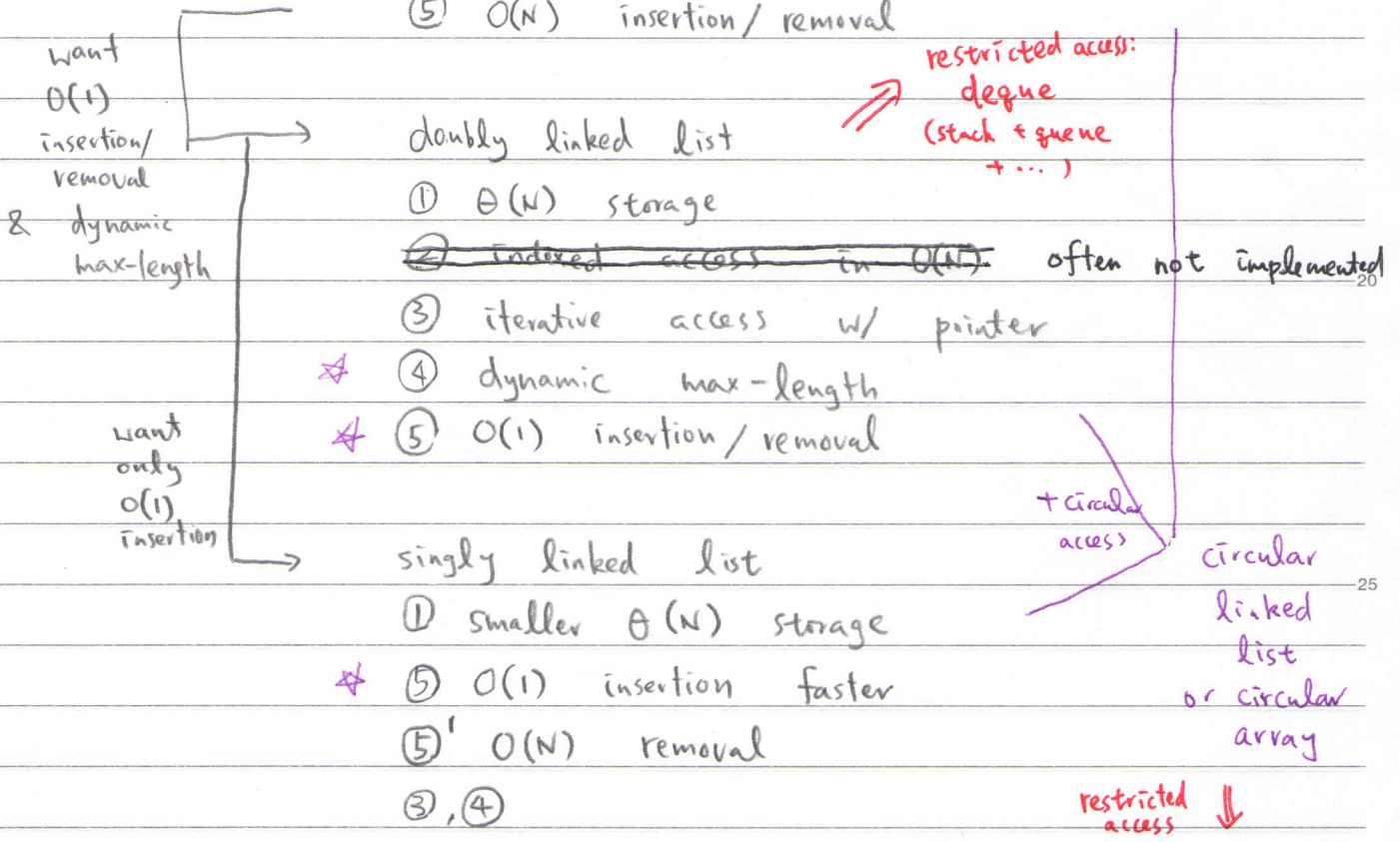
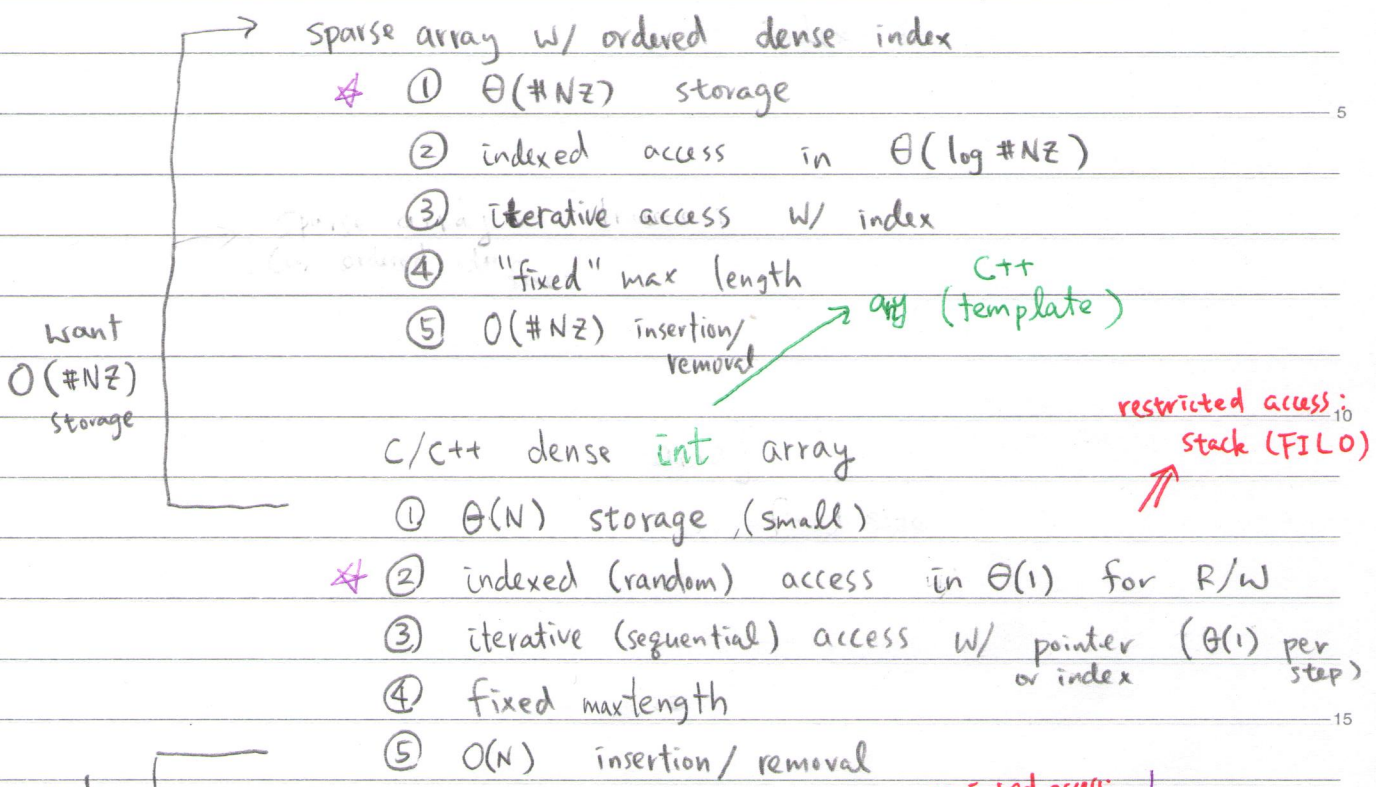


* "containers" we encountered



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 implementation⁵

* 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 ^{internal} array A overflows
grow the array

allocate new array B $O(1)$
copy contents to the new array B $O(n)$ ^{#elem}
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(A)
1	1
2	2
3	3
4	4
...	...
M	copy(M-1)

size(A)	size(A)
1	1
2	2
3	4
4	4
5	8
6	

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

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

② better than ①, implemented in `std::vector`
(caveats?)

w/ "reserve" functionality

* functionality abstraction

doubly & singly linked list & array

list : iterative ^(positional) access

`insert(p, e)` insert at position
`elem(p)` element at position
`begin()` starting position

`p != end()` \Leftarrow `isEnd(p)` is p the end?
`nextOf(p)` go to next element of p
`erase(p)`

^p iterator: abstraction for

- index in sparse array
- index/pointer in dense array
- pointer in linked list

	sparse array	dense array	list
begin	0	&arr[0]	head
end	n+1	&arr[n+1]	NULL
nextOf(p)	p+1	p++	p → next
elem(p)	at(p)	(*p)	p → value

- `iterator < container-type >`, "safe" pointer in some sense
- overload "++" to do `nextOf`, override "*" to do `elem`
- `int sum = 0;`

```
for ( iterator < list<int> > p = c.begin(); p != c.end(); p++ ) {
```

can now "freely" change this one

```
sum += (*p);
```

```
}
```

STL list : doubly linked list

* sequence : vector + list + i \Leftrightarrow p