Middlebury

CSCI 201: Data Structures

Fall 2024

Lecture 6T: Linked Lists

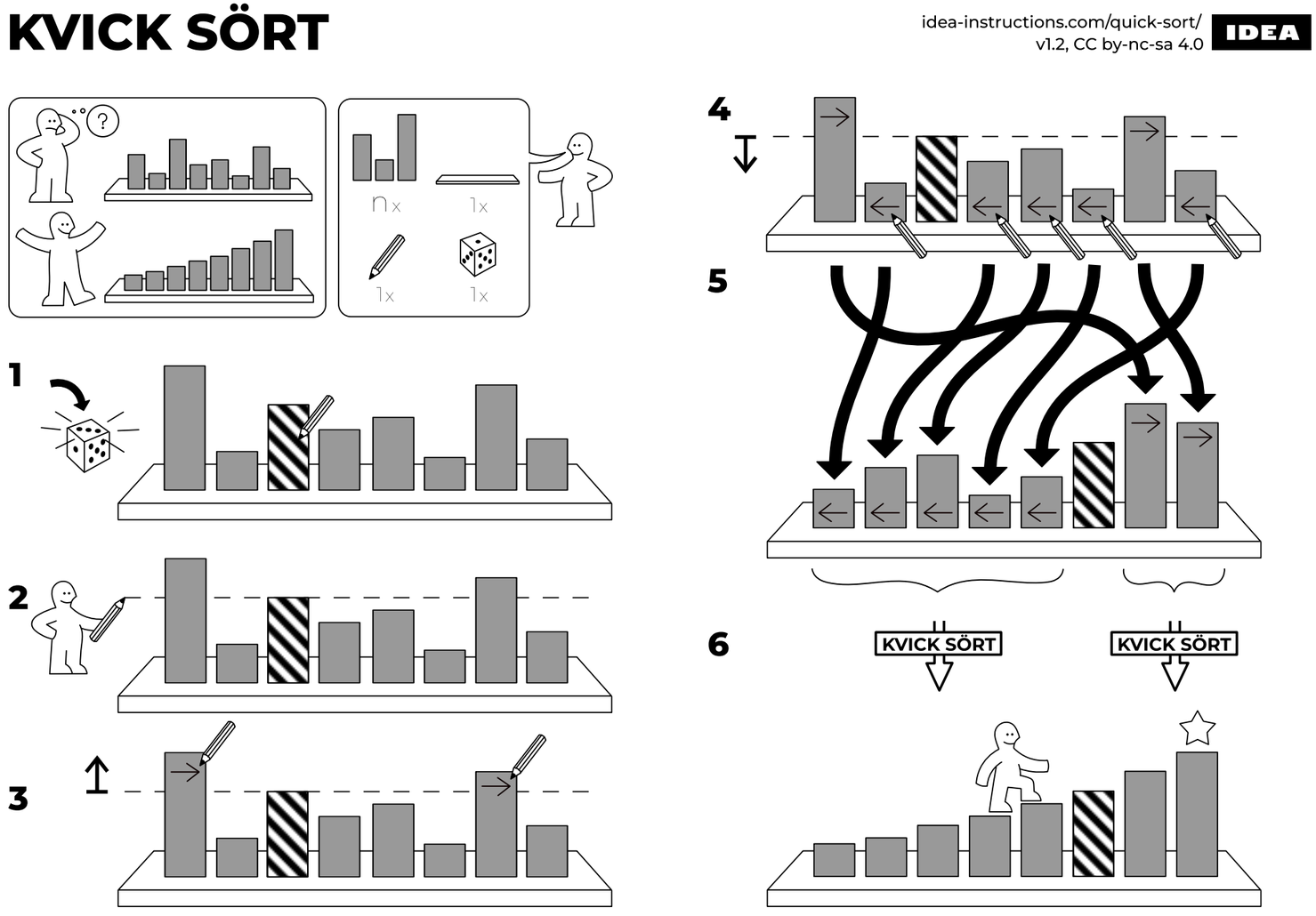
Goals for today:

- Analyze the runtime complexity of `QuickSort` (since we didn't have time last class).
- Motivate the use of the singly-`LinkedList` data structure.
- **Insert** a node into a `LinkedList` right after a given node.
- **Remove** a node from a `LinkedList` right after a given node.
- Reinforce the concept of **references**, which can be used to link nodes.

] + some other methods

Sorting algorithm #6: `QuickSort` (from last class).

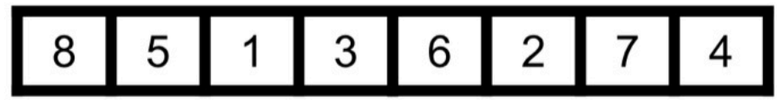
1. Pick a pivot.
2. Partition items so than any item < pivot is in left subarray and any item > pivot is in the right subarray.
3. Call `QuickSort` on each subarray.



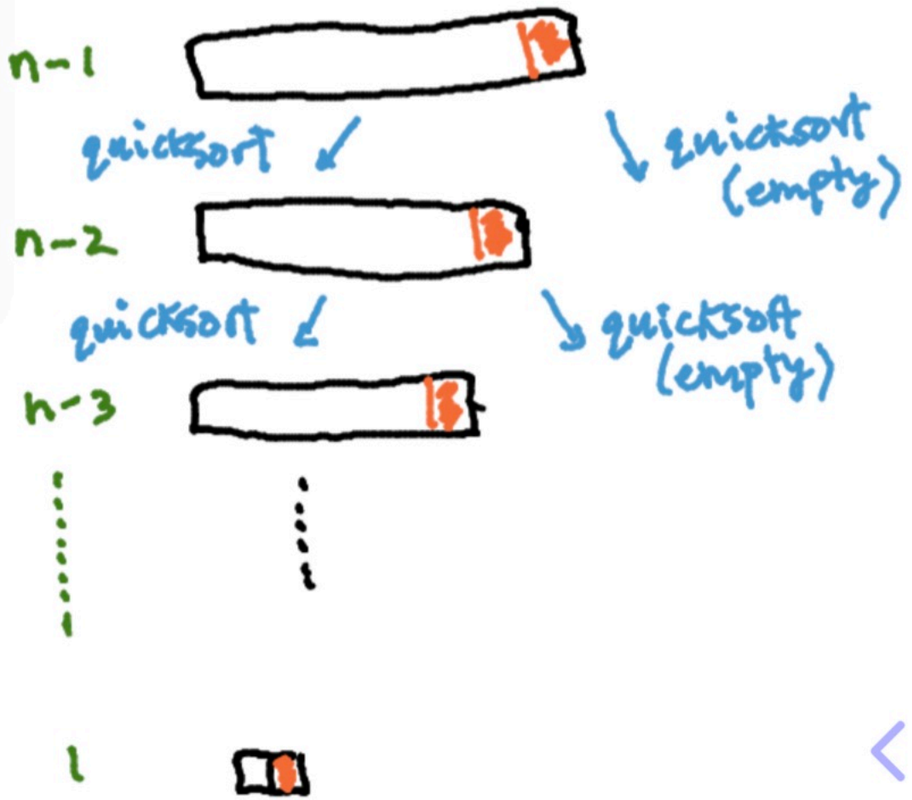
Sorting algorithm #6: `QuickSort` (from last class).

1. Pick a pivot. we'll use the last item in current array.
2. Partition items so than any item < pivot is in left subarray and any item > pivot is in the right subarray.
3. Call `QuickSort` on each subarray.

```
1  public static void sort(int[] items) {
2    quicksortHelper(items, 0, items.length - 1);
3  }
4
5  private static void quicksortHelper(int[] items,
6                                      int left, int right) {
7    if (left >= right) return;
8
9    // pick a pivot and partition items to the left/right
10   int p = partition(items, left, right);
11
12   // call quicksort on left and right subarrays
13   quicksortHelper(items, left, p - 1);
14   quicksortHelper(items, p + 1, right);
15 }
```



worst case: array is already sorted



total # comparisons: $(n-1) + (n-2) + (n-3) + \dots + 3 + 2 + 1$

$1 + 2 + 3 + \dots + (n-1) + n = \frac{n(n+1)}{2}$

$\approx \frac{n(n+1)}{2}$

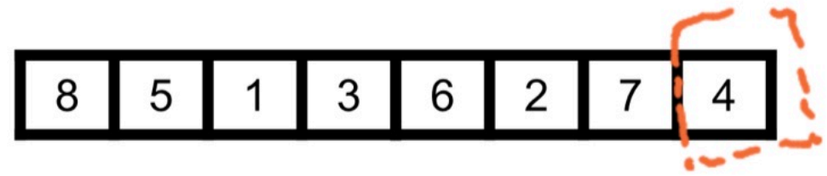
$O(n^2)$

worst case

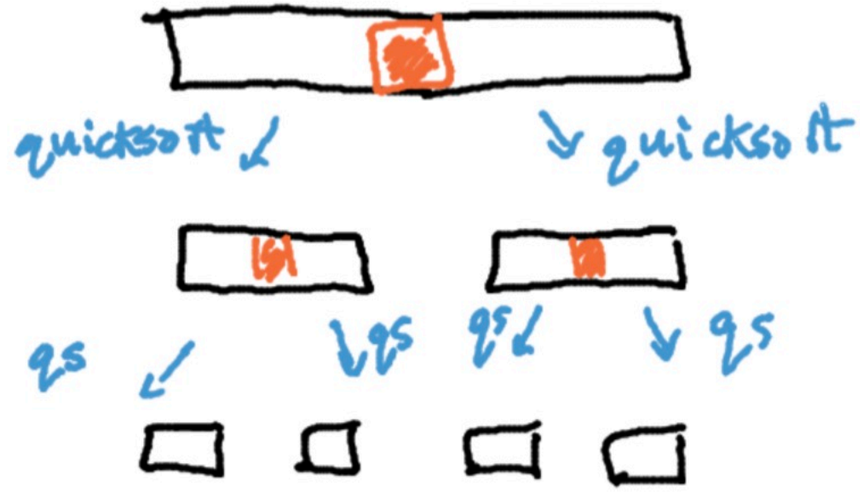
Sorting algorithm #6: `QuickSort` (from last class).

1. Pick a pivot.
2. Partition items so than any item < pivot is in left subarray and any item > pivot is in the right subarray.
3. Call `QuickSort` on each subarray.

```
1  public static void sort(int[] items) {
2    quicksortHelper(items, 0, items.length - 1);
3  }
4
5  private static void quicksortHelper(int[] items,
6                                      int left, int right) {
7    if (left >= right) return;
8
9    // pick a pivot and partition items to the left/right
10   int p = partition(items, left, right);
11
12   // call quicksort on left and right subarrays
13   quicksortHelper(items, left, p - 1);
14   quicksortHelper(items, p + 1, right);
15 }
```



best, average case : pivot ends up in the middle of array



how many levels? (until we get to arrays of size 1)

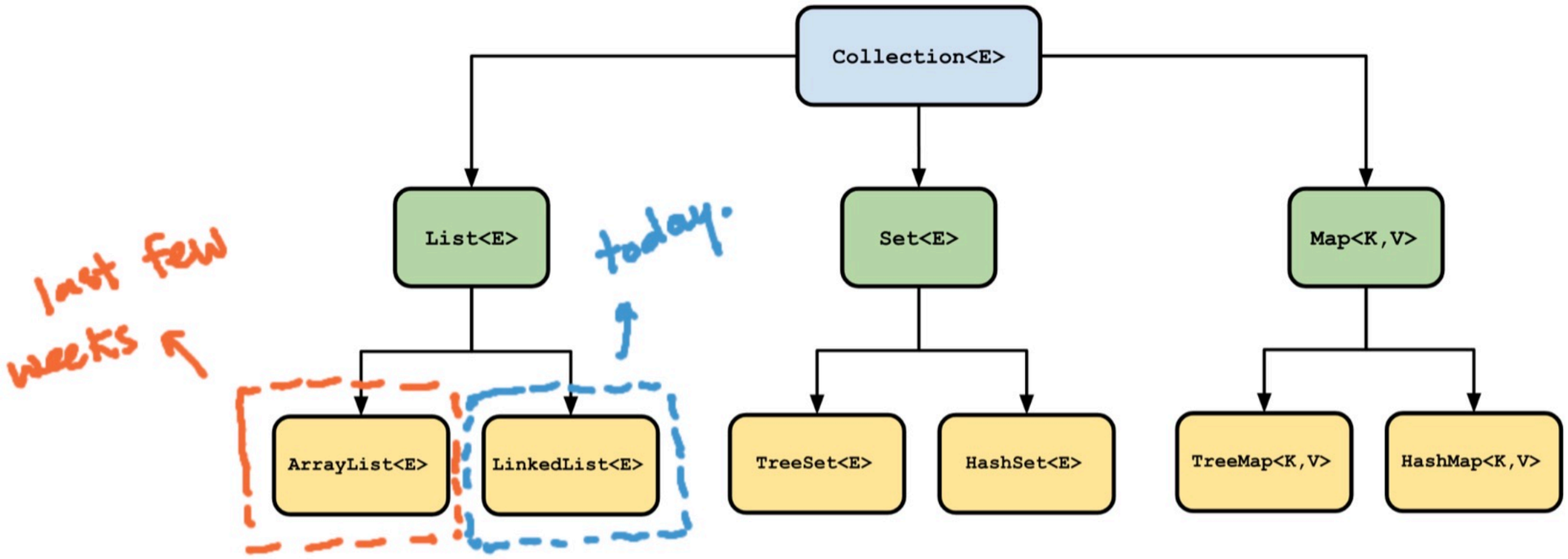
$\log_2(n)$ (similar to mergesort)

≈ n comparisons per level

total work = $n \times \log n \rightarrow O(n \log n)$

best, average case

Now, let's talk about the `List` *interface* again!

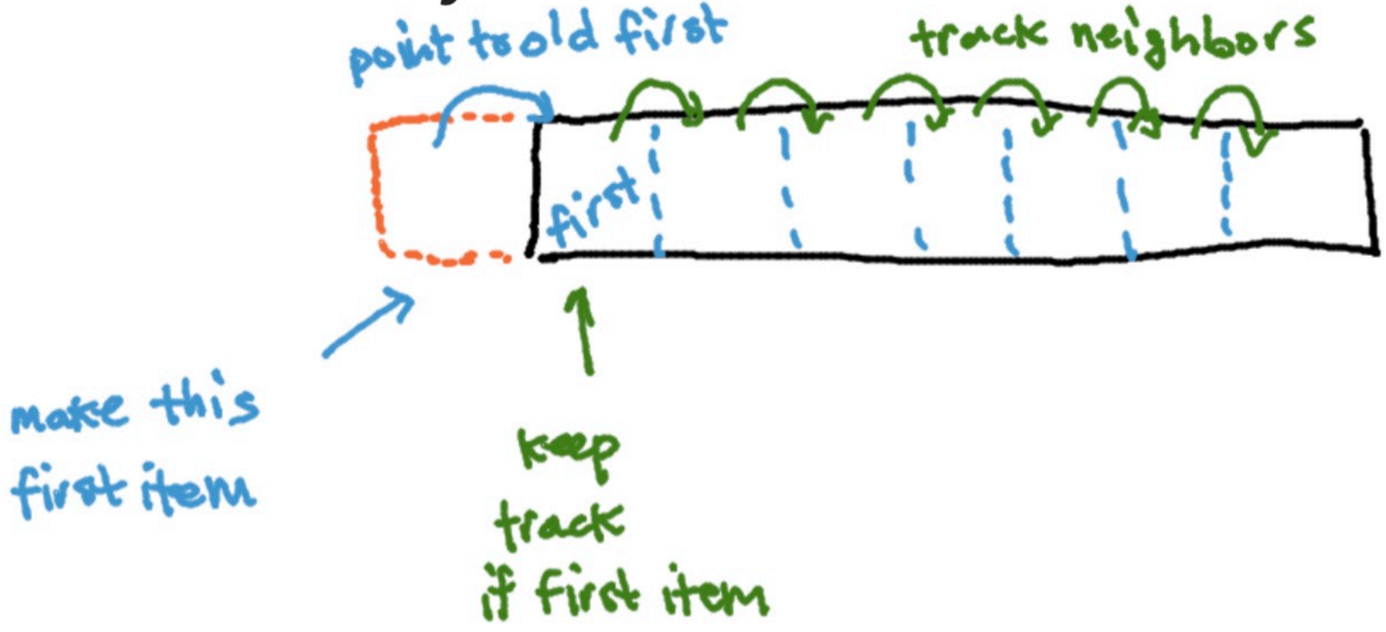


`ArrayList` was good for adding to the end, but what if we want to add to the **front**?

Let's experiment and see what happens... (open up `ListAddFrontTest.java`).

n	time (s)
10k	5×10^{-7}
100k	4.7×10^{-6}
1M	slow :(

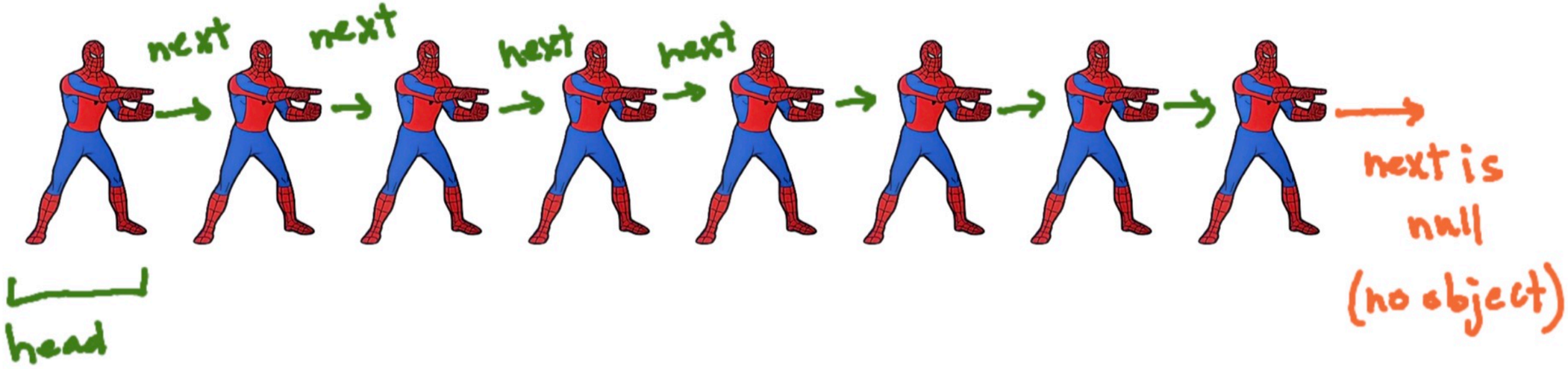
Is there a way to achieve *constant* time addition to the front?



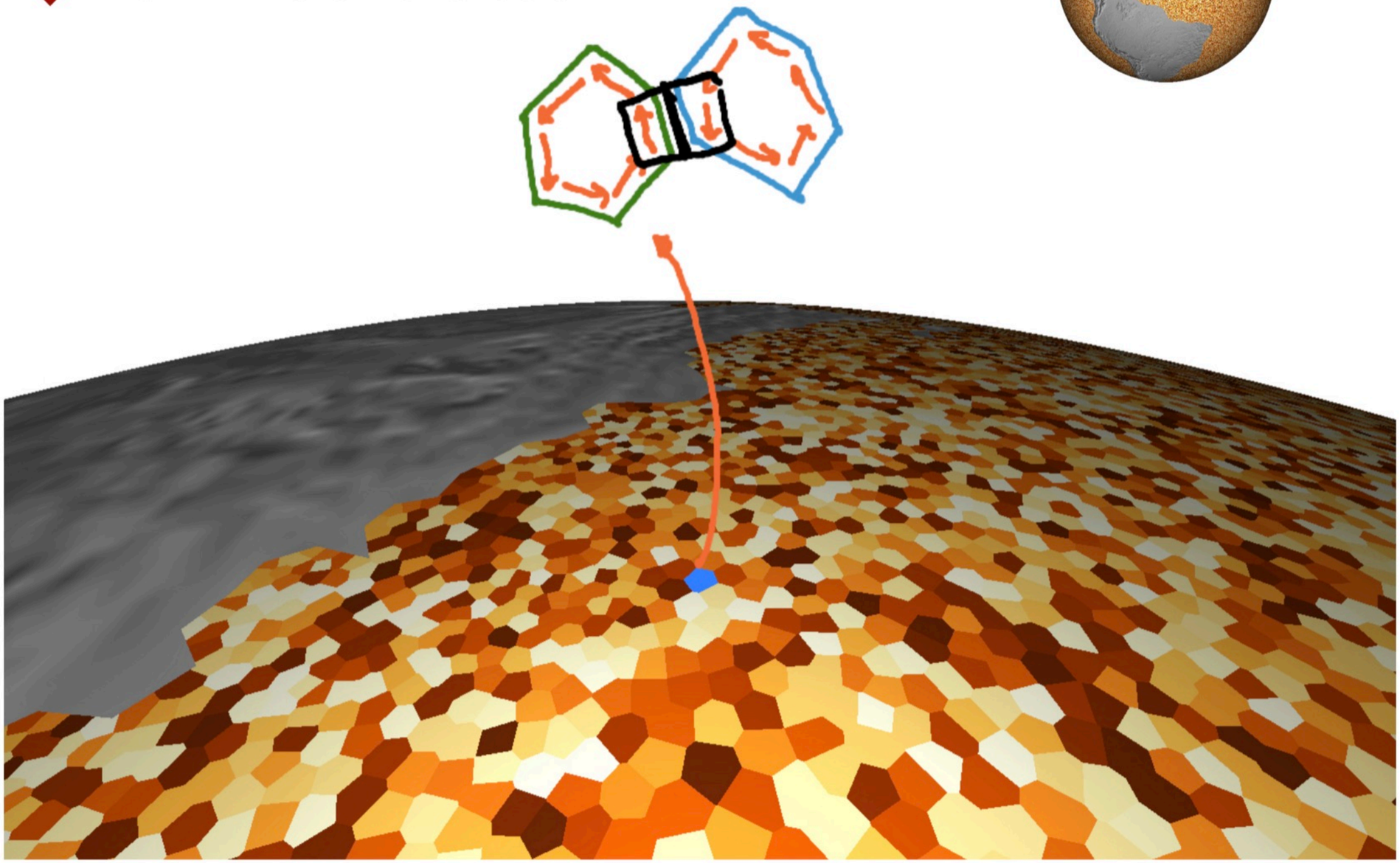
with ArrayList, need to shift everything to the right, O(n)

💡 **Idea:** what if we kept track of which elements of the list are next to each other?

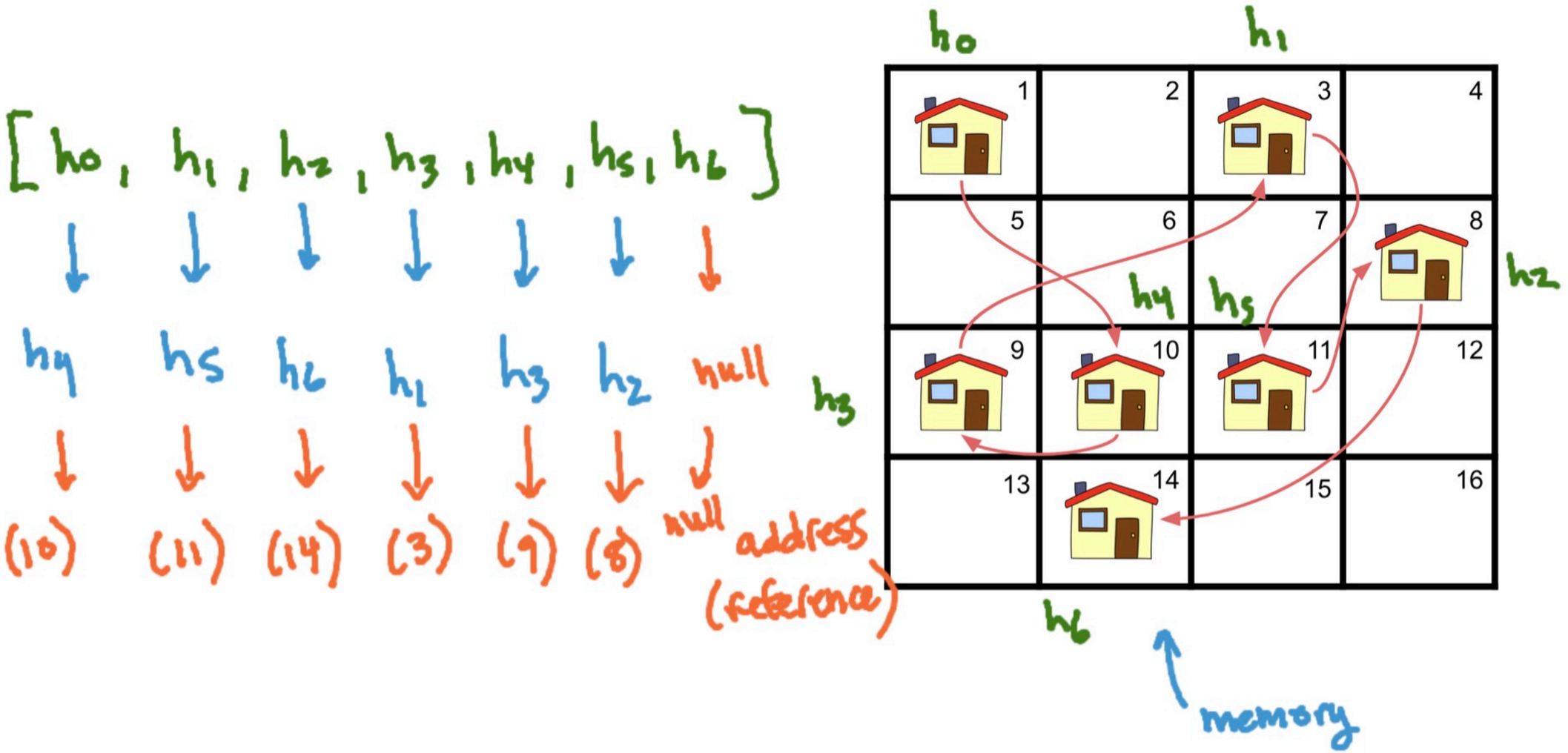
This is the main idea of linked lists!



I ❤️ linked lists.



How do we achieve this idea of "pointing" to other nodes?

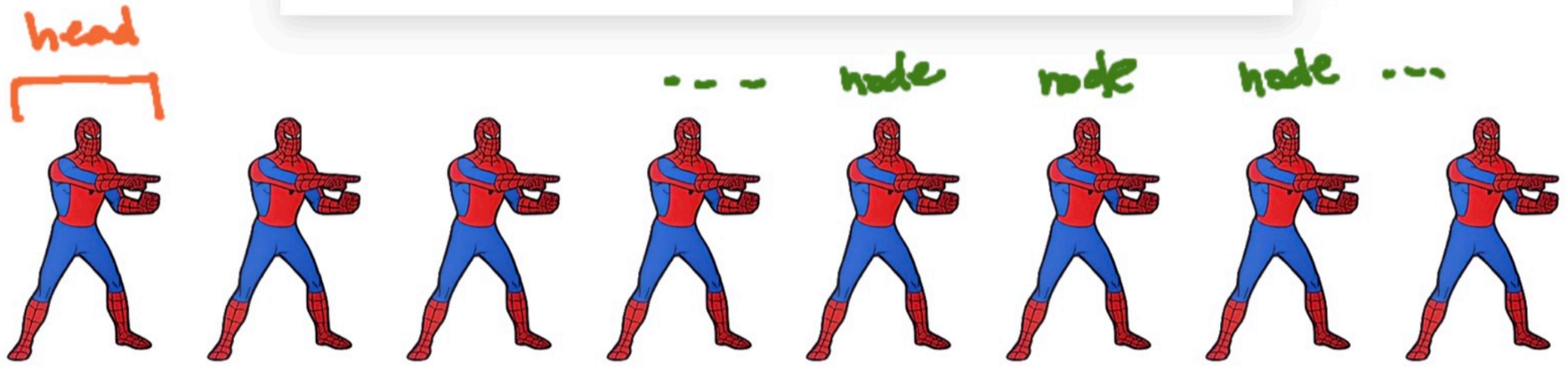


👍 👎 😳 🤔 😊 ✋ ❤️

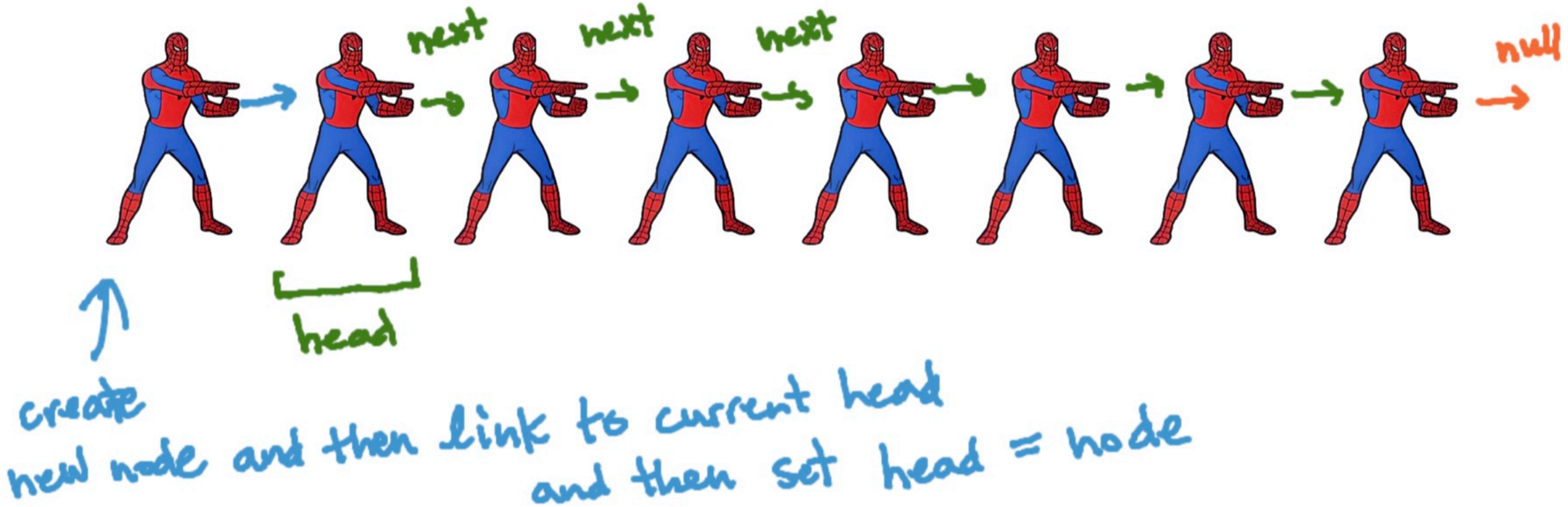
15 0 1 0 2 0 1

Designing our own `LinkedList` (for integers)!

```java
class ListNodeInt {
  public ListNodeInt next;
  public int data;

  public ListNodeInt(int data) {
    this.data = data;
    next = null;
  }
}

public class LinkedListInt {
  private ListNodeInt head;

  public LinkedListInt() {
    head = null;
  }
}
```



Inserting a node at the beginning of the list.




```
1 public void addFront(int data) {
2   ListNodeInt node = new ListNodeInt(data);
3   node.next = head;
4   head = node;
5 }
```

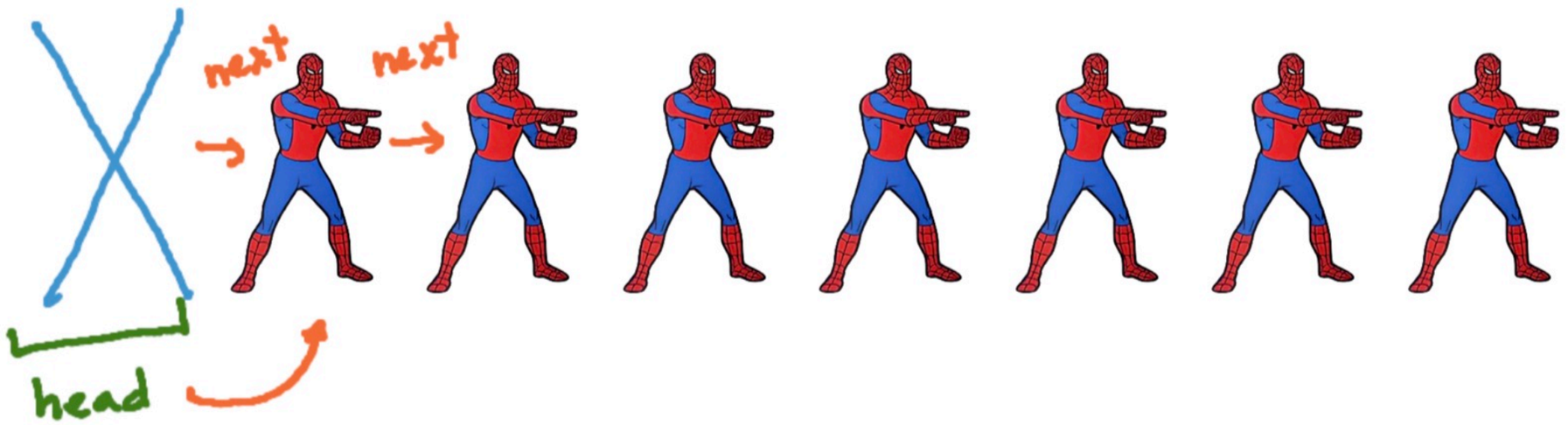
Okay, let's see how fast this is!

```
1 public void add(int index, int data) {
2   if (index == 0) {
3     addFront(data);
4   } else {
5     // we'll be able to do this soon...
6   }
7 }
```

Then go back to `ListAddFrontTest.java` and change the following line:

```
1 //ArrayList<Integer> list = new ArrayList<>();
2 LinkedListInt list = new LinkedListInt();
```

Removing a node from the beginning of the list.



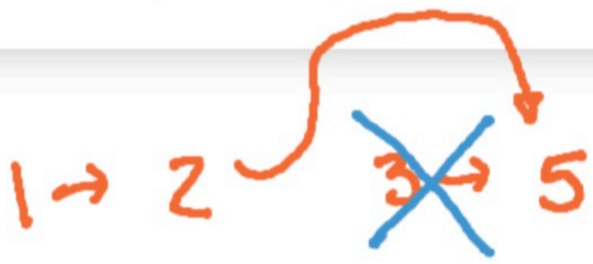

```
1 public void removeFront(int data) {
2   if (head != null) {
3     head = head.next;
4   }
5 }
```

Let's write a method to print out the linked list.

```
1  public String toString() {
2    String result = "";
3    ListNodeInt node = head;
4    while (node != null) {
5      result += node.data;
6      if (node.next != null) result += " -> ";
7      node = node.next;
8    }
9  }
10
11 // then in a PSVM
12 System.out.println(list);
```

What will be printed here?

```
1 LinkedListInt list = new LinkedListInt();
2 list.addFront(5);
3 list.addFront(3);
4 list.removeFront();
5 list.addFront(2);
6 list.addFront(1);
7 System.out.println(list);
```



cs201-lecture06T

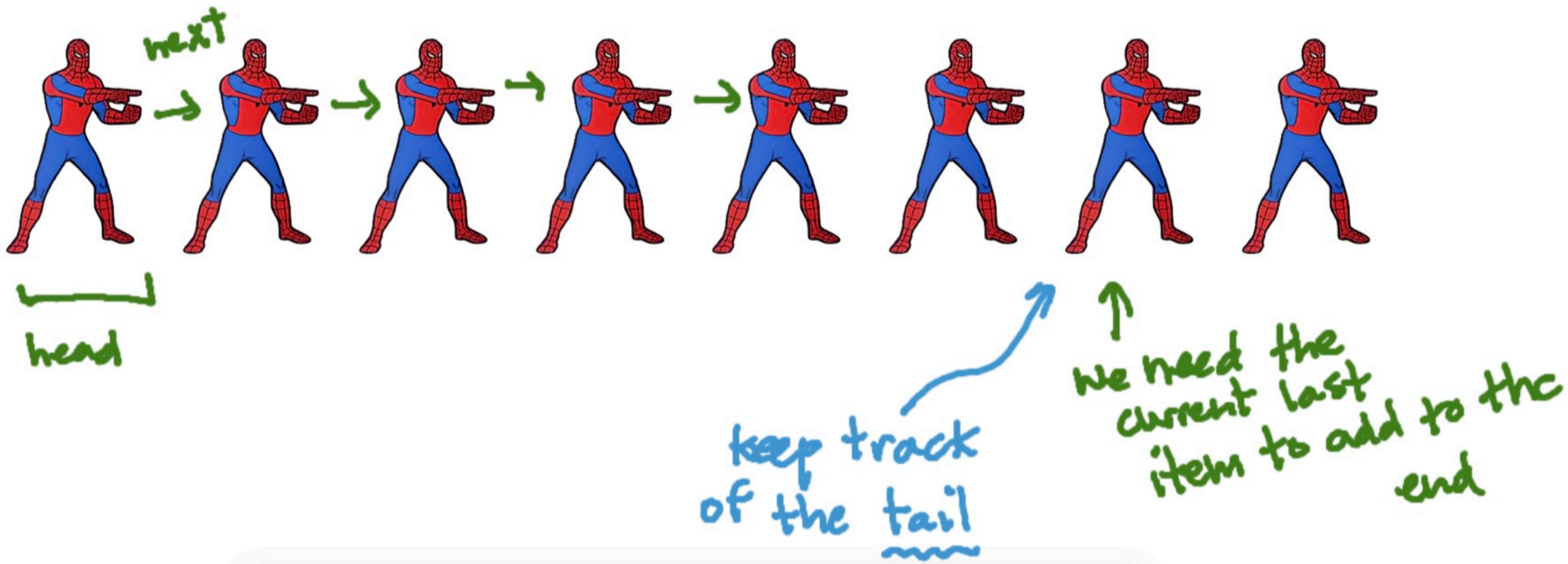
What will be printed? (slido.com #7122080) 40

- 1 -> 2 -> 5
- 1 -> 3 -> 5
- 1 -> 2 -> 3
- 3 -> 2 -> 1
- 5 -> 3 -> 1
- 5 -> 2 -> 1

Send

Voting as Anonymous

What if we want to add a node to the end?




```java
public class LinkedListInt {
  ListNodeInt head;
  ListNodeInt tail;

  void addLast(int data) {
    ListNodeInt node = new ListNodeInt(data);
    tail.next = node;
    tail = node;
  }
}
```

And finally, what if we wanted to design our own text editor?

See you on Thursday!

- We'll extend these ideas to implement a **doubly-linked list**.
- We'll also **generic**-ify our linked list implementation.
- **Homework 4** due tonight (10/15) at 11:59pm.
    1. Implement Radix Sort.
    2. Derive number of `=` for `DIYList add` method with a different growth technique.
- **Midterm Study Guide** will be posted by the end of the week.
- Reminder that Noah (go/noah) and Smith (go/smith) have office hours throughout the week and the 201 Course Assistants have drop-in hours in the late afternoons/evenings (go/cshelp).
- Submit exit ticket 6T today.