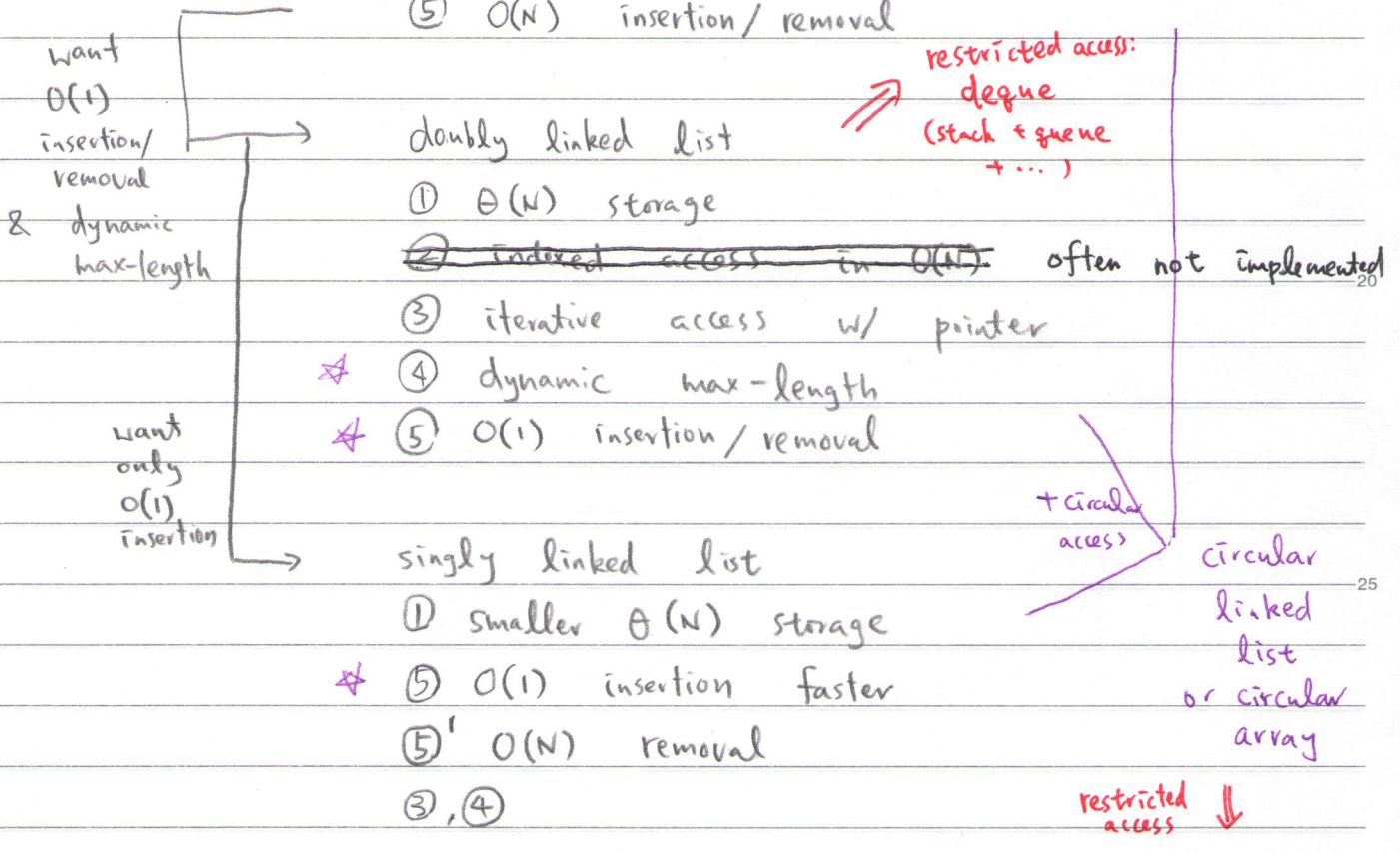
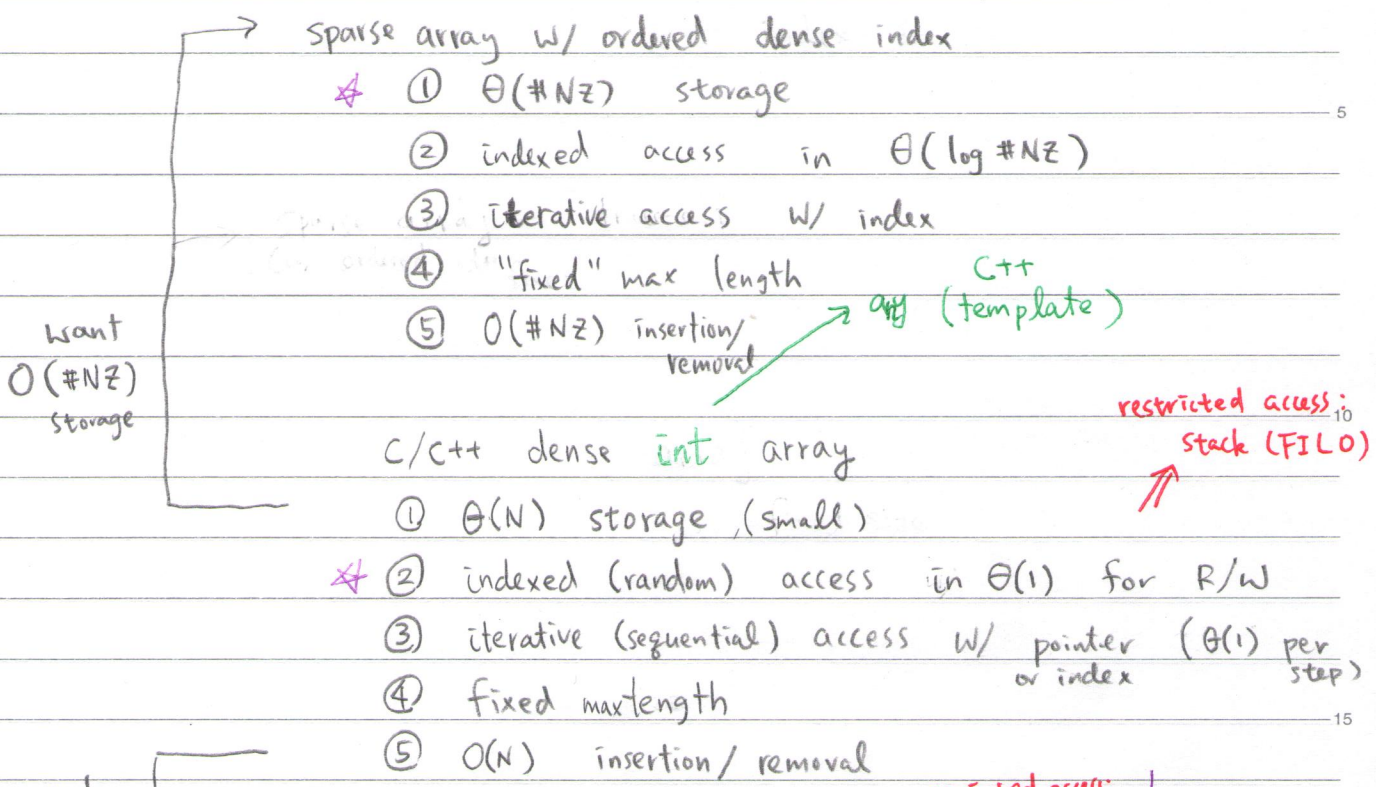


\* "containers" we encountered



mixed : 2-D array (e.g. array of array)

other sparse array (e.g. w/ linked list)

actual C++ deque (linked list of arrays)

\* abstraction: get the "essence" of what we need

① save implementation efforts (e.g. type abstraction by template)

② "easy" change of implementation

singly or doubly? "same functionality," different

underlying implementation<sup>5</sup>

\* functionality abstraction (contract)

dense array & sparse array & extendable (dense) array

**vector**: indexed (random) access

at(i) (random access, R)

set(i, e) (random access, W)

insert(i, e) (insertion)

erase(i) (removal)

\* extendable array

if <sup>internal</sup> array A overflows  
grow the array

allocate new array B  $O(1)$   
copy contents to the new array B  $O(n)$  <sup>#elem</sup>  
remove A  $O(1)$   
and assign B to A

consider M "pushes" to the array

① size(B) = size(A) + 1

② size(B) = size(A) \* 2

size(A)	size(B)
1	2
2	3
3	4
4	5
⋮	⋮
M	M+1

# allocate =  $O(M)$   
# copy =  $\frac{(M-1)M}{2} = O(M^2)$

size(A)	size(B)
1	2
2	4
4	8
8	16
⋮	⋮
$2^k$	$2^{k+1}$

# allocate =  $k+1 = O(\log M)$   
# copy =  $1+2+\dots+2^k = 2^{k+1}-1 = O(M)$