LINKED LISTS

RECALL: ARRAYS

- Single chunk of memory underneath
- Access with `arrname[i]`
- Good for:
  - direct access/modification of elements at certain index
- Bad at:
  - inserting/deleting: need to shift and potentially get new array

ARRAYLIST

- A higher level list implementation - underneath uses arrays
- Convenient
- Efficient for:
    - accessing / setting value at index - O(1)
    - adding elements to the end - typically O(1), sometimes O(n)
- Not efficient for:
    - adding in middle - O(n)
    - removing - O(n)

LINKED LIST

- Data structure
- A list
    - Supports things like get, add, insert, remove, etc.
- Different underlying implementation
    - collection of nodes that are "linked" together
    - forms a sequence of elements

LINKED LIST - NODE

- Object
- Single value in list
- Stores
    - element
    - reference to next node (self-referential)

LINKED LIST

- Sometimes a separate class from node
- `head` - reference to first `Node` in list
  - adding to start is O(1)
- `tail` (optional) - reference to last node in list
  - makes adding to / removing from end O(1)

LINKED LIST (CONT.)

- downside:
  - access is inefficient (must traverse)
  - extra memory (store pointer to next for each node)

DOUBLY LINKED LIST

- Modification
    - `Node` has additional reference to previous
- Advantages
    - `remove(Node n1)` - more efficient (no need to traverse)
    - fast removal from end