public static boolean contains(int[] nums, int x) {
    return containsHelper(nums, x, nums.length);
}

private static boolean containsHelper(int[] nums, int x, int curIdx) {
    if (curIdx == 0)
        return false;
    return nums[curIdx] == x || containsHelper(nums, x, curIdx - 1);
}

• What is the time complexity of the code above?

  A. O(n)
  B. O(log n)
  C. O(n log n)
  D. O(n^2)
  E. Whatever
Administrative Details

• Lab 1
  • I apologize for not having it returned yet
  • Feedback will show up on github as a Pull Request (PR)
  • PRs give you the option to view comments line-by-line, and respond to comments

• (New workflow this semester, so it is taking time to get the kinks worked out. It should be faster turnaround than printouts once it is working.)
Agenda

- Induction
  - List
(Proof by) Induction

- The mathematical cousin of recursion is induction:

  **Base case:**
  - Prove for 0, 1, ...

  **Inductive case:**
  - Assume true for \( n-1 \)
  - Prove true for \( n \)

**Recursion**

- In recursion, we always use the same basic approach/structure
  - base case
  - recursive case
Mathematical Induction

• Prove that for every $n \geq 0$

\[ 2^0 + 2^1 + 2^2 + \ldots + 2^n = 2^{n+1} - 1 \]

**Base Case:**

$n = 0$: $2^0 = 2^0 - 1$

$1 = 2 - 1$

$1 = 1 \checkmark$

**Inductive Case:**

Assume $2^0 + 2^1 + \ldots + 2^{n-1} = 2^{n+1} - 1$ (IH)

Show $2^0 + 2^1 + \ldots + 2^n = 2^{n+1} - 1$

$2^0 - 1 \ldots + 2^n = 2^{n+1} - 1$ by IH

$= 2 \cdot 2^n - 1$

$= 2^{n+1} - 1 \checkmark$
Mathematical Induction

• Prove that for every \( n \geq 0 \)

\[
0 + 1 + \ldots + n = \frac{n(n+1)}{2}
\]

\[
\begin{align*}
\text{h = 0} & \quad 0 = \frac{0(0+1)}{2} \quad \checkmark \\
\text{Assume} & \quad 0 + \ldots + (n-1) = \frac{(n-1)(n-1+1)}{2} \quad \text{(IH)} \\
\text{Show} & \quad 0 + \ldots + n = \frac{n(n+1)}{2} \\
0 + \ldots + n & = \frac{n(n-1)}{2} + n \quad \text{by IH} \\
& = \frac{n(n-1) + 2n}{2} \\
& = \frac{n^2 + n}{2} \\
& = \frac{n(n+1)}{2}
\end{align*}
\]

\( \Box \)
Agenda

• Induction

☐ List
The List Interface

interface List {
    size()
    isEmpty()
    contains(e)
    get(i)
    set(i, e)
    add(i, e)
    remove(i)
    addFirst(e)
    getLast()
    ...
    ...
    ...
}

Vector implements List

Singly Linked List
Pros and Cons of Vectors

**Pros**

- Fast access to elements
- Dynamically Resizeable?

**Cons**

- Slow updates to front of list
- Hard to predict time for add (depends on internal array size)
- Potentially wasted space

---

**Array**

- An array is stored in consecutive memory locations:

```java
int[] nums;
nums = new int[5];
```

New location = size of Array * Index

---

**Yes, but inefficient**
Singly Linked List

- There are two key components of Lists
  - The list itself
    - Instance variables
      - (Pointer to) the head node of the list
    - Methods
      - Those declared in the List interface
  - Nodes
    - Instance variables
      - data
      - (Pointer to) the “next” element
    - Define methods
      - Getters and setters
Singly Linked List Methods

```java
public E get(int index) {
    Node finger = head;
    for (int i = 0; i < index; i++) {
        finger = finger.next;
    }
    return finger.value;
}
```
public E set(E d, int index) {
    E ori = finger.value();
    finger.setValue(d);
    return ori;
}
public void add(E d, int index) {

```java
16
myList

head
myNode

D

value

myNode

next

B

value

myNode
del

C

value

myNode
del

D

value

myNode
del

C

value

myNode
del

D

value

myNode
del

C

value

myNode
del

D

value

myNode
del

C

value

myNode
del

D

value

myNode
del

C

value
```