Array-Based Implementations

Observations –

- things arrays are good for – Θ(1)
  - accessing a particular slot (random access)
  - inserting or removing elements at the end
  - inserting or removing elements in the middle when the order doesn't need to be preserved (can swap with the last thing)

doesn't involve a loop – same number of steps regardless of the size of the array

- things arrays are less good for – Θ(n)
  - inserting or removing elements in the middle when the order needs to be preserved
  - varying-size collections when you have to grow or shrink
    - doubling the size mitigates the expense of copy over a series of insertions

involve a loop – number of steps depends on the size of the array

Linked List-Based Implementations

Observations –

- things linked lists are good for – Θ(1)
  - accessing the head
  - inserting or removing elements at the head
    - inserting at the tail with a tail pointer
    - removing the tail if doubly-linked
  - inserting or removing after a node
    - inserting or removing before a node if doubly-linked

doesn't involve a loop – same number of steps regardless of the length of the list

- things linked lists are less good for – Θ(n)
  - accessing a particular position (no random access)
  - inserting or removing at a particular position
  - inserting or removing before a node (if singly-linked)

involve a loop – number of steps depends on the length of the list

Arrays vs. Linked Lists

Advantages of linked lists –

- no need to grow when full because nodes are allocated/deallocated as needed
- no empty slots
  - though arrays still have an advantage in space usage as long as they are at least half full
- insert/remove don't require shifting
  - much faster than array if insertion point is known (otherwise requires time to find node)

Advantages of arrays –

- random access
  - linked lists support sequential access only – must scan forward from head
- simpler if the number of elements doesn't change