, 7, 8, 9\}$

# Static Interfaces

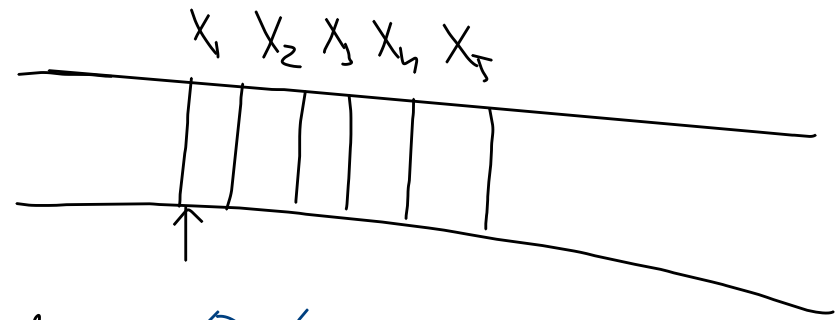
- build (X)
- iterate (X)
- get_at (i)
- Set_at (x, i)

$\langle x_1, x_2, \cdots, x_n \rangle$

O/ps supported

$x_1 - \cdots - x_5$

(Word RAM Model)




build → $O(n)$

iterate → $O(n)$

get_at → $O(1)$

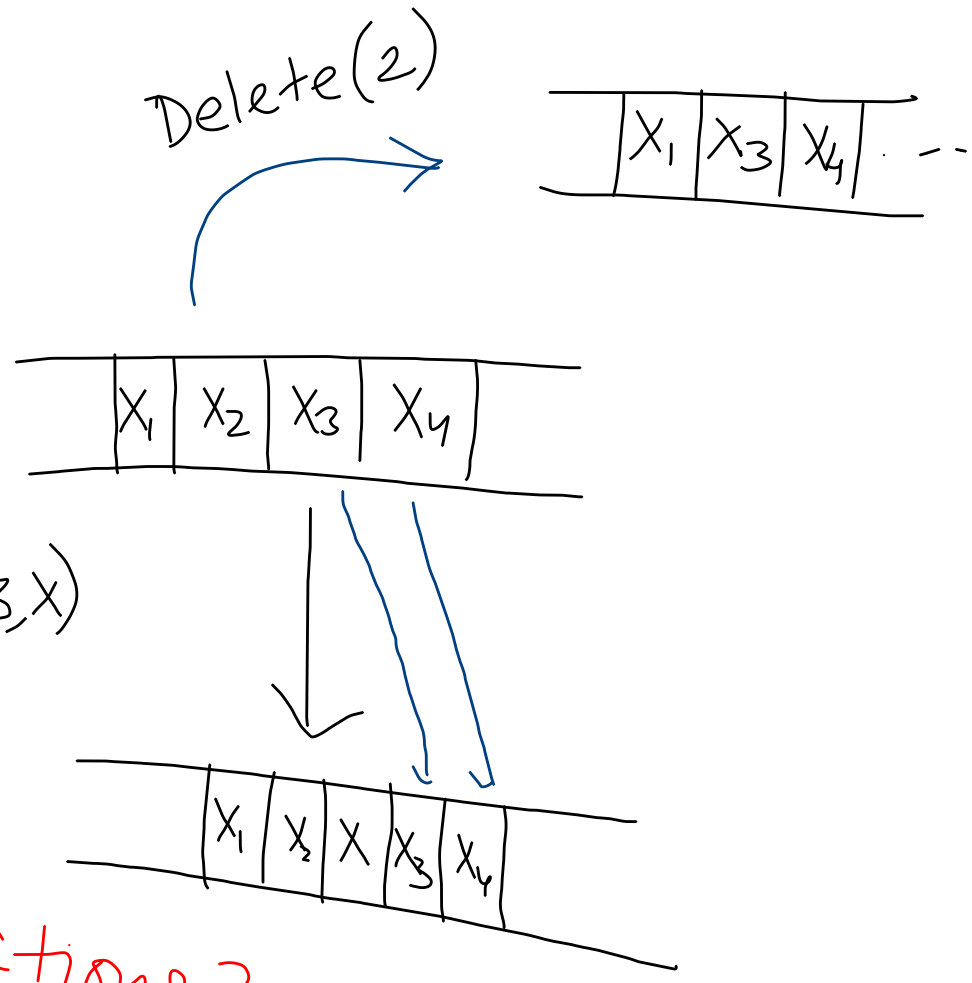
Set_at → $O(1)$

Static : Size of fixed ($n$)

- Static Array

# Dynamic Sequence Interfaces

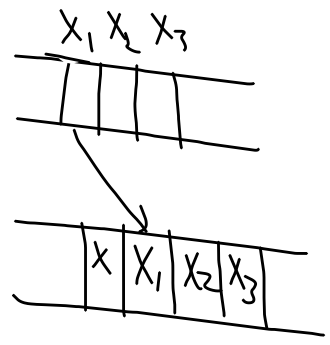
- Insert_at (i, X)
- Delete_at (i)




Does static array supports these functions?

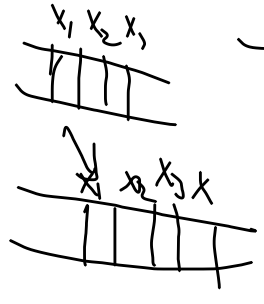
Yes, but not very efficient

Insert_at_begining ⇒




Θ(n) operations (Shifting)

Insert_at_end ⇒

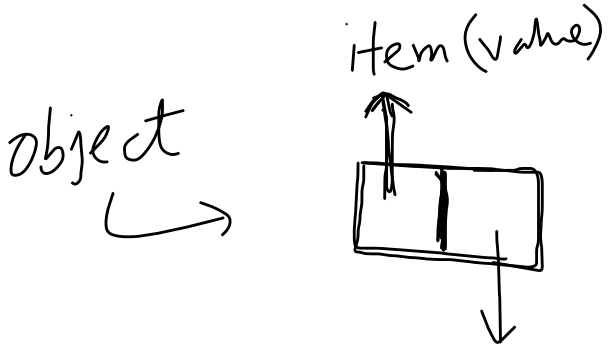



Θ(n)

Copying
Shifting +

# How to implement these dynamic operations efficiently?

## Link-List (Data Structures for dynamic operations)




(Link to the next element in order)

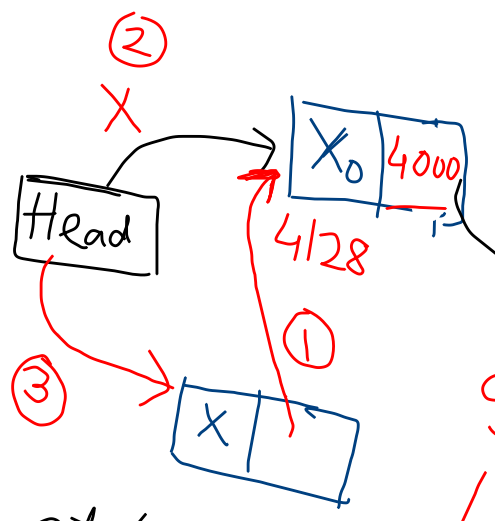
- insert_at (i, x)
- delete_at (i)

insert_at_begin(x)

time: $\Theta(1)$

insert_at(i, x)

time: $\Theta(i)$

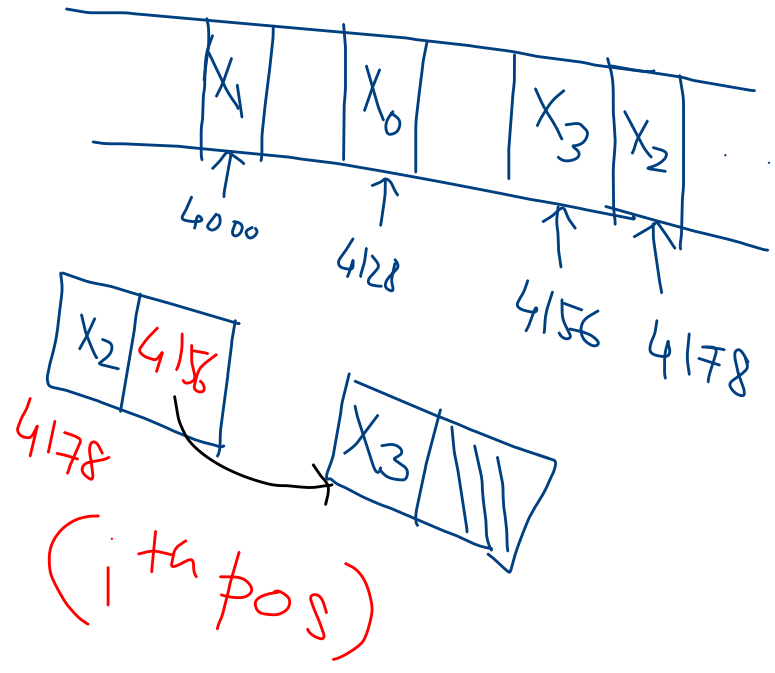



delete_at_begin(x)

$\Theta(1)$

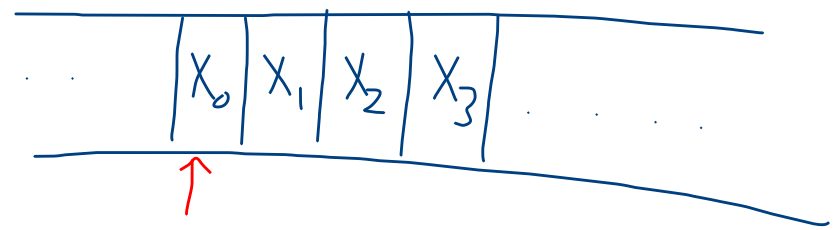
get/set

$\Theta(i)$




(i th pos)

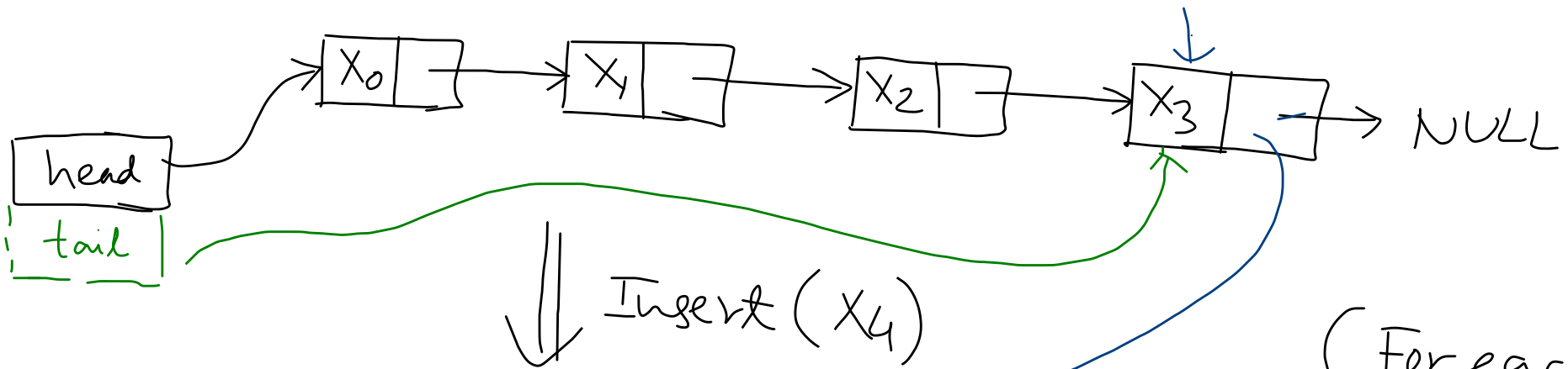
Array



# Insert at end

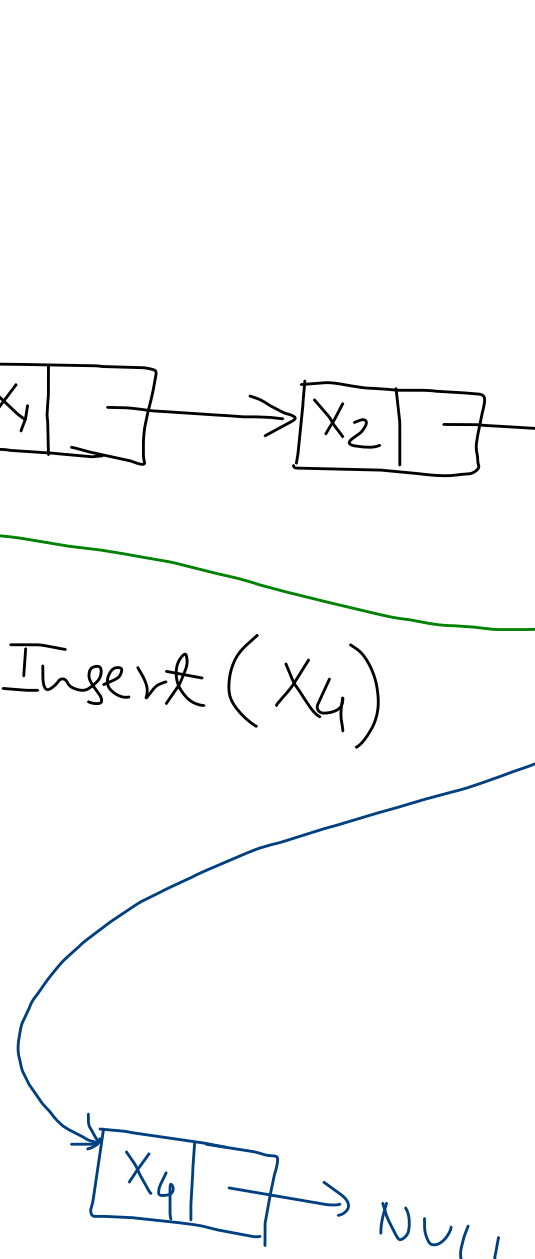
Delete_at_end

↳ $\Theta(n)$




Insert ($X_4$)

Traverse through the link-list & Check whether the current element link points to NULL or not
- if not continue
- else

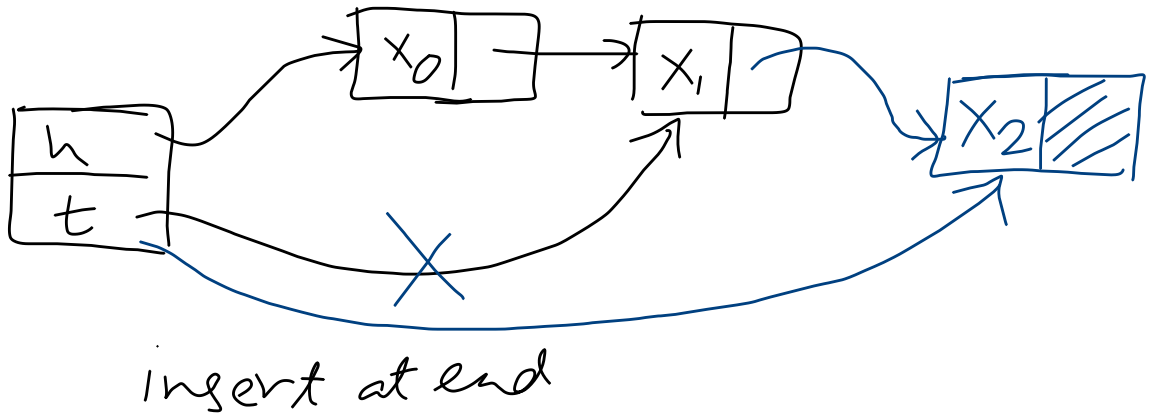



$\Theta(n)$

For each insert operation
↳ Update the tail value

$X_3 \rightarrow next \rightarrow X_4$

tail → $X_4$



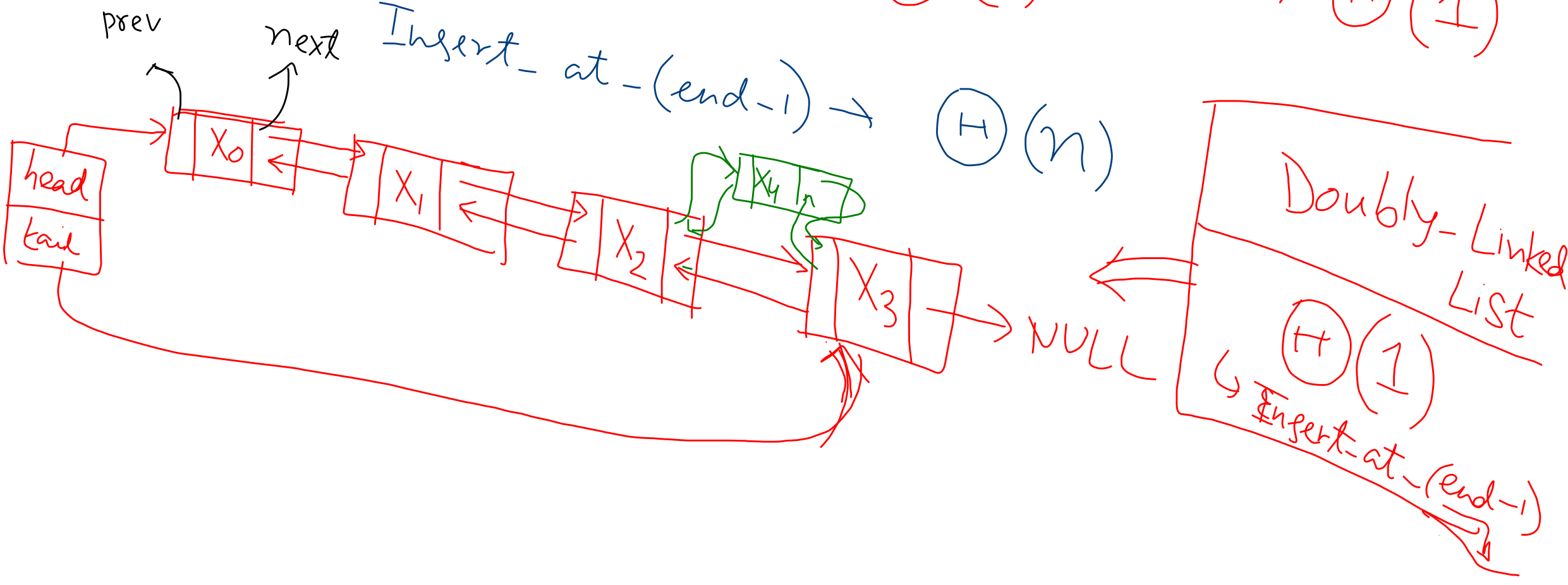

{ Update the tail only when insert at end

Use of tail :

Insert_at_end → Θ(1)

Delete_at_end → Θ(1)

Insert_at_(end-1) → Θ(n)



# Summary

| | Insert_at_begin Delete | Insert at end | Delete at end |
|---|---|---|---|
| Linked-List | $\Theta(1)$ | $\Theta(n)$ | $\Theta(n)$ |
| ✓ Linked-List (+tail) | $\Theta(1)$ | $\Theta(1)$ | $\Theta(n)$ |
| Doubly Link list | $\Theta(1)$ | $\Theta(1)$ | $\Theta(1)$ |

# Dynamic Arrays

$$\begin{array}{r} 0111111 \\ \quad\quad\quad 1 \\ \hline 100\text{---} \end{array}$$

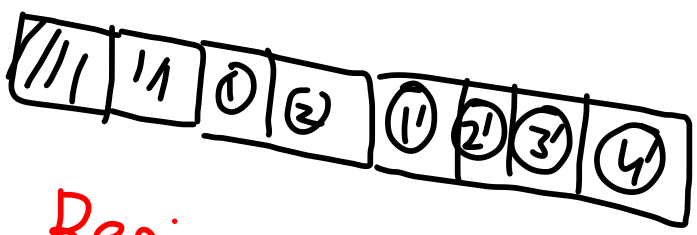
$$1+2+2^2+\cdots+2^k = 2^{k+1}-1$$

Static → At each insertion → $\Theta(n)$ operations.

- relax the constraint (size exactly n)
- if (len == size)

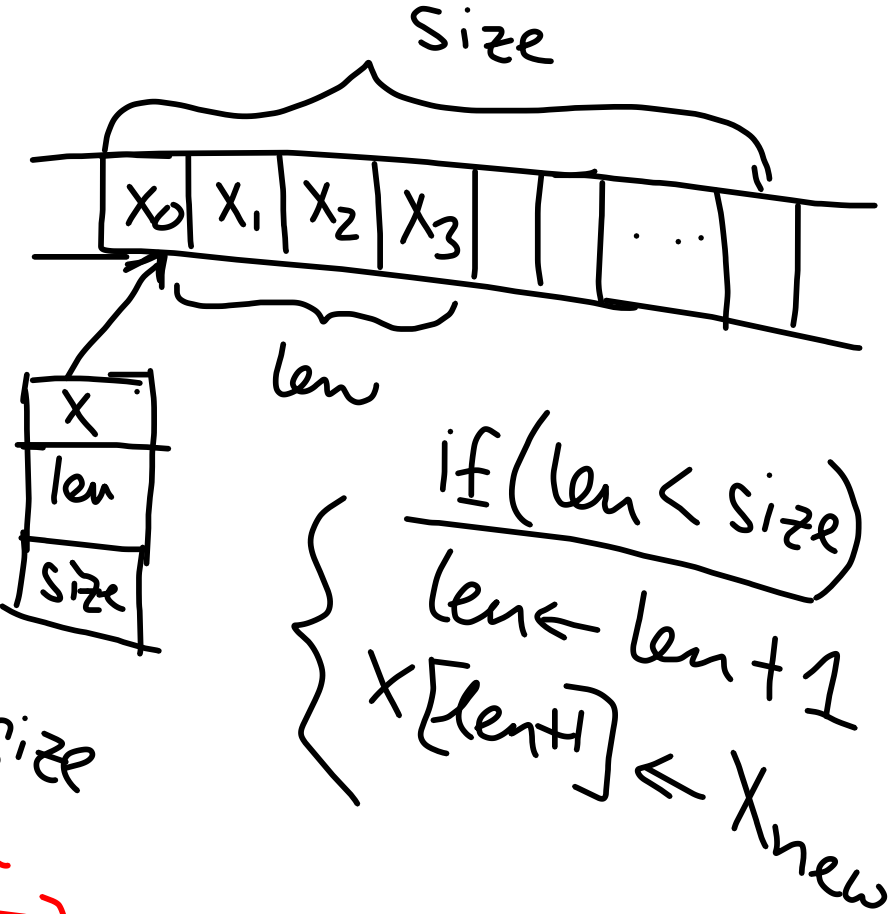
  Size = 2 * Size

  Allocate memory for an array of size $= n$



Resize cost $= \Theta(1+2+2^2+2^3+\cdots+2^{(\lg n)})$

$= \Theta(n)$




if (len < size)

{ len ← len + 1

X[len] ← $X_{new}$

# Amortized Cost

An operation takes $T(n)$ amortized time if any $k$-operations take $\leq k \cdot T(n)$ time.

$n$-operations $\longrightarrow$ Amortized Cost $O(1)$

# Summerize

| DS | Static | Dynamic | | |
|---|---|---|---|---|
| X | get_at(i)<br>set_at(i,x) | insert_first<br>delete_first | insert_last<br>delete_last | insert_at(i,x)<br>delete_at(i) |
| Array | $\Theta(1)$ | $\Theta(n)$ | $\Theta(n)$ | $\Theta(n)$ |
| Linked List | $\Theta(n)$ | $\Theta(1)$ | $\Theta(n)$ | $\Theta(n)$ |
| Dynamic Array | $\Theta(1)$ | $\Theta(n)$ | $\Theta(1)$<br>Amortize | $\Theta(n)$ |