Implementing the List ADT

- We want to support the operations:
  - **INSERT** and **DELETE**
  - **FIND** and **FINDKTH**
  - **NEXT** and **PREV**
  - **ISEMPTY**

- We could use an ordinary C array to implement these operations. For a list of length N:
  - **FIND** - linear search of the array, requires O(N) time
  - **INSERT, DELETE** - O(N) because we need to move everything after the insert/delete position. This requires O(N), unless there’s a magic way to "open" or "close" a hole in the array. (With C arrays, this is not the case)

  Note that if we always insert/delete from the end, it takes constant time: O(1).

  - **FINDKTH** - O(1) - **THIS IS EASY USING C ARRAYS!**
  - **NEXT, PREV, ISEMPTY**: all O(1), but require some extra information. **NEXT** and **PREV** require keeping track of the index of the “current” position, and **ISEMPTY** requires keeping a flag to indicate whether there is anything in the list at all.
Drawbacks of Using Arrays to Implement Lists

• Arrays are fixed size (or clumsy to change size). Unless you have a good idea of what N is going to be:
  • You might guess too small!
  • You might guess too large, and waste space.
• The potential slowness of INSERT and DELETE
Another Approach - "Linked Lists"

- Instead of a structure where the location of the next and previous elements are statically defined:

  ![Diagram of an array with prev and next pointers](image)

- Use a structure where the location of the neighbors is dynamic - each "cell" of the list contains info about its neighbors, and this info can be changed.

- Pointers make this easy, although this can be done without them (see Weiss, 3.2.8)

- What are the Big-O's for the linked-list operations?

  - **INSERT** and **DELETE**
  - **FIND** and **FINDKTH**
  - **NEXT** and **PREV**
  - **ISEMPTY**

  It depends on how the linked list is implemented.
List ADT - Implementation Data Structures

<table>
<thead>
<tr>
<th></th>
<th>Array</th>
<th>Singly-Linked List</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSERT</td>
<td>O(n)</td>
<td>O(1)</td>
</tr>
<tr>
<td>DELETE</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>FINDKTH</td>
<td>O(1)</td>
<td>O(n)</td>
</tr>
</tbody>
</table>

Q: Why is DELETE O(n) for a singly-linked list?
A: Because searching for the previous element is O(n).

Q: Why is FINDKTH O(n) for a singly-linked list?
A: Because we have to start at the front and work our way down the list. It takes a fixed amount of time to move from one element to the next, and k is O(n), so FINDKTH = O(n)