abstraction "essense"

1. save implementation efforts (template: type abstraction)
2. "easy" change of data structure

* functionality abstraction

dense array & sparse array (fast, storage-saving)

indexed (random) access : vector

extendable array → C++ vector

- at(i) . getatIndex(i)
- set(i) . putatIndex(i)
- insert .
- erase .

* exendable array

if array A "overflow"

grow the array

1. allocate new array B O(1)
2. copy contents from A to B O(N)
3. remove A and assign B to A O(1)

consider M "pushes" to array

a. size(B) = size(A) + 1

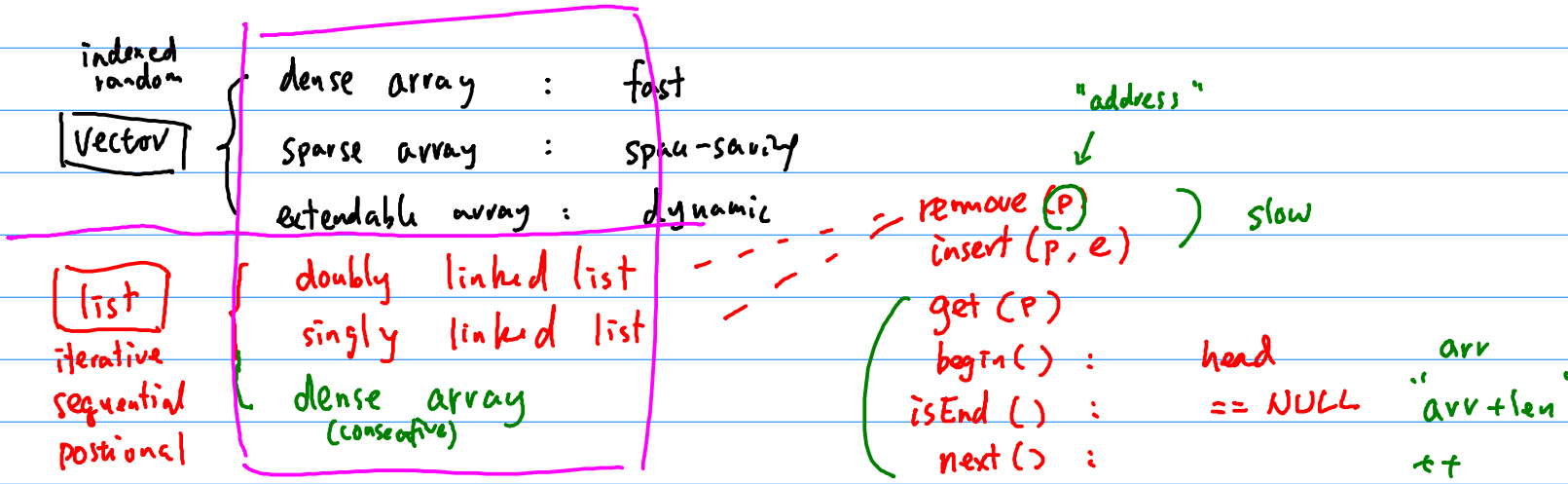
N		size(A)	
1	↓ !, allocate, copy(1)	1	on average $\frac{O(M^2)}{M} = O(M)$
2		2	
3	↓ !, allocate, copy(2)	3	
4	↓ !, allocate, copy(3)	4	
⋮		⋮	
M	↓ !, allocate, copy(M-1)		

M-1 allocate ⇓ $1+2+3+\cdots+(M-1)$ copy ⇓ $\Theta(M^2)$

b. size(B) = size(A) * 2 C++ vector

N		size(A)	
1	↓ !, allocate, copy(1)	1	on average $\frac{O(M)}{M} = O(1)$
2	↓ !, allocate, copy(2)	2	
3	↓ !, allocate, copy(3)	4	
4	↓	4	
5	↓ !, allocate, copy(4)	8	
6	↓	8	
7	↓	8	
8	↓	8	
9	↓ !, allocate, copy(8)	16	
⋮		⋮	
2^K		$\sim 2^K$	
M			

K allocate $O(\log M)$ ⇓ $1+2+4+8+\cdots+2^{K-1}$ copy $O(M)$



	dense array	linked list
begin	&arr[0]	head
end	&arr[N]	NULL
next	p++	p->next

p.next() ===> p++

```
for(iterator<vector<int> > p = c.begin(); p != c.end(); p = p.next()){
  sum += elem(p);
}
```

elem(p) ===> (*p)