Basic Implementation of Containers

Vector / List / Sequence	classic array vs linked list tradeoffs • insert/remove not at the end requires shifting in the array (O(n)), but access by rank (index) is O(r) for linked list • dynamic array has overhead in time (resizing) and space (empty slots), linked list has overhead in space (pointers)
Stack	O(1) push, pop with array and linked list • top of stack = end of array, head of linked list choice of array vs linked list is largely determined by whether there is an upper bound on the size of the stack that is known in advance (static array) or not (dynamic array or linked list – time vs space overhead tradeoff)
Queue	O(1) enqueue or dequeue, O(n) for other with array, linked list

Can we do better?

- Vector/List/Sequence – tradeoff is due to the nature of the data structures (random access vs sequential access)
- Stack – can't beat O(1)
- Queue – …

Improving an Implementation – Queue

Consider the array implementation with the head of the queue at the beginning of the array.

- enqueue(x) is O(1) – insert at the end of the array
- dequeue() is O(n) – removing from head of array requires shifting

Do we have to shift?

- we shift to keep the head of the queue at 0 and the tail position based on the size

Can we store the head and tail positions instead?

- O(1) to locate head/tail
- O(1) to update head/tail – new value is next position
  - "next position" at the end of the array wraps around to 0

→ O(1) enqueue and dequeue using a circular array

Improving an Implementation – Queue

Consider the linked list implementation with the head of the queue at the beginning of the list.

- enqueue(x) is O(n) – inserting at the end of the list requires finding the last node
- dequeue() is O(1) – removing from head just involves updating pointers

enqueue is slow because we have to find the tail of the list.

Can we store a tail pointer instead?

- enqueue – O(1) to locate the node before the insertion point (current tail), O(1) to create new node and link to current tail, O(1) to update current tail to new node
- dequeue is not affected (unless the last element is removed – tail becomes `null`)

→ O(1) enqueue and dequeue using a linked list with a tail pointer

Doing Better

- if the slowness is because of having to find or compute something, can you store it instead?
  - must consider the cost of updating the stored info
  –
- if the slowness is the result of not storing something, can you store it instead?
  - must consider the cost of updating the additional info stored