Person in CS: Ellen Ochoa

- BS physics (‘75), PhD Electrical Eng. (‘85).
- Starting working on software for optical recognition systems (computer vision)
- Applied to be an astronaut in...
  - ‘85...rejected
  - ‘87...rejected
  - ‘90...accepted!!!
- Worked on flight software, computer hardware, and robotics
- First Hispanic woman in space ’93
- Director of NASA Johnson Space Flight Center (Houston) ‘13
Announcements, Coming up

• Today, Monday 2/27
  • Nothing due

• Wednesday 3/1
  • APT 5 (linked list problems) due

• Next Monday 3/6
  • Project P3: DNA (linked list project) due

• Midterm exam 2? Pushed back to Wed. 3/22 after spring break.
Today’s outline

1. Wrapping up DIYLinkedList:
   • Add to front of linked list
   • Writing an Iterator

2. Introducing Recursion
   • Counting ListNodes
   • Reversing a LinkedList

3. Sorting algorithms: Recursive Mergesort
More DIYLinkedList

Live Coding

• Add to front?
• Efficient iterator?
Toward Recursion by counting nodes: Iterative vs. Recursive

• Standard linked list iteration
  • Advance local pointer, do something at each node
    ```java
    public int countIter(ListNode list) {
        int total = 0;
        while (list != null) {
            total += 1;
            list = list.next;
        }
        return total;
    }
    ```

• Recursion?
  • Base Case?
  • General case?
  • Define size using size?
    ```java
    public int size(ListNode list) {
        if (list == null) return 0;
        return 1 + size(list.next);
    }
    ```
Key ideas in recursion

1. **Base case**: Easy answer when input small
2. **Recursive call(s)**: Get answer on subset of input
3. **Do something** with the result of the recursive call(s) and then return

• **Note**: Method does not call itself
  • Calls identical clone, with its own state
  • Methods/calls stacked, like all methods
Thinking recursively

1. When is the input small enough that the answer is trivial? Base case.

2. Otherwise, suppose an oracle could solve the exact some problem on a smaller subset of the input.

3. Could you solve the larger problem given what the oracle told you?
The call stack: How recursion works on a machine

- Each method call gets its own **call frame** (local variables, etc.)

- **Eager evaluation:** Invoking method does not resume until invoked method returns.

```java
public int size(ListNode list) {
    if (list == null) return 0;
    return 1 + size(list.next);
}
```
Eager evaluation and substitution

- Return value will be substituted into the expression calling the method.

```java
public int size(ListNode list) {
    if (list == null) return 0;
    return 1 + size(list.next);
}
```
Eager evaluation and substitution

- Return value will be substituted into the expression calling the method.

```java
public int size(ListNode list) {
    if (list == null) return 0;
    return 1 + size(list.next);
}
```
Eager evaluation and substitution

• Return value will be substituted into the expression calling the method.
Eager evaluation and substitution

• Return value will be substituted into the expression calling the method.

```java
public int size(ListNode list) {
    if (list == null) return 0;
    return 1 + size(list.next);
}
```
Counting Nodes

```java
public int size(ListNode list) {
    if (list == null) return 0;
    return 1 + size(list.next);
}

int result = size(ptr);
System.out.println(result);
```
Recursive runtime

- Concept is the same: Count the number of constant time operations...across all recursive calls!
- Ensure each recursive call gets closer to the base case, else code may run forever.

```java
public int size(ListNode list) {
    if (list == null) return 0;
    return 1 + size(list.next);
}
```

- Moves one node toward the base case at each step.
- List of N nodes, makes O(N) recursive calls, each is O(1).
- Overall O(N) runtime complexity.
Recall the reverse problem

• How do we reverse nodes in a linked list
  • Go from A->B->C to C->B->A
  • Typical interview style question
  • [https://leetcode.com/problems/reverse-linked-list/](https://leetcode.com/problems/reverse-linked-list/)
  • [https://www.hackerrank.com/challenges/reverse-a-linked-list](https://www.hackerrank.com/challenges/reverse-a-linked-list)
Base case, words and code

• Base case: When is there nothing to do?
  • A list with 0 or 1 nodes is its own reverse

```java
3 public static ListNode reverse(ListNode list) {
4     if (list == null || list.next == null) {
5         return list;
6     }
```
Recursive step in words

• Suppose the recursion oracle (a recursive call) reverses the list \textit{after the first node}.

• How to use? Just put the first node at the end!

• Restated: The reverse of a list is \textit{the reverse of all but the first element}, with the first element added to the end.
Recursive step in pictures

\[ \text{list} \rightarrow \text{A} \rightarrow \text{B} \rightarrow \text{C} \rightarrow \text{|||} \]

Returned by recursive call on \text{list.next}

\[ \text{list} \rightarrow \text{A} \rightarrow \text{C} \rightarrow \text{B} \rightarrow \text{|||} \]

\text{reversedFirst} \rightarrow \text{reversedLast} \rightarrow \text{list} \rightarrow \text{|||}

Make \text{reversedLast} point to what \text{list} points to

Return \text{reversedFirst}
Recursive step in code

7       ListNode reversedLast = list.next;
8       ListNode reversedFirst = reverse(list.next);

Note that list.next still refers to reversedLast
Recursive step in code (continued)

9 \text{ reversedLast}.next = \text{list}; \quad \text{Make B point to A}
10 \text{list}.next = \text{null}; \quad \text{Make A point to null}
11 \text{return reversedFirst}; \quad \text{Return overall reversed list}
Putting it all together

```java
public static ListNode reverse(ListNode list) {
    if (list == null || list.next == null) {
        return list;
    }
    ListNode reversedLast = list.next;
    ListNode reversedFirst = reverse(list.next);
    reversedLast.next = list;
    list.next = null;
    return reversedFirst;
}
```
Revisiting the call stack: How it really works

reverse(list) → A → B → C ───> | | |
Revisiting the call stack: How it *really* works

reverse(list) → A → B → C

reverse(list)
reversedFirst
Revisiting the call stack: How it really works

Back to the case we considered first
WOTO

Go to duke.is/6effd

Not graded for correctness, just participation.

Try to answer *without* looking back at slides and notes.

But do talk to your neighbors!
Consider the rec method. If the input list is ['A', 'B', 'C'], what will be returned by rec(list)?

```java
3   public static ListNode rec(ListNode list) {
4       if (list == null || list.next == null) {
5           return list;
6       }
7       ListNode after = rec(list.next);
8       if (list.info <= after.info) {
9           list.next = after;
10          return list;
11       }
12       return after;
13   }
```

- ['A']
- ['C']
- ['A', 'B']
- ['C', 'B']
- ['A', 'B', 'C']
- ['C', 'B', 'A']
Same rec method. If the input list is ['C', 'B', 'A'], what will be returned by `rec(list)`?

```java
public static ListNode rec(ListNode list) {
    if (list == null || list.next == null) {
        return list;
    }
    ListNode after = rec(list.next);
    if (list.info <= after.info) {
        list.next = after;
        return list;
    }
    return after;
}
```

- ['A']
- ['C']
- ['A', 'B']
- ['C', 'B']
- ['A', 'B', 'C']
- ['C', 'B', 'A']
Same rec method. For an input list with \( N \) nodes, the best characterization of the runtime complexity of \( \text{rec(list)} \) is... *

```java
public static ListNode rec(ListNode list) {
    if (list == null || list.next == null) {
        return list;
    }
    ListNode after = rec(list.next);
    if (list.info <= after.info) {
        list.next = after;
        return list;
    }
    return after;
}
```

- \( O(1) \)
- \( O(N) \)
- \( O(N^2) \)
- \( O(N^3) \)
Consider the mystery method. Note that it is the same as rec except for lines 24-29. If the input list is ['C', 'B', 'A'], what will be returned by mystery(list)? *

```java
public static ListNode mystery(ListNode list) {
    if (list == null || list.next == null) {
        return list;
    }

    ListNode after = mystery(list.next);
    if (list.info <= after.info) {
        list.next = after;
        return list;
    }

    ListNode current = after;
    while (current.next != null && list.info > current.next.info) {
        current = current.next;
    }

    list.next = current.next;
    current.next = list;
    return after;
}
```

- ['A']
- ['C']
- ['A', 'B']
- ['B', 'C']
- ['A', 'B', 'C']
- ['C', 'B', 'A']
Same mystery method. For an input list with $N$ nodes, the best characterization of the runtime complexity of $\text{mystery(list)}$ is...

\begin{verbatim}
    public static ListNode mystery(ListNode list) {
        if (list == null || list.next == null) {
            return list;
        }
        ListNode after = mystery(list.next);
        if (list.info <= after.info) {
            list.next = after;
            return list;
        }
        ListNode current = after;
        while (current.next != null && list.info > current.next.info) {
            current = current.next;
        }
        list.next = current.next;
        current.next = list;
        return after;
    }
\end{verbatim}