Interface (API/ADT) vs Data Structure

Interface (API/ADT)	Data Structure
- Specification	- representation
- what data can store	- how to store data
- what operations are supported & what they mean	- algorithms to support operations
- problem	- solution

2 main interfaces	2 main DS approaches
- set	- arrays
- sequence	- pointer based (link-list)

$\boxed{\text{Static sequence } \boxed{\text{interface}}}$: maintain a sequence of items $x_0, x_1, \cdots, x_{n-1}$, subject to these operations:

- build(X): make new DS for items in X
- len() : return n
- iter-seq(): Output $x_0, x_1, \cdots x_{n-1}$ in sequence order
- get_at(i): return x_i

Memo No.
Date / /

- set_at(x, i): set x_i to x

Solution: (nature) static array — O(1) per get_at / set_at

key: word RAM model of computation

- memory = array of w-bit words

- "array" = consecutive chunk of w-bit memory — every w store a address of data

$\Rightarrow$ array[i] $\equiv$ memory[address(array) + i]

$\Rightarrow$ array access is O(1) time

$\boxed{\text{Assume } w \geq \lg n}$

$\boxed{\text{Static array}}$

- O(1): per get_at / set_at / len
- O(n): builded / iter_seq

Memory allocation model: allocate (分配) array of size n in O(n) time

$\Rightarrow$ space = O(time)

Mo | Tu | We | Th | Fr | Sa | Su

Memo No.

Date / /

Dynamic sequence interface

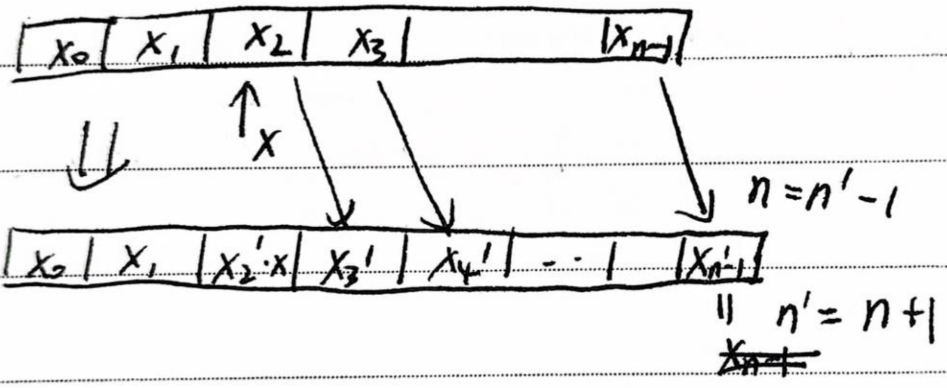
static sequence, plus:

- insert_at (x, i) : make x the new x_i.

  shifting $x_i \to x_{i+1} \to x_{i+2} \to \cdots \to x_{n-1} \to x_{n'-1}$, $n' = n+1$

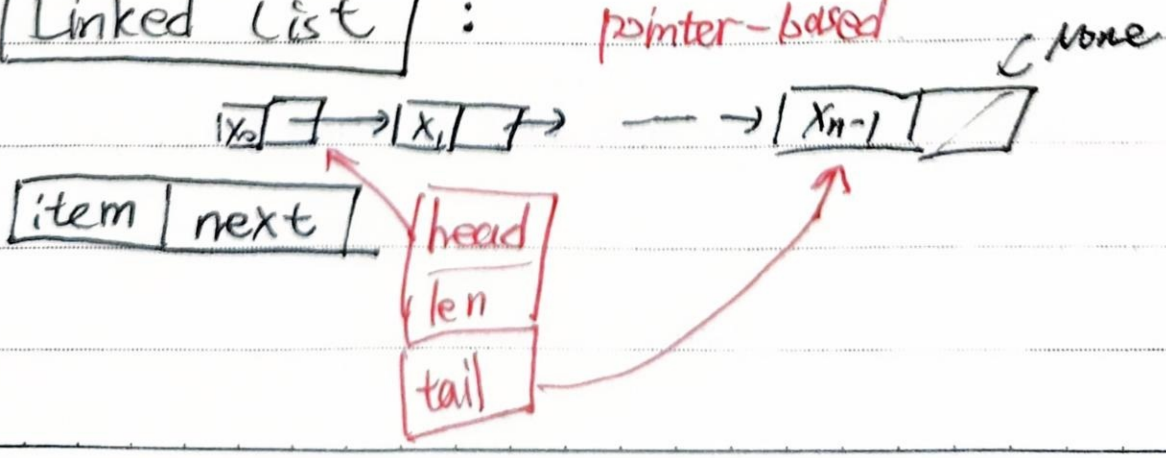
- delete_at(i) : shift $x_i \leftarrow x_{i+1} \leftarrow \cdots \leftarrow x_{n'-1} \leftarrow x_{n-1}$, $n' = n-1$

- get_first/last()   - set_first/last(x)



- insert/delet_first/last(x)/()

Linked List : pointer-based



Dynamic Seq ops

static array linked list

$A[i] = x_i$

insert/delete_at() cost $\Theta(n)$ time

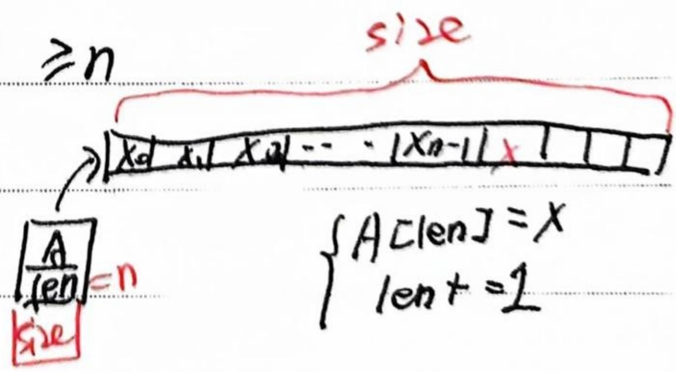
① shifting
OR
② allocation a new array /copying

linked list

- insert/delete_first() : $O(1)$ time
- get/set_at need $O(i)$ time
  ↓
  ($O(n)$ worst case)

Dynamic array (Python lists)

- relax constraint size(array) = n ← # items in sequence
- enforce size = $\Theta(n)$ & $\geq n$
- maintain $A[i] = x_i$
- insert_last(x): add to end unless n = size



$\begin{cases} A[len] = x \\ len += 1 \end{cases}$

- if n = size : allocate new array of size bigger, (2·size) or size + 5 ← bad

Memo No. ____
Date / /

- n insert_last() from empty array

$\boxed{x}\,\boxed{x}$ $n = 1, 2, 3, \ldots$

resize at $n = 1, 2, 4, 8, 16, \cdot$ (time 2×size)

$\Rightarrow$ resize cost $= \Theta(1+2+4+8+16+\cdots)$

$= \Theta\left(\sum_{i=1}^{\lg n} 2^i\right) = \Theta(2^{\log n}) = \Theta(n)$

(均~~[illegible]~~ 摊)

Amortization:

operation takes $T(n)$ amortized time

if any k operations take $\le k \cdot T(n)$ time

(averaging over the operation sequence)

it means $\Theta(n)$ in n operation, every operation takes (a)mortized $O(1)$ time

	Static		Dynamic					
	get_at(i)	set_at(i,x)	insert_first(x)	delete_first()	insert_last(x)	delete_last()	insert_at(i,x)	delete_at(i)
Array	1		n		n		n	
LinkL	n		1		n		n	
Dyna array	1		n		1(a)		n	

averaging