Wrap-up of bigoh

Worst case / best case / avg case

How do we handle code w/ "if" statement.

- Linear Search Alg

Bigoh

- What is the best case scenario? → $O(1)$
  - item is at index [0]
- Worst case? - item is at the end of the list → $O(n)$
- Avg case? → $O(\frac{n}{2}) \rightarrow O(n)$

Ω Θ

Primarily concerned w/ worst/avg case.

"Hidden coefficients"

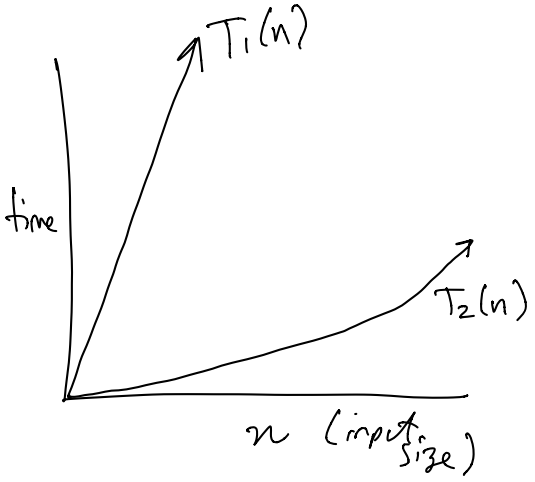
Allowed to drop coefficients in bigoh

$$T(n) = 3n^2 + 5n = O(n^2)$$

2 diff algs → solve the same problem

$T_1(n) = 2{,}000{,}000n \rightarrow O(n)$

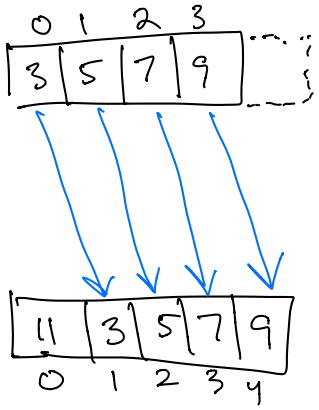
$T_2(n) = 0.0000001n^2 \rightarrow O(n^2)$

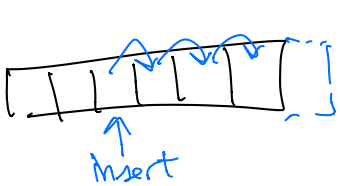


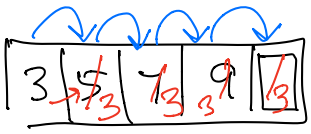
Big-oh times for RArrayList class

	WORST
size	O(1)
get	O(1)
set	O(1)
append	O(n)
prepend	O(n)

prepend







Linked Lists

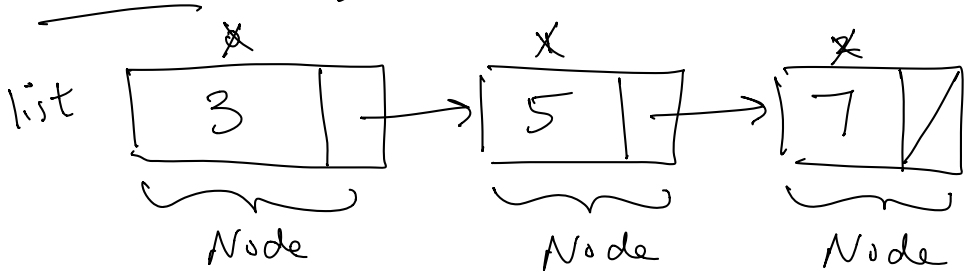
Operations
- inserting an item @ beginning/middle/end
- remove " " " " " "
- search for a specific item
- iterate through a LL
- get size

Big idea — ArrayLists vs LinkedLists
Yes, we have random access | No, we do not

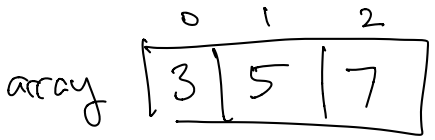
Random access — means we can retrieve/change any item in constant time

Review

Structure called a "Node"



Singly-linked list



print(array[2])

In a LL, we can't "skip" to any item in the list anytime we want to.

Different types of LL

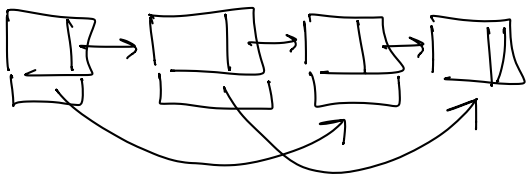
Distinguished by a number properties

① # of pointers within each node.

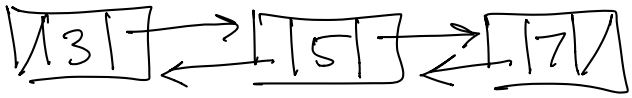
SLL

singly linked list

Skip list

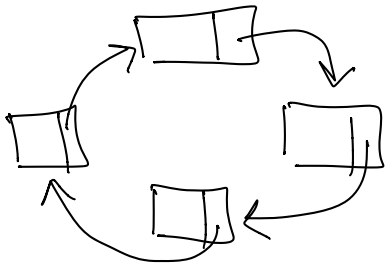


doubly linked list (DLL)

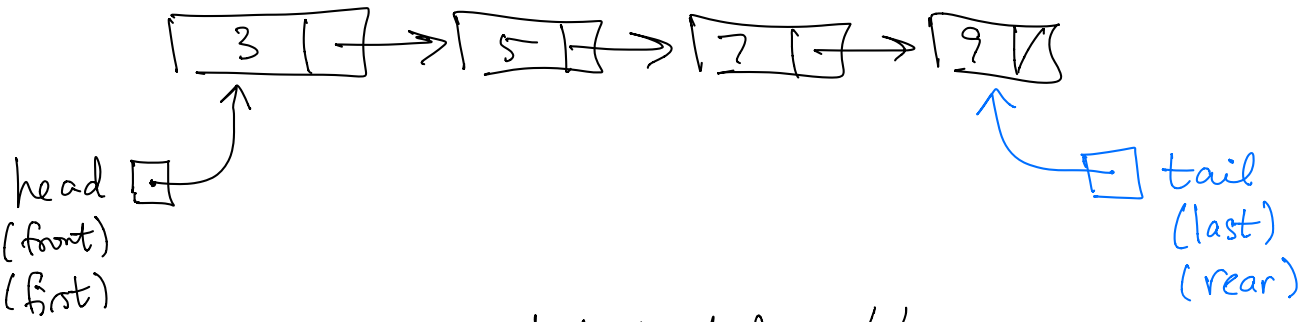


② Shape in which the nodes are connected.

Circularly-linked list



③ How many "special" pointers/references we keep around to access the list.



Func: print the last element in a LL.

What is the big-oh? O(n) – head pointer only

O(1) – head & tail pointers

Code for LL – DRAW A LOT OF PICTURES

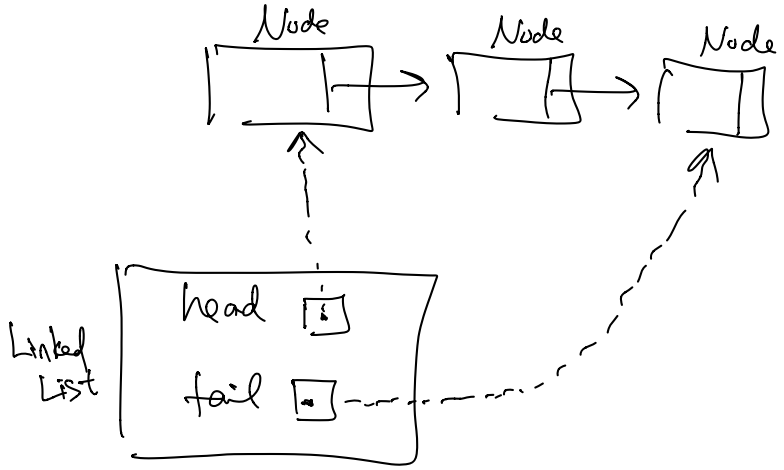
Build a Singly-linked-list class to hold ints.

Node class 

Node — 2 pieces → data; pointer/reference to the next item in the list.

```
class Node {
    int data;
    Node next;   <-- this is not a Node "inside" a Node, it is a reference to the next Node.
}

class LinkedList {
    Node head;
    (Node tail;)
}
```



Traversals : going through the LL one element at a time.

Forward direction (left to right)

```
Node curr = (wherever you want to start traversing)   [normally this is the head]
while( stopping condition
       is not true
       (normally curr != null)
{
    //processing step with
    // curr.data

    curr = curr.next;
}
```