→ Asymptotic performance of mutating behaviors of the list class

Operation	Run Time
data[j] = val	O(1)
data.append(val)	O(1)*
data.insert(k, val)	O(n-k+1)*
data.pop()	O(1)
data.remove(val)	O(n)*
data1 *= data2	O(n²)
data.reverse()	O(n)
data.sort()	O(nlogn)

Adding Elements to a List

→ two complicating factors for efficiency:

1.) the addition of an element may require us to resize dynamic array

2.) shifting elements to make room

```
def insert(self, k, val):
  if self._n == self._capacity:          // not enough room
    self._resize(2 * self._capacity)     // double capacity
  for j in range(self._n, k, -1):        // shift right first
    self._A[j] = self._A[j-1]
  self._A[k] = val                       // store newest element
  self._n += 1
```

Removing: there is no improvement to removing by value, always $\Omega(n)$

```
def remove(self, val):
  for k in range(self._n):
    if self._A[k] == val:                // found match
      for j in range(k, self._n-1):      // shift others to fill gap
        self._A[j] = self._A[j+1]
      self._A[self._n-1] = None          // help garbage collection
      self._n -= 1                       // one less item
      return
  raise ValueError
```