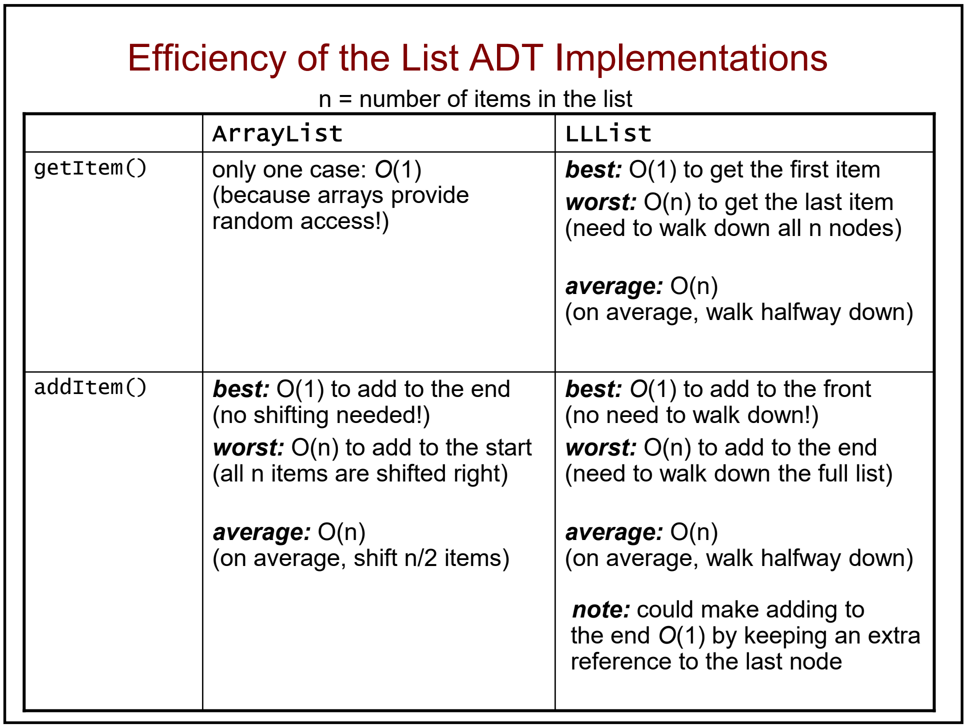
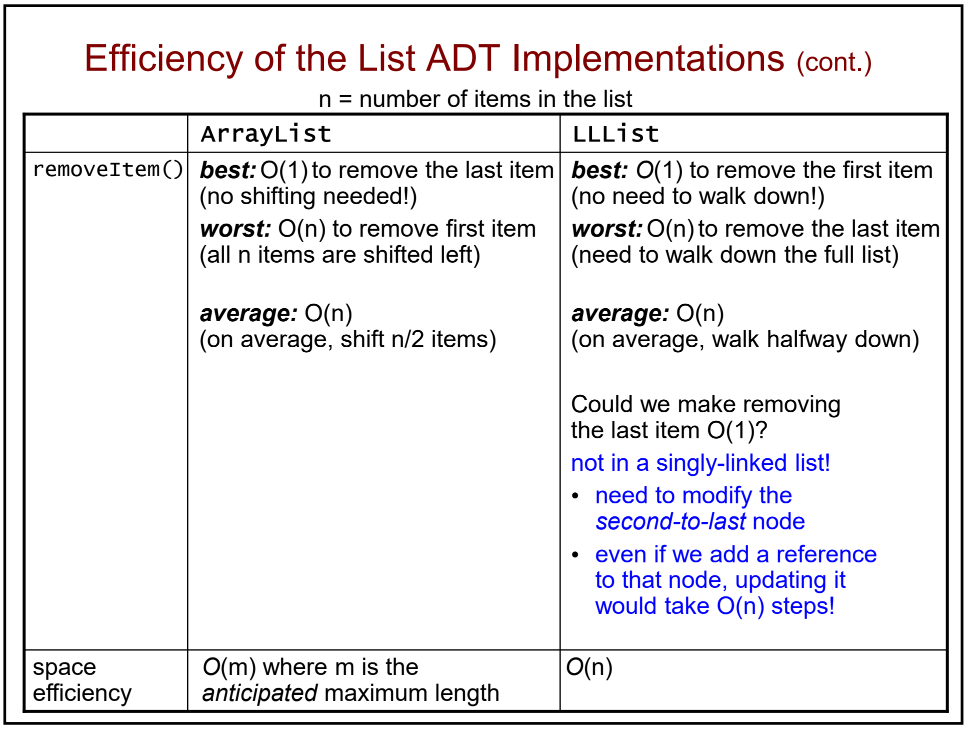


Efficiency of the List ADT Implementations

n = number of items in the list

	`ArrayList`	`LLList`
`getItem()`	only one case: *O*(1) (because arrays provide random access!)	***best:*** O(1) to get the first item ***worst:*** O(n) to get the last item (need to walk down all n nodes) ***average:*** O(n) (on average, walk halfway down)
`addItem()`	***best:*** O(1) to add to the end (no shifting needed!) ***worst:*** O(n) to add to the start (all n items are shifted right) ***average:*** O(n) (on average, shift n/2 items)	***best:*** *O*(1) to add to the front (no need to walk down!) ***worst:*** O(n) to add to the end (need to walk down the full list) ***average:*** O(n) (on average, walk halfway down) ***note:*** could make adding to the end *O*(1) by keeping an extra reference to the last node



Efficiency of the List ADT Implementations (cont.)

n = number of items in the list

	`ArrayList`	`LLList`
`removeItem()`	***best:*** O(1) to remove the last item (no shifting needed!) ***worst:*** O(n) to remove first item (all n items are shifted left) ***average:*** O(n) (on average, shift n/2 items)	***best:*** *O*(1) to remove the first item (no need to walk down!) ***worst:*** O(n) to remove the last item (need to walk down the full list) ***average:*** O(n) (on average, walk halfway down) Could we make removing the last item O(1)? not in a singly-linked list! • need to modify the *second-to-last* node • even if we add a reference to that node, updating it would take O(n) steps!
space efficiency	*O*(m) where m is the *anticipated* maximum length	*O*(n)