* "containers" we encountered

Want O(#NZ) storage

- Sparse array w/ ordered dense index
  - ★ ① Θ(#NZ) storage
  - ② indexed access in $\Theta(\log \#NZ)$
  - ③ iterative access w/ index
  - ④ "fixed" max length
  - ⑤ O(#NZ) insertion/removal

C++ (template)

- C/C++ dense int array
  - ① $\Theta(N)$ storage (small)
  - ★ ② indexed (random) access in $\Theta(1)$ for R/W
  - ③ iterative (sequential) access w/ pointer or index ($\Theta(1)$ per step)
  - ④ fixed max-length
  - ⑤ O(N) insertion/removal

restricted access: stack (FILO)

Want O(1) insertion/removal & dynamic max-length

- doubly linked list
  - ① $\Theta(N)$ storage
  - ~~② indexed access in $\Theta(1)$~~ often not implemented
  - ③ iterative access w/ pointer
  - ★ ④ dynamic max-length
  - ★ ⑤ O(1) insertion/removal

restricted access: deque (stack + queue + ...)

Want only O(1) insertion

- singly linked list
  - ① smaller $\Theta(N)$ storage
  - ★ ⑤ O(1) insertion faster
  - ⑤' O(N) removal
  - ③, ④

+ circular access

circular linked list or circular array

restricted access

queue (FIFO)

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

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

actual C++ deque (linked list of arrays)

* abtraction: get the "essence" of what we need
  ① save implemention efforts (e.g. type abstration by template)
  ② "easy" change of implemention
  singly or doubly? "same functionality," different underly imp

* 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

n	size(A)
1	1
2	2
3	3
4	4
⋮	
M	copy(M-1)

! , allocate, copy(1), ! , allocate, copy(2), ! , allocate, copy(3)

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

② size(B) = size(A) * 2

n	size(A)
1	1
2	2
3	4
4	4
5	8
6	
⋮	
2^K	
$M=2^K+1$	

! , allocate, copy(1) ! , allocate, copy(2) ! , allocate, copy(4) ! , allocate, copy(2^K)

allocate = K+1 = O(log M)

copy = $1+2+\cdots+2^K$ = $2^{K+1}-1$ = O(M)