Basic Implementation of Dictionary/Map

We need some kind of collection to hold the keys/elements in the map.

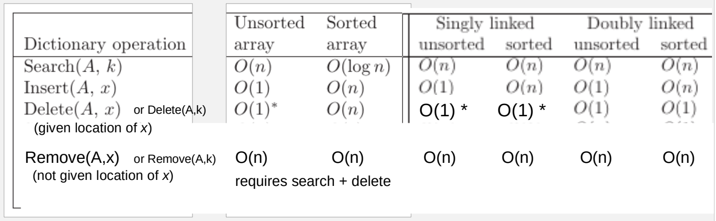
There are two basic collections

- array
- linked list

and two basic ways elements can be ordered within those collections

- not sorted
- sorted

Basic Implementation of Map/Dictionary



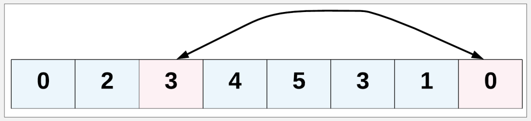
	Unsorted	Sorted	Singly linked		Doubly linked	
Dictionary operation	array	array	unsorted	sorted	unsorted	sorted
Search(A, k)	$O(n)$	$O(\log n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Insert(A, x)	$O(1)$	$O(n)$	$O(1)$	$O(n)$	$O(1)$	$O(n)$
Delete(A, x) or Delete(A,k) (given location of *x*)	$O(1)^*$	$O(n)$	O(1) *	O(1) *	$O(1)$	$O(1)$
Remove(A,x) or Remove(A,k) (not given location of *x*)	O(n) requires search + delete	O(n)	O(n)	O(n)	O(n)	O(n)

A is the dictionary, *k* is a key, *x* is a key-value pair (*k*,*v*)

delete operation as defined in ADM assumes that the element is already found (known array index, pointer to the linked list node) – otherwise find operation is required first

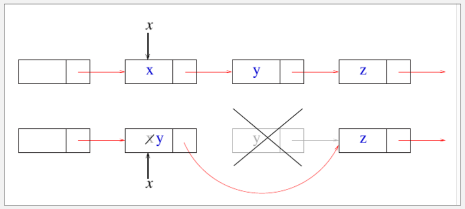
* denotes cleverness or subtlety

Constant-Time Deletion in an Unsorted Array



- O(1) deletion
  - swap element to be deleted with the last element, then remove the (new) last element

Constant-Time Deletion in a Singly-Linked List



- O(1) deletion

```
x.setValue(x.getNext().getValue())
x.setNext(x.getNext().getNext())
```

Basic Implementation of PriorityQueue

We need some kind of collection to hold the keys/elements in the PQ.

There are two basic collections

- array
- linked list

and two basic ways elements can be ordered within those collections

- not sorted
- sorted

Basic Implementation of PriorityQueue

operation	array - unsorted	array - sorted	linked list - unsorted	linked list - sorted
find min				
insert				
remove min				

Basic Implementation of PriorityQueue

operation	array - unsorted	array - sorted	linked list - unsorted	linked list - sorted
find min	**O(n)** – search	**O(1)** – in slot 0	**O(n)** – search	**O(1)** – at head
insert	**O(1)** – add at end	**O(n)** – binary search + shift	**O(1)** – add at head	**O(n)** – sequential search
remove min	**O(n)** – search + delete (swap)	**O(n)** – shift	**O(n)** – search + delete	**O(1)** – at head

Can we avoid (some) searching and shifting?

– store min location (update on insert, remove)

– circular array or reverse sorted array

Basic Implementation of PriorityQueue

operation	array - unsorted	array – reverse sorted	linked list - unsorted	linked list - sorted
find min	**O(1)** – store index of min	**O(1)** – in last slot	**O(1)** – store node with min	**O(1)** – at head
insert	**O(1)** – add at end	**O(n)** – binary search + shift	**O(1)** – add at head	**O(n)** – sequential search
remove min	**O(n)** – delete (swap) + update min index	**O(1)** – in last slot	**O(n)** – update min node	**O(1)** – at head

Tradeoff: fast insert or fast remove, but not both.

Can we do better?