* priority queue

want A.insert(key, value)
A.retrieveMax()

fast

* what if

want A.insert(key, value)
A.retrieve(key)

fast ?

map $key_1 \rightarrow value_1$
$key_2 \rightarrow value_2$
⋮
$key_K \rightarrow value_K$

unordered / ordered

dictionary $key_1 \rightarrow value_{1.1}$, $value_{1.2}$, ⋮

		search insert	retrieve
* sparse array (or dense) linked list	w/ key	O(1)	O(n)
	w/ ordered key	O(n)	O(log n)
	w/ key	O(1)	O(n)
	w/ ordered key	O(n)	O(n)

* if keys are integers
O(1) insert O(1) retrieve
by dense array (that wastes space)

* how about using a "simple" (almost O(1)) function
h: key → integer

A.insert(key, value) ⇒ A[h(key)] = value

v = A.retrieve(key) ⇒ v = A[h(key)]

bucket array — hash function

* if {keys} ⟺ (one-to-one) {0, 1, 2, ⋯, k-1}

"perfect hashing"

* what if not ?

e.g. # keys > k

⇒ two keys of the same h(·)

⇒ collision !

* e.g. h(str) = str[0] - 'a' ; 26 possibilities

h("act") = h("apple")

$h(str) = \sum_i (str[i] - 'a')$ 2^{32} ? possibilities not uniform

$h(str) = \sum_i (str[i] - 'a') \% k$ k possibilities

"stop" "pots" "spot"

⇒ easy collision

* how to hash

h(key) = compress (Hashcode(key))

- Hashcode(key):
  - sum [not using position info]
  - polynomial (like shift-left)
  - cyclic shift
- compress:
  - division (% k)
  - MAD ((a·□+b) % k)