CompSci 201, L13: Recursion

Logistics, Coming up

- APT 6 due *tomorrow*, Thursday 10/13

- Project 3: DNA due next Monday, 10/17

- APT Quiz 1 due next Wednesday, 10/19
  - Will release over the weekend
  - No regular APTs next week, just the quiz

What is an APT Quiz?

- Set of 3 APT problems, 2 hours to complete.
  - Will be available starting this weekend (sakai announcement to release)
  - Must *complete* by 11:59 pm Wednesday 10/19 (so start before 10)

- Start the quiz on Sakai assessments tool, begins your timer and shows you the link to the problems and submission page.
  - Will look/work just like the regular APT page, just with only 3 problems.

What is allowed?

Yes, allowed

Same as Mini exam:

- Zybook
- Course notes
- API documentation
- VS Code
- JShell

No, not allowed

- Collaboration or sharing any code.
- Communication about the problems ***at all*** during the window.
- Searching internet, stackoverflow, etc. for solutions.

Don’t do these things

1. Do not collaborate. Note that we log all code submissions and will investigate for academic integrity.

2. Do not hard code the test cases (if(input == X) return Y, etc.).

   We show you the test cases to help you debug. But we search for submissions that do this and **you will get a 0 on the APT quiz if you hard code the test cases** instead of solving the problem.

How is it graded?

Not curved, adjusted. 3 problems, 10 points each.

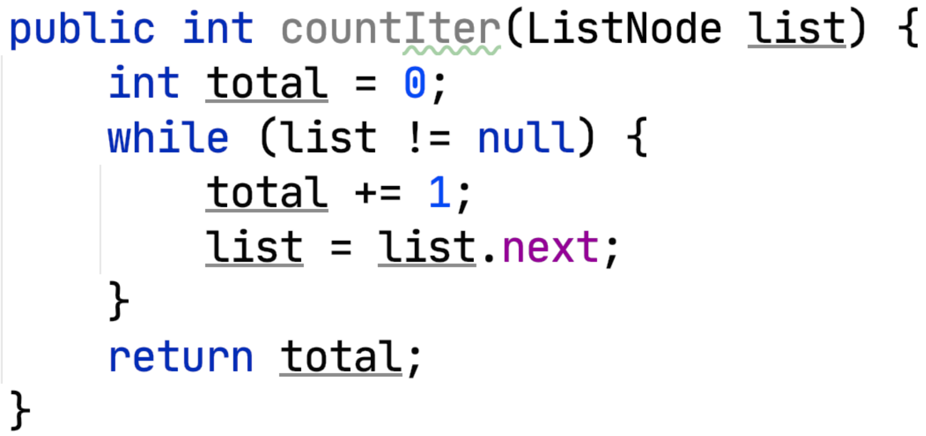
Raw score R out of 30.	Adjusted score A out of 30.	100 point grade scale
27 <= R <= 30	A = R	90 – 100
24 <= R <= 26	A = 26	~87
21 <= R <= 23	A = 25	~83
18 <= R <= 20	A = 24	80
15 <= R <= 17	A = 23	~77
12 <= R <= 14	A = 22	~73
9 <= R <= 11	A = 21	70
6 <= R <= 8	A = 20	~67
3 <= R <= 5	A = 19	~63
1 <= R <= 2	A = 18	60

Can still get in the B range even if you can't solve one; don't panic!

Only going to get a 0 if you collaborate or hard code test cases. Don't do it!

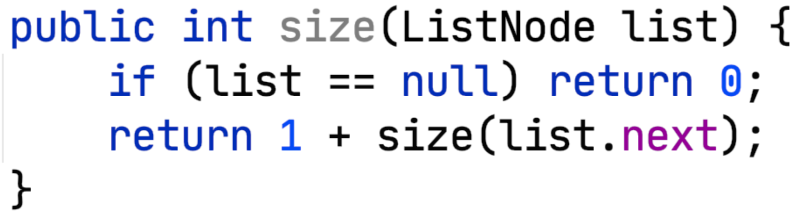
Toward Recursion by counting nodes

- 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);
}
```

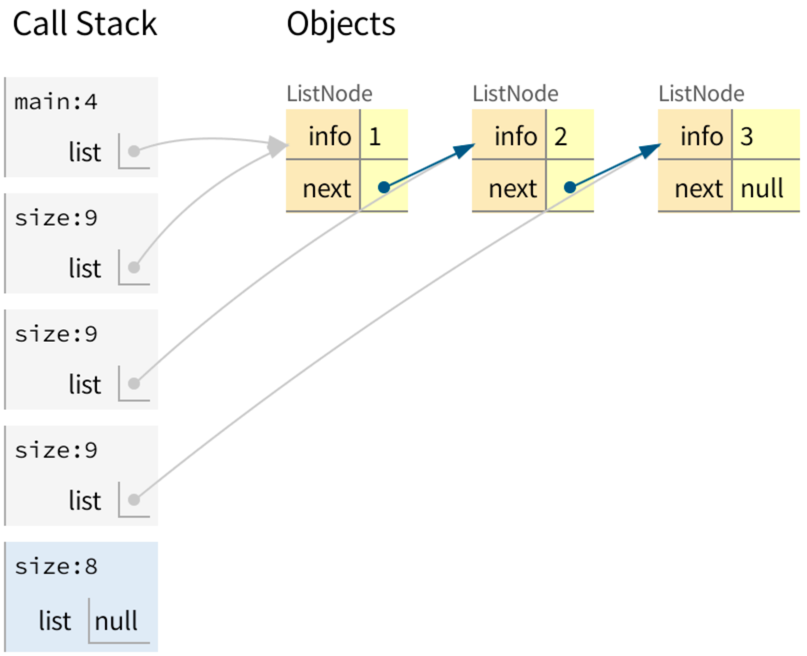
Key ideas in recursion

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

The call stack

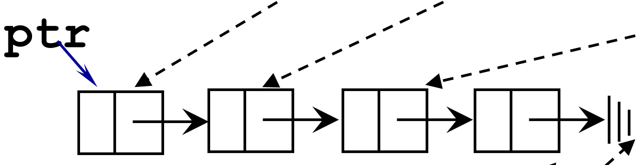
- Each method call gets its own *frame* (local variables, etc.)
- Invoking method does not resume until invoked method returns.



Counting Nodes

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

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



```
int size(Node list)
return 0;
```

4

```
int size(Node list)
 if (list == null) return 0;
 return 1 + size(list.next)
```

3

```
int size(Node list)
 if (list == null) return 0;
 return 1 + size(list.next)
```

2

```
int size(Node list)
 if (list == null) return 0;
 return 1 + size(list.next)
```

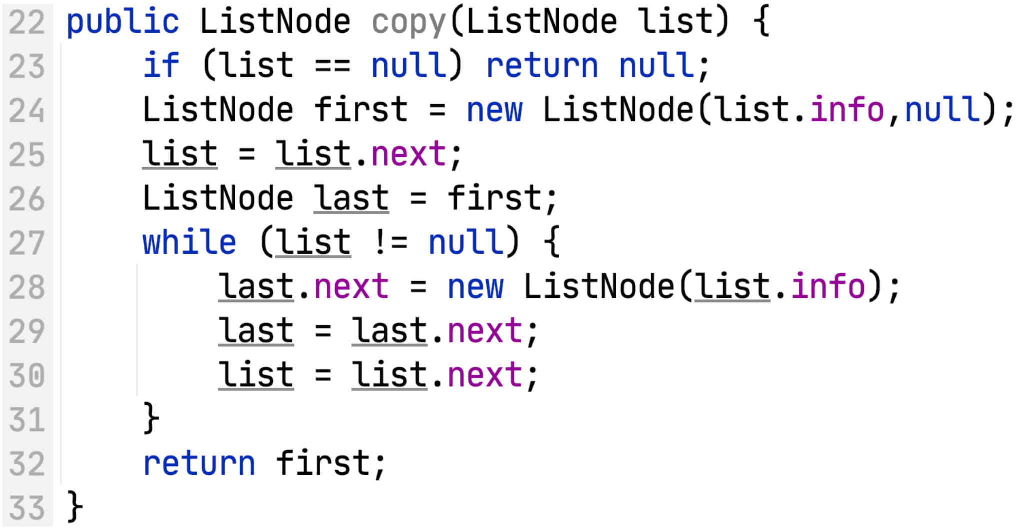
1

```
int size(Node list)
 if (list == null) return 0;
 return 1 + size(list.next)
```

Recursion can be simpler/shorter

- Making a copy of a linked list
  - Iterative: traverse front to back, add@back

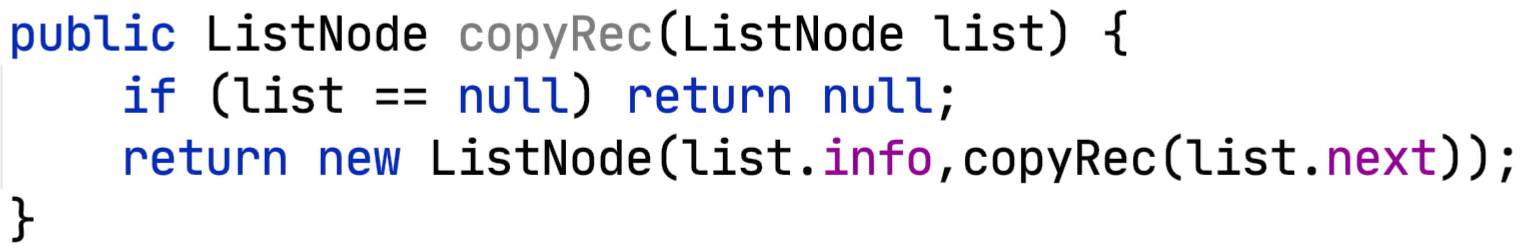
- Initialize first
- Call new, link
- Advance ptrs



```java
public ListNode copy(ListNode list) {
    if (list == null) return null;
    ListNode first = new ListNode(list.info,null);
    list = list.next;
    ListNode last = first;
    while (list != null) {
        last.next = new ListNode(list.info);
        last = last.next;
        list = list.next;
    }
    return first;
}
```

Copy via recursion

- Create one node, link to copy of rest
  - Base case is null, sometimes one node
  - Use result of recursive call
  - Note: it's assigned to a .next field, where?



```
public ListNode copyRec(ListNode list) {
    if (list == null) return null;
    return new ListNode(list.info,copyRec(list.next));
}
```

Developing and verifying recursive code

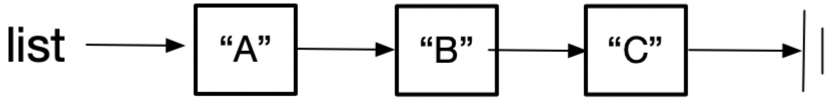
- Always verify base case
    - Always null, sometimes one node
- Check solution with small size: one or two
    - Trace through, look at recursive call
    - Be sure result of call is used
- Generalize from N nodes, trust N-1 nodes
    - Similar to proof by induction in math

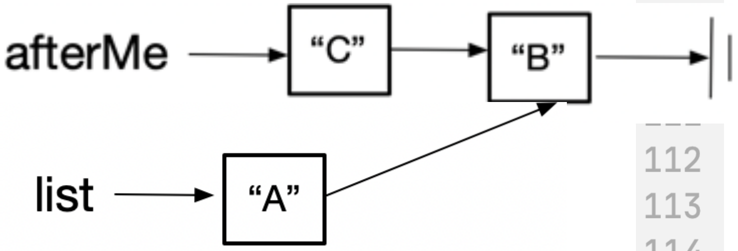
Recall the reverse problem

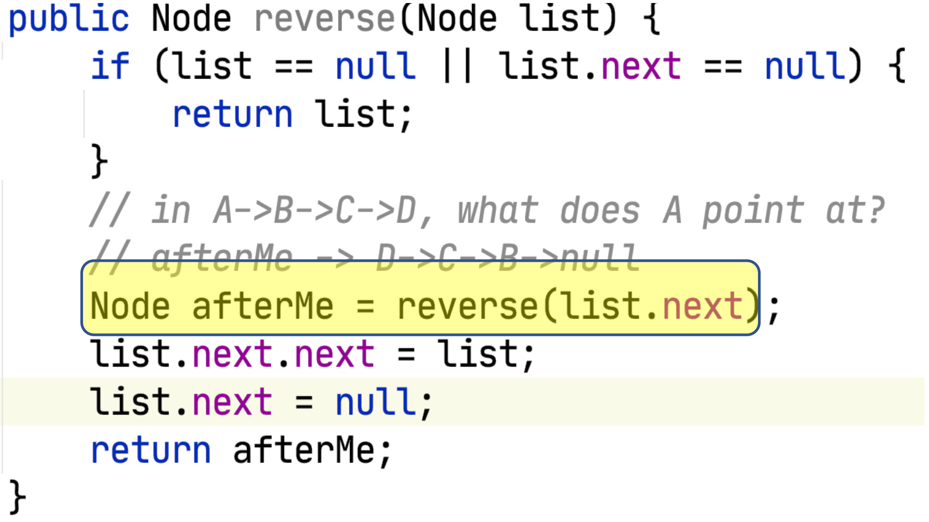
- How do we reverse nodes in a linked list
  - Go from A->B->C to C->B->A
  - Typical interview style question
  - https://leetcode.com/problems/reverse-linked-list/
  - https://www.hackerrank.com/challenges/reverse-a-linked-list

Recursive reverse

- list and list.next?
- trust recursion





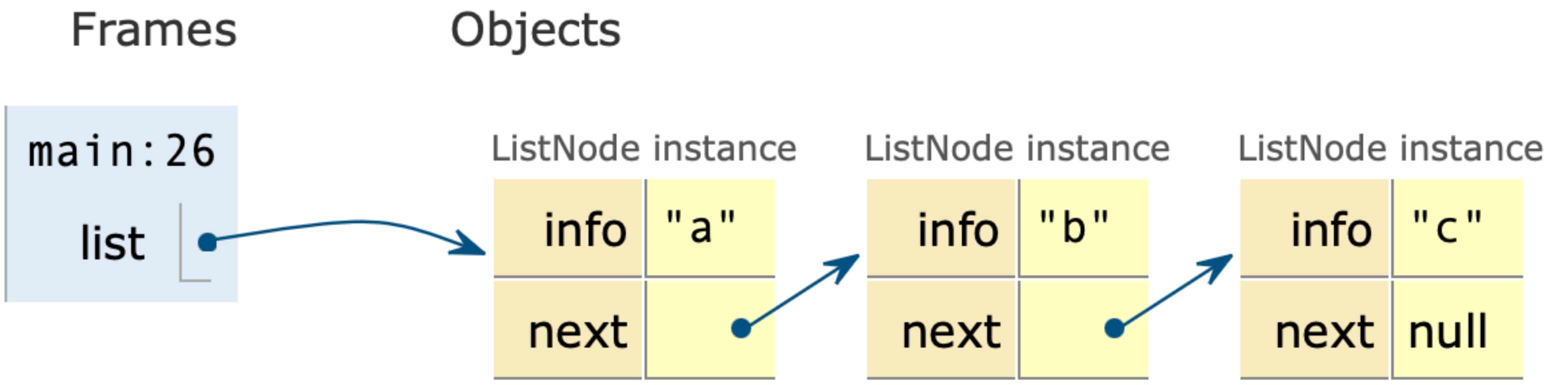


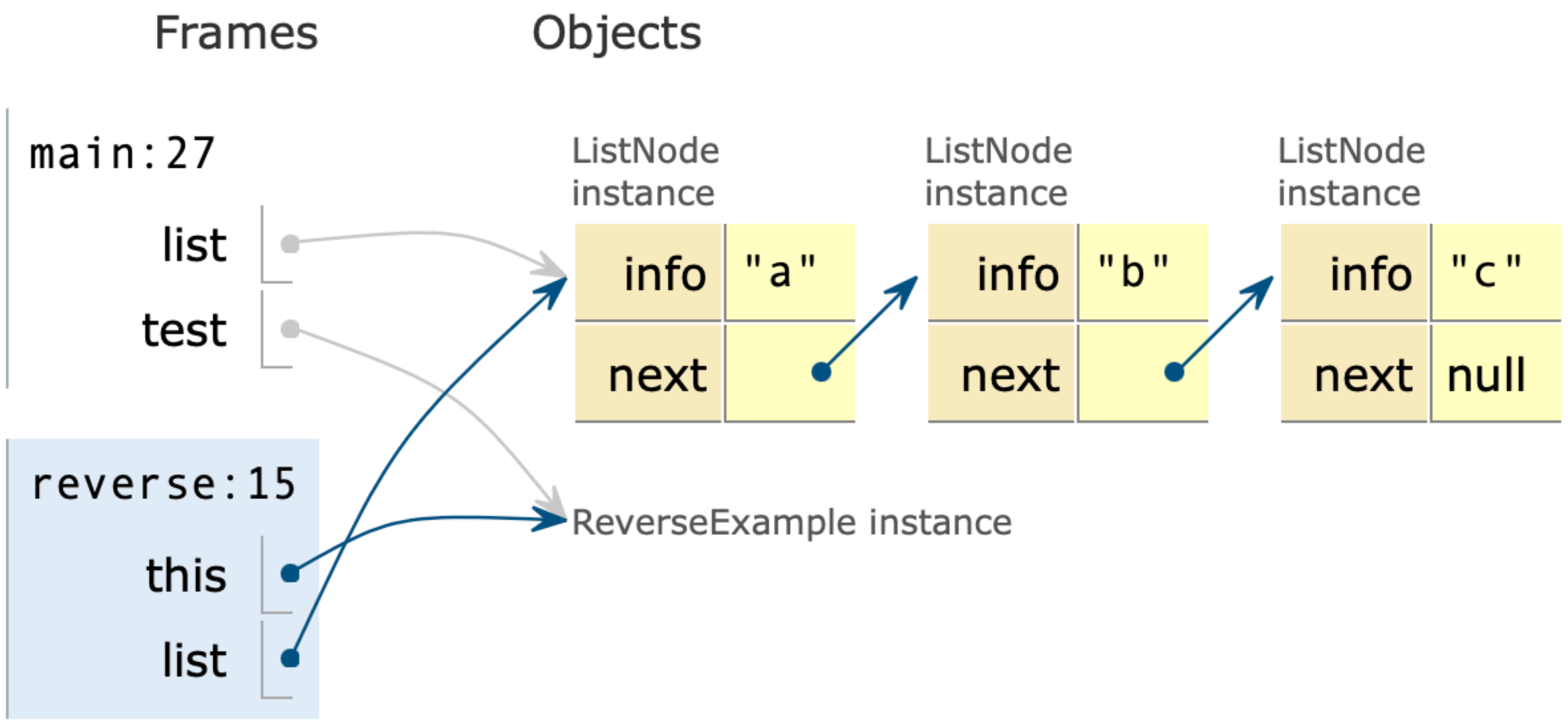
```
108 public Node reverse(Node list) {
        if (list == null || list.next == null) {
            return list;
        }
112     // in A->B->C->D, what does A point at?
113     // afterMe -> D->C->B->null
114     Node afterMe = reverse(list.next);
115     list.next.next = list;
116     list.next = null;
117     return afterMe;
118 }
```

- Without 116?
  - circular!

Visualizing Recursive reverse

- https://pythontutor.com/java.html





Frames

Objects

main:27

list

test

reverse:15

this

list

ListNode instance

info	"a"
next	

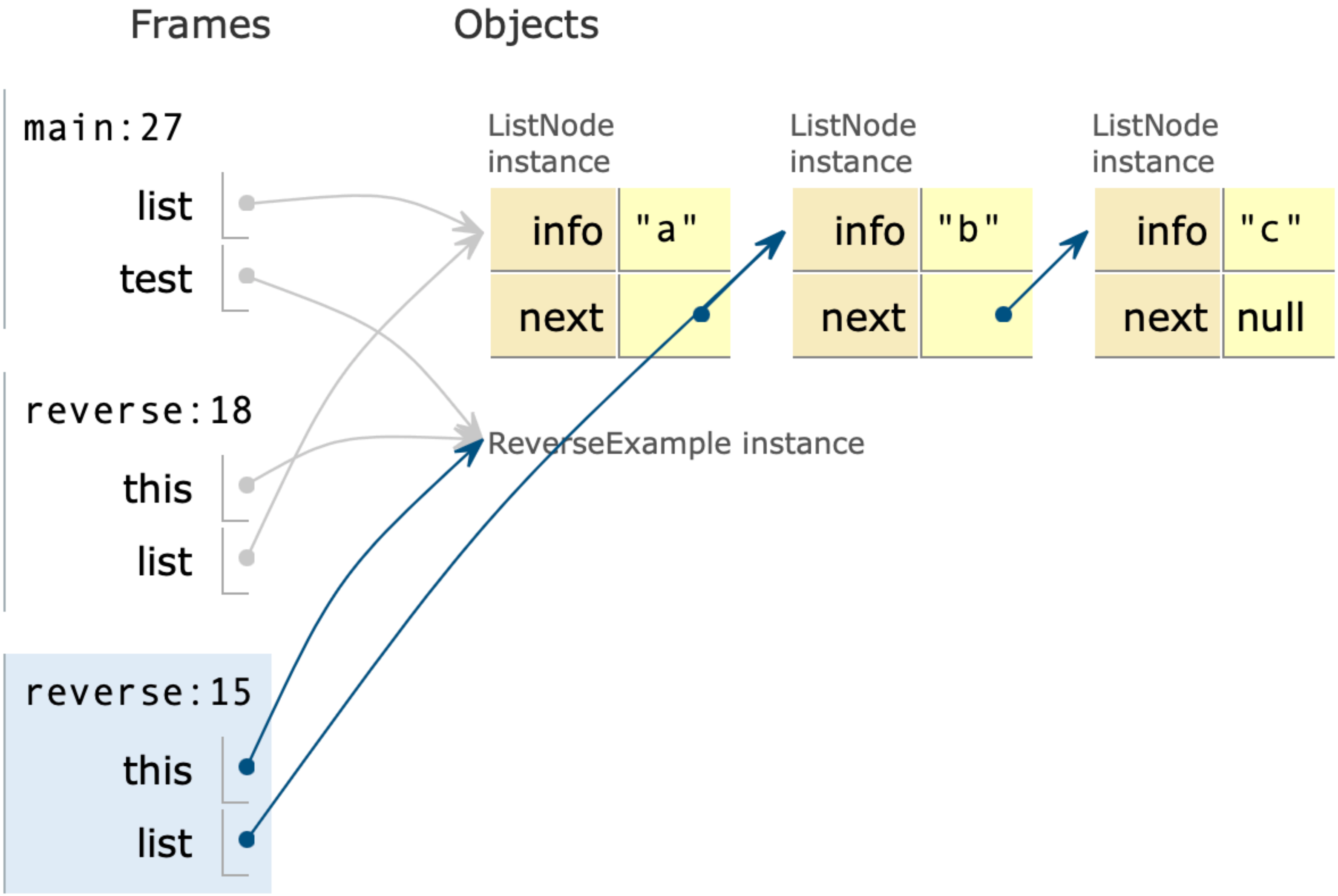
ListNode instance

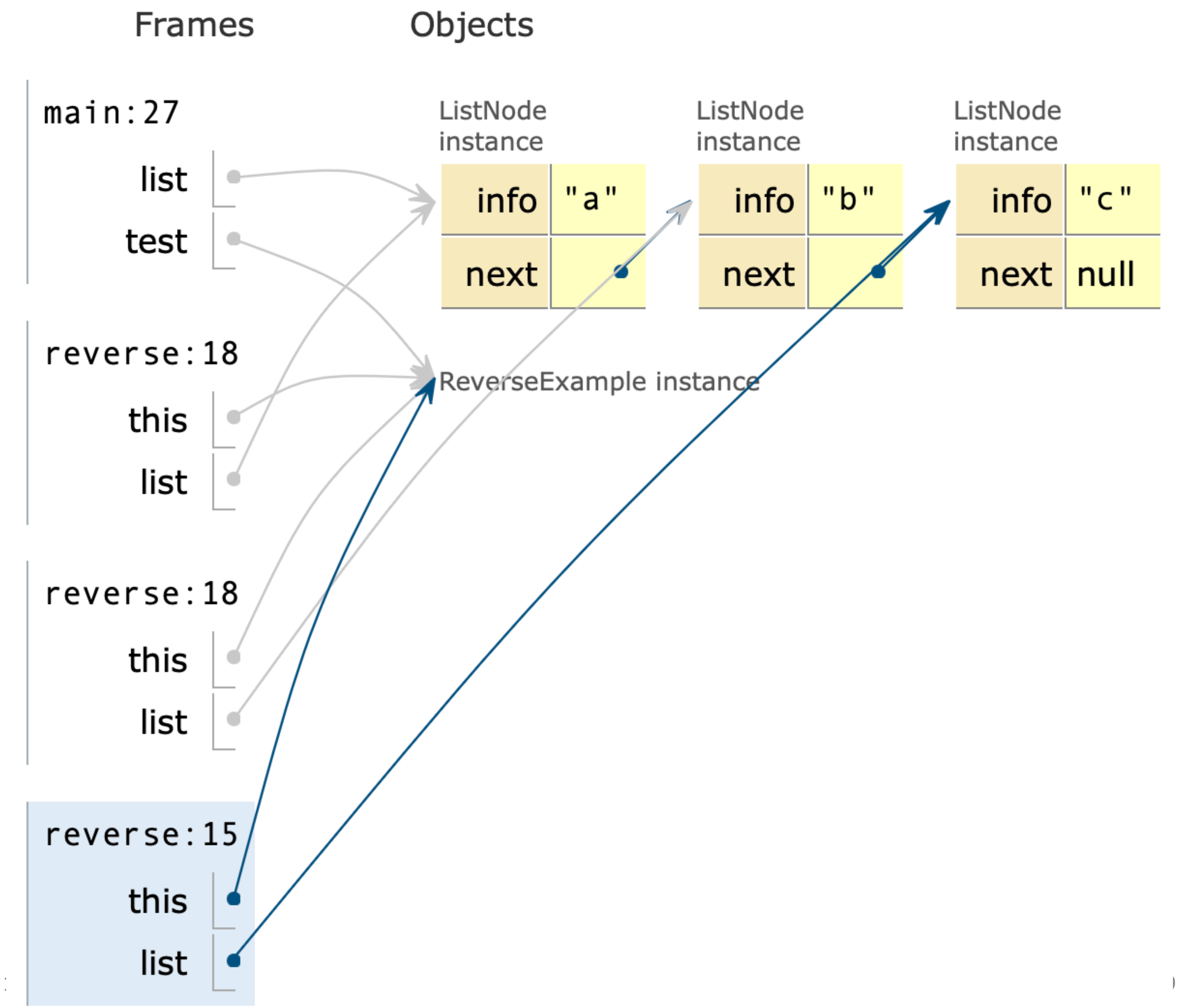
info	"b"
next	

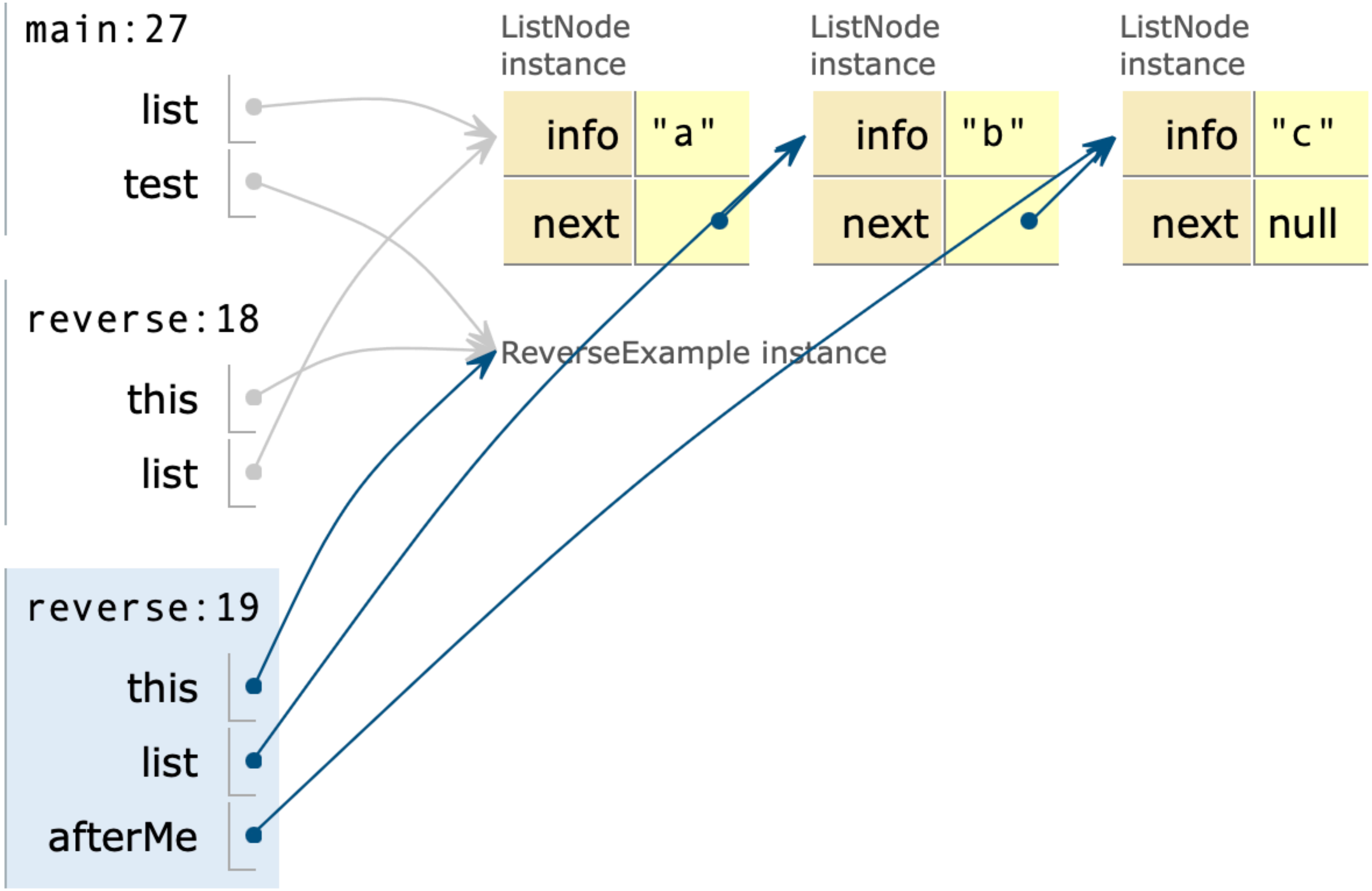
ListNode instance

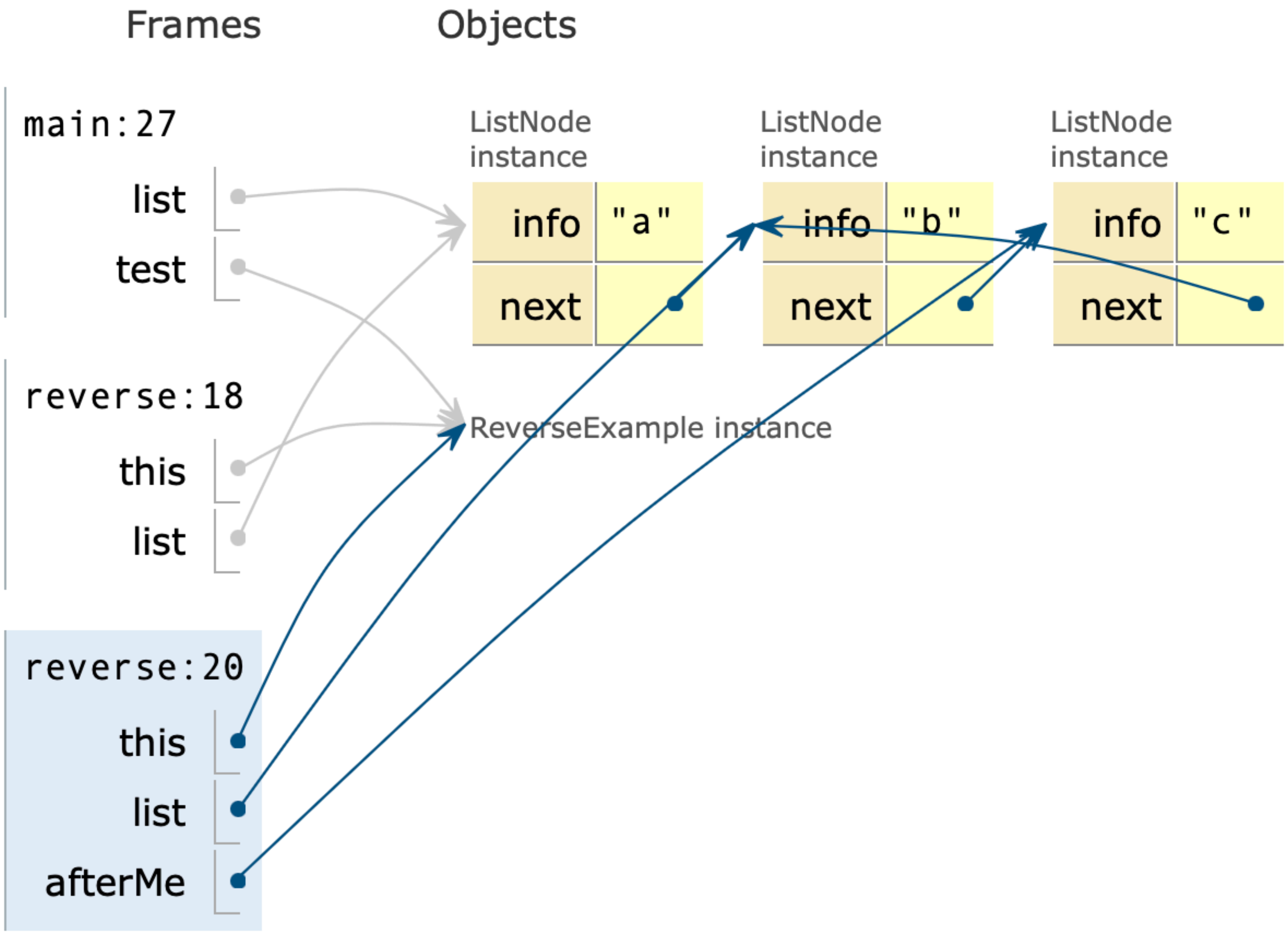
info	"c"
next	null

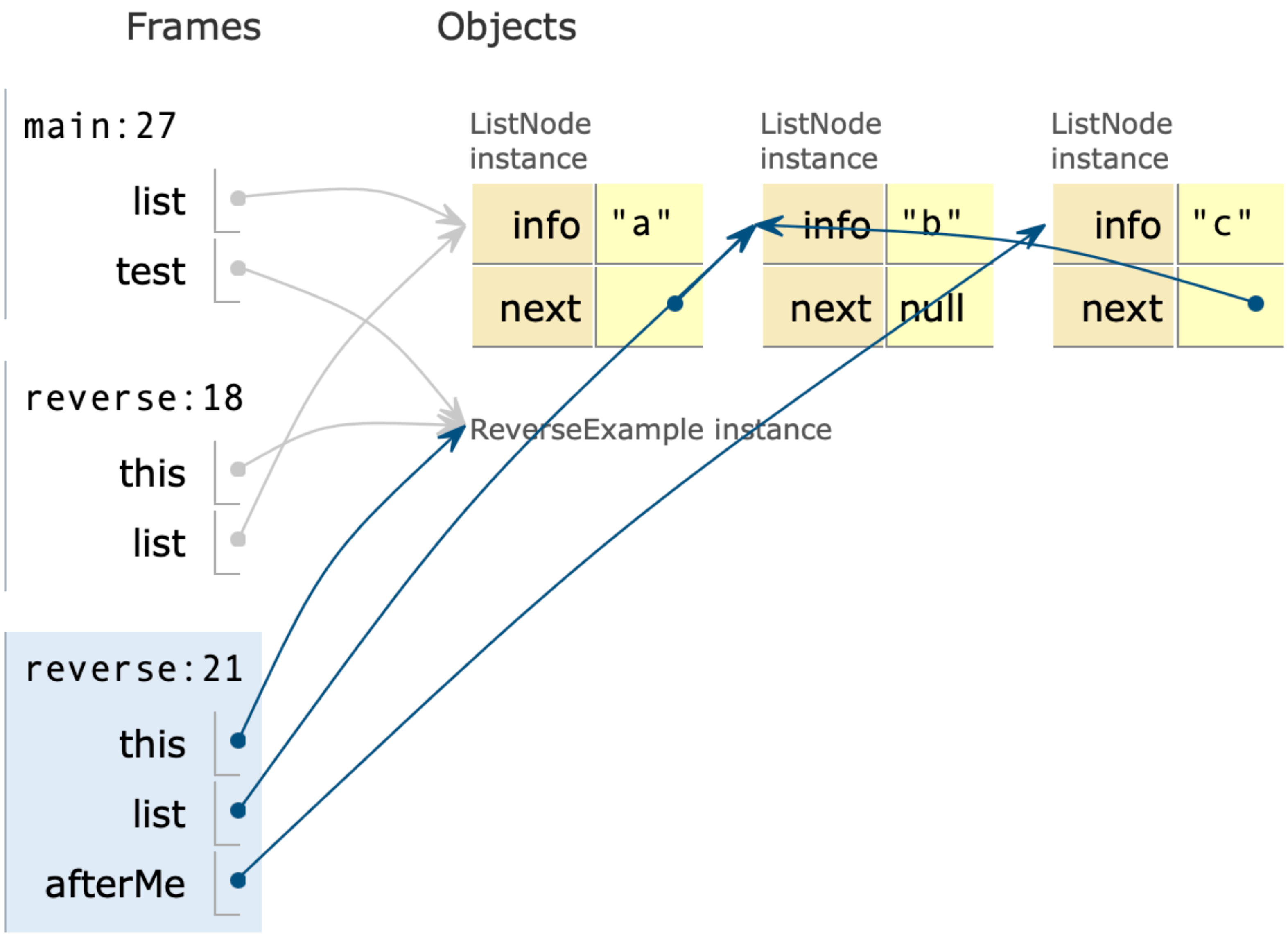
ReverseExample instance

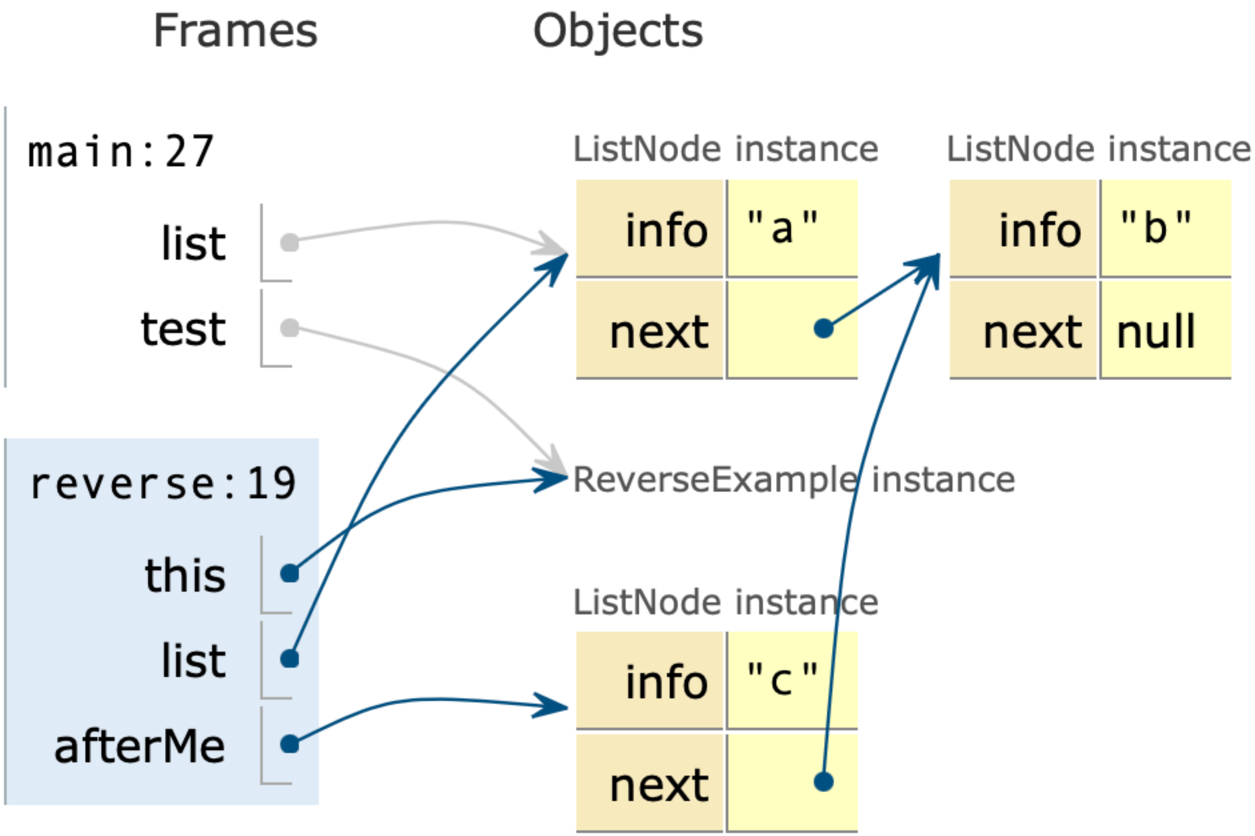

Frames
Objects
main:27
list
test
reverse:18
this
list
reverse:15
this
list
ListNode instance
info "a"
next
ListNode instance
info "b"
next
ListNode instance
info "c"
next null
ReverseExample instance

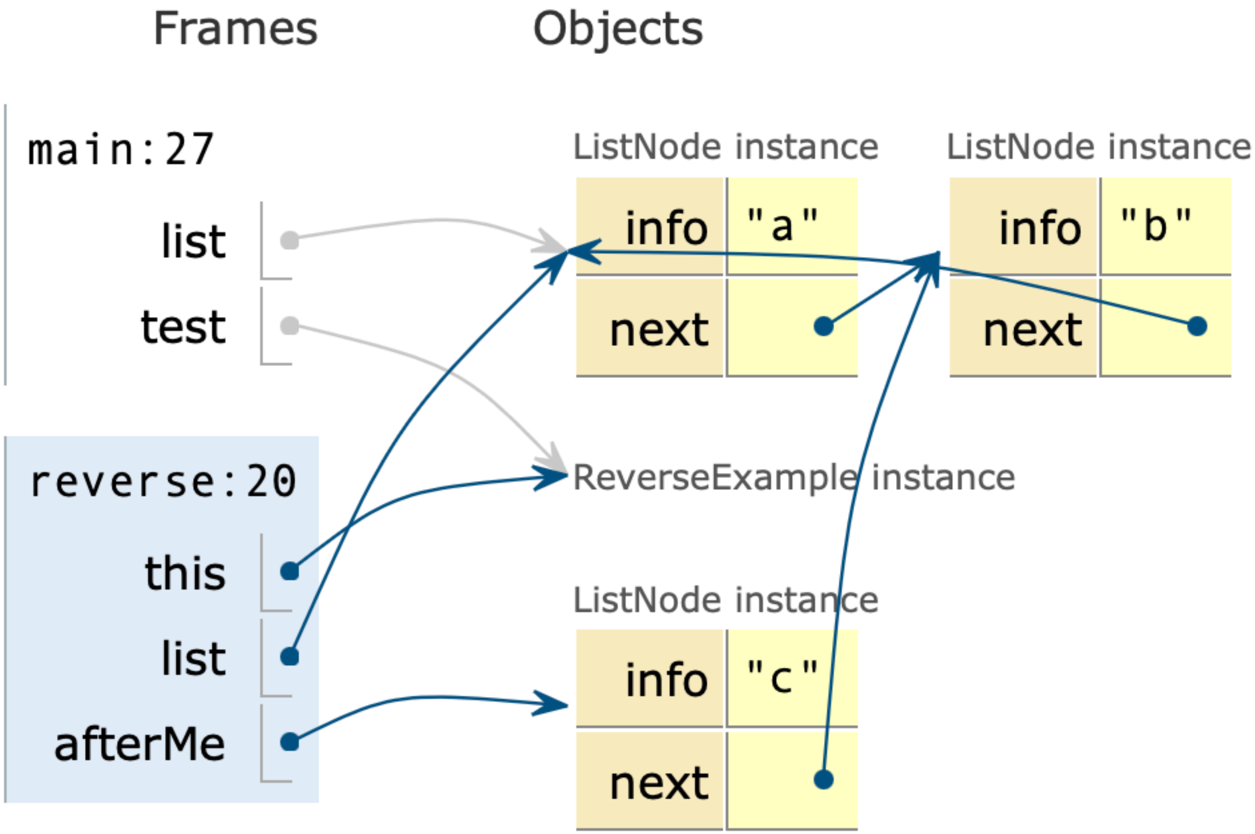

Frames
Objects
main:27
list
test
reverse:18
this
list
reverse:18
this
list
reverse:15
this
list
ListNode instance
info "a"
next
ListNode instance
info "b"
next
ListNode instance
info "c"
next null
ReverseExample instance

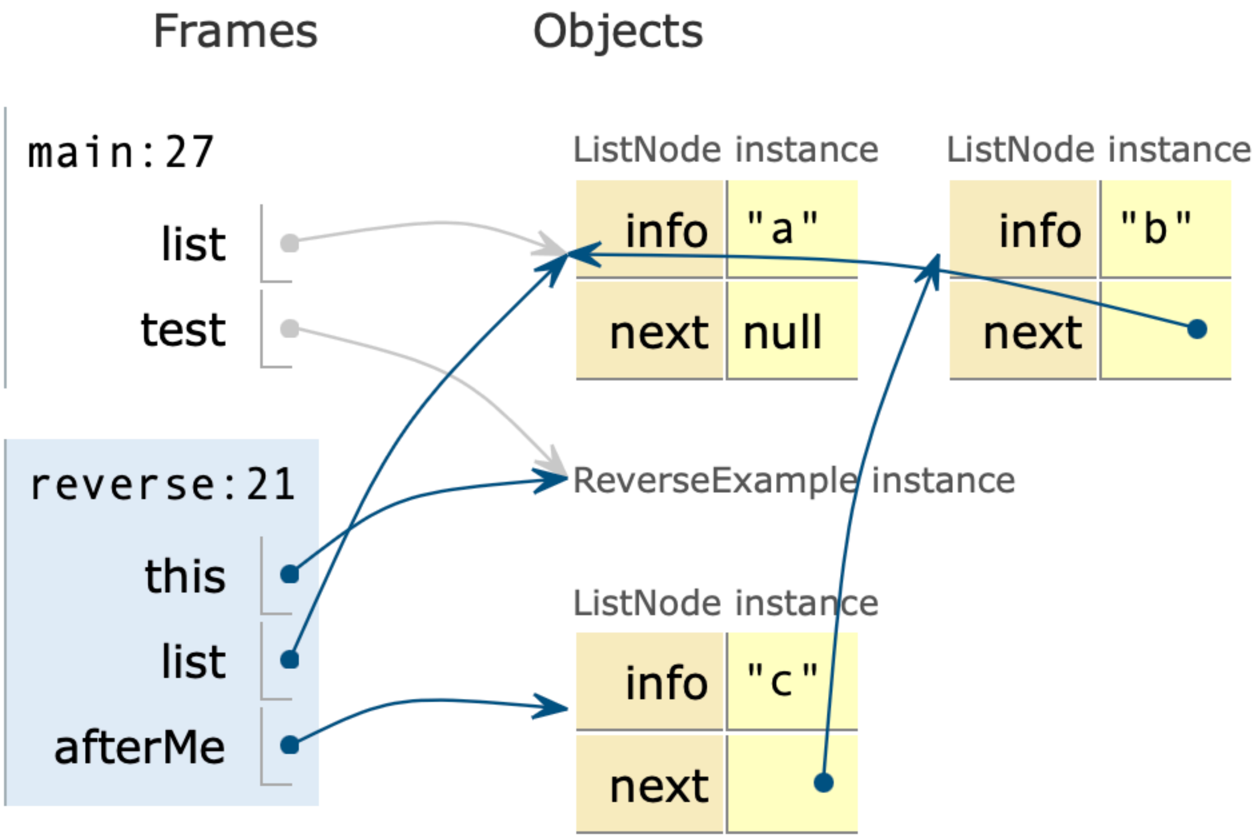

main:27
list
test
reverse:18
this
list
reverse:19
this
list
afterMe
ListNode instance
info
"a"
next
ListNode instance
info
"b"
next
ListNode instance
info
"c"
next
null
ReverseExample instance

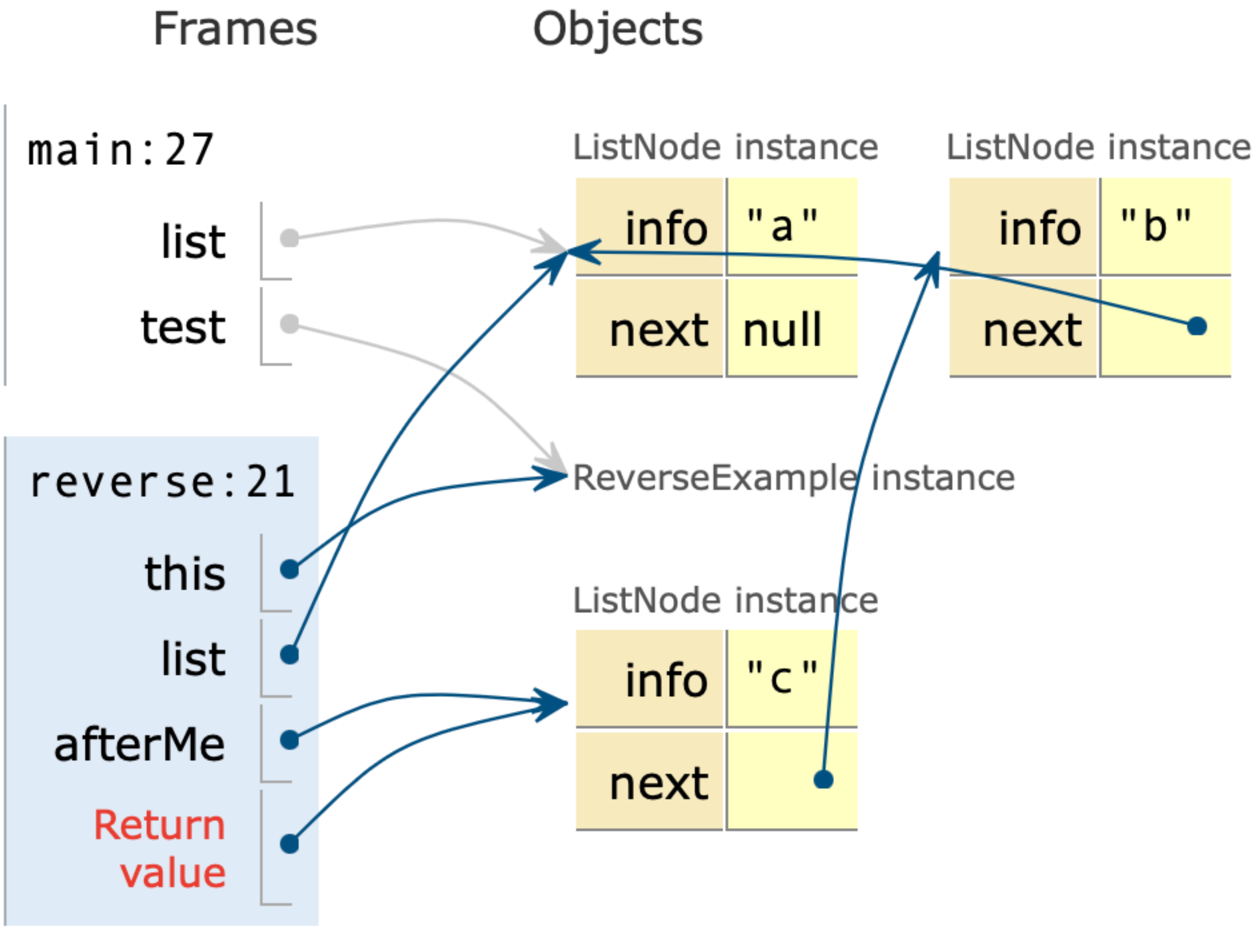

Frames
Objects
main:27
list
test
reverse:18
this
list
reverse:20
this
list
afterMe
ListNode instance
info "a"
next
ListNode instance
info "b"
next
ListNode instance
info "c"
next
ReverseExample instance


Frames
Objects
main:27
list
test
reverse:18
this
list
reverse:21
this
list
afterMe
ListNode instance
info "a"
next
ListNode instance
info "b"
next null
ListNode instance
info "c"
next
ReverseExample instance


Frames
Objects
main:27
list
test
reverse:19
this
list
afterMe
ListNode instance
info "a"
next
ListNode instance
info "b"
next null
ReverseExample instance
ListNode instance
info "c"
next


Frames
Objects
main:27
list
test
reverse:20
this
list
afterMe
ListNode instance
info
"a"
next
ListNode instance
info
"b"
next
ReverseExample instance
ListNode instance
info
"c"
next


Frames
Objects
main:27
list
test
reverse:21
this
list
afterMe
ListNode instance
info
"a"
next
null
ListNode instance
info
"b"
next
ReverseExample instance
ListNode instance
info
"c"
next


Frames
Objects
main:27
list
test
reverse:21
this
list
afterMe
Return value
ListNode instance
info
"a"
next
null
ListNode instance
info
"b"
next
ReverseExample instance
ListNode instance
info
"c"
next

Analyzing Recursive Runtime

Develop a recurrence relation of the form

$$T(N) = a \cdot T\big(g(N)\big) + f(N)$$

Total runtime

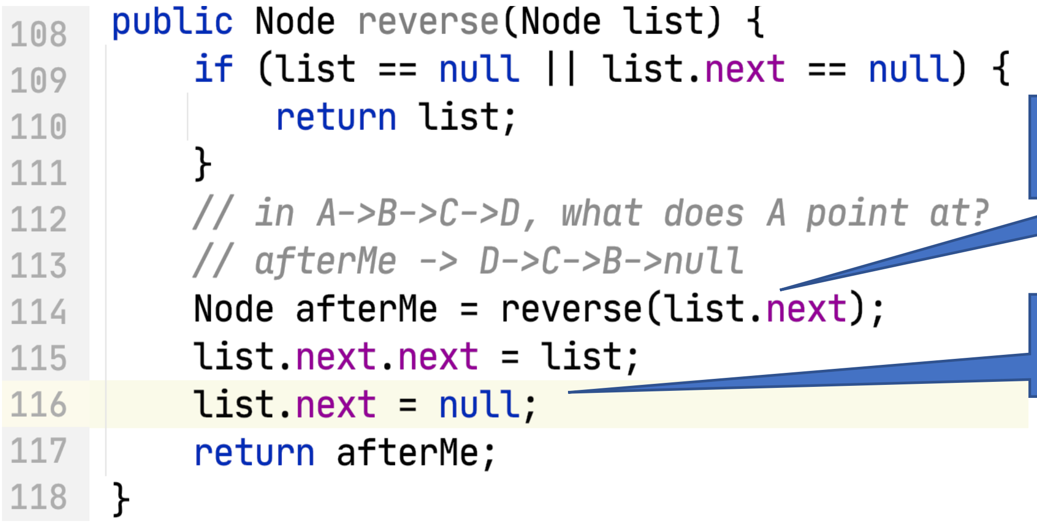
Recursive call(s)

Non-recursive runtime

Where:

- $T(N)$ - runtime of method with input size *N*
- a is the number of recursive calls
- $g(N)$ - how much input size decreases on each recursive call
- $f(N)$ - runtime of non-recursive code on input size N

Analyzing Runtime of Recursive Reverse



```java
public Node reverse(Node list) {
    if (list == null || list.next == null) {
        return list;
    }
    // in A->B->C->D, what does A point at?
    // afterMe -> D->C->B->null
    Node afterMe = reverse(list.next);
    list.next.next = list;
    list.next = null;
    return afterMe;
}
```

$g(N) = N - 1$

$f(N) = O(1)$

$$T(N)$$
$$= T(N-1) + O(1)$$

Solving Recurrence Relation

$$
\begin{aligned}
T(N) &= T(N-1) + O(1) \\
&= \big(T(N-2) + O(1)\big) + O(1) \\
&= (T(N-3) + 3 \cdot O(1) \\
&\vdots \\
&= T(1) + N \cdot O(1) \\
&= O(N)
\end{aligned}
$$

Apply recurrence again to T(N-1)

Total runtime

T(1) is base case, just O(1)

recurrence relations and expectations in 201

- In general, will **not** be asked to solve recurrent relations on exams (for later classes in theory).
- You **will** be asked to determine the recurrence relation of a given algorithm/code.

Recurrence	Algorithm	Solution
$T(n) = T(n/2) + O(1)$	binary search	$O(\log n)$
$T(n) = T(n-1) + O(1)$	sequential search	$O(n)$
$T(n) = 2T(n/2) + O(1)$	tree traversal	$O(n)$
$T(n) = T(n/2) + O(n)$	qsort partition ,find kth	$O(n)$
$T(n) = 2T(n/2) + O(n)$	mergesort, quicksort	$O(n \log n)$
$T(n) = T(n-1) + O(n)$	selection or bubble sort	$O(n^2)$

WOTO

Go to duke.is/by8a9

Not graded for correctness, just participation.

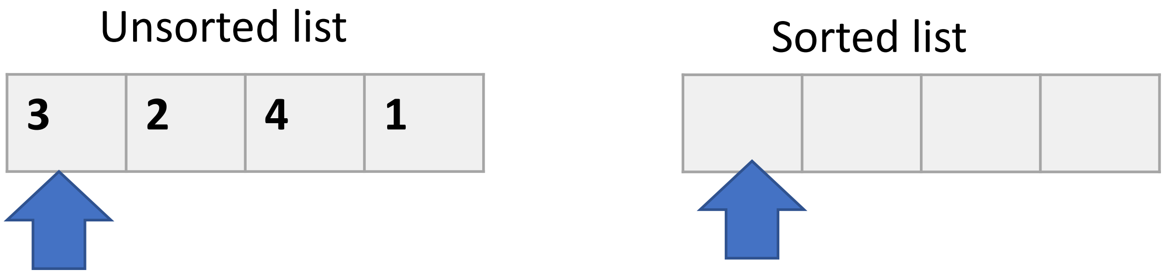
Try to answer *without* looking back at slides and notes.

But do talk to your neighbors!

Recursive Sorting: Mergesort

A simple insertion sort without recursion

- Loop through original unsorted list.
- Maintain a sorted list so far, add one value at a time. Insert into the correct position in sorted list.



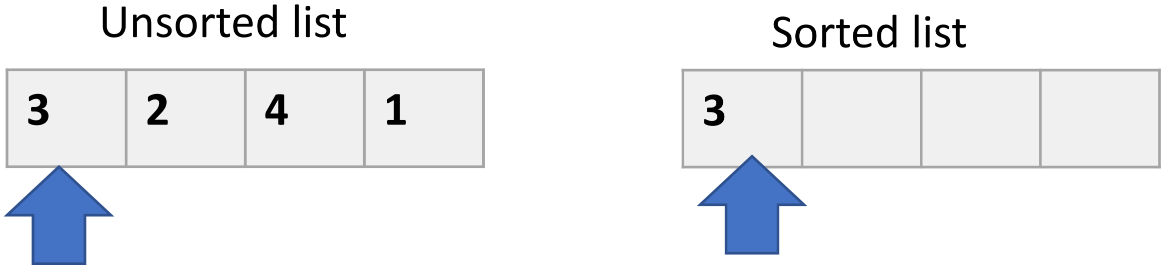
Unsorted list

3	2	4	1

Sorted list

A simple insertion sort without recursion

- Loop through original unsorted list.
- Maintain a sorted list so far, add one value at a time. Insert into the correct position in sorted list.



Unsorted list

3	**2**	**4**	**1**

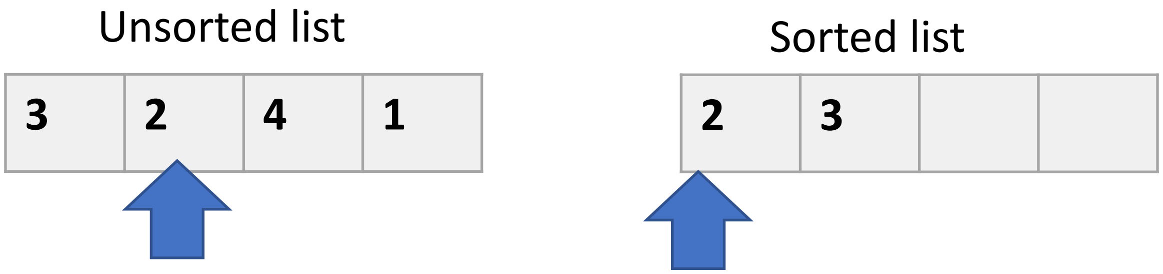
Sorted list

3			

- First value to insert, nothing to compare with.

A simple insertion sort without recursion

- Loop through original unsorted list.
- Maintain a sorted list so far, add one value at a time. Insert into the correct position in sorted list.



Unsorted list

3	2	4	1

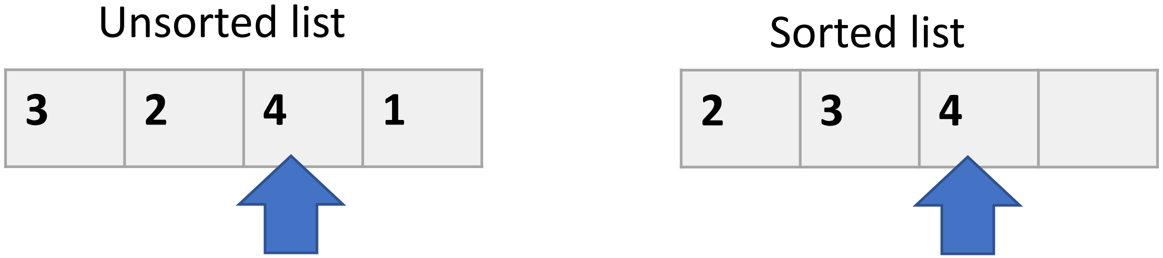
Sorted list

2	3		

- Insert 2 before 3

A simple insertion sort without recursion

- Loop through original unsorted list.
- Maintain a sorted list so far, add one value at a time. Insert into the correct position in sorted list.



Unsorted list

3	2	4	1

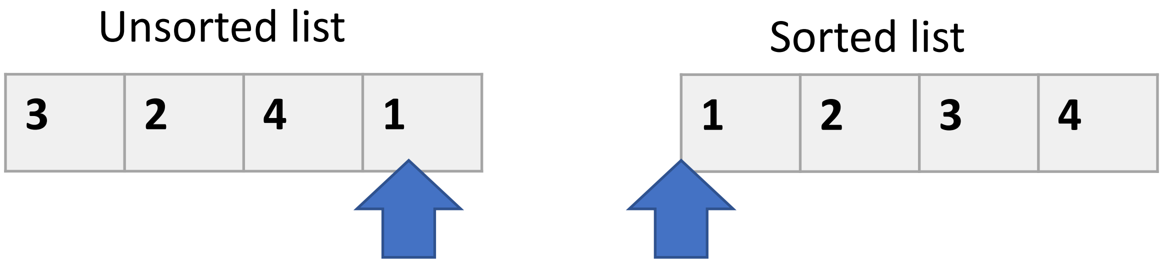
Sorted list

2	3	4	

- Insert 4 after 2 and 3

A simple insertion sort without recursion

- Loop through original unsorted list.
- Maintain a sorted list so far, add one value at a time. Insert into the correct position in sorted list.



Unsorted list

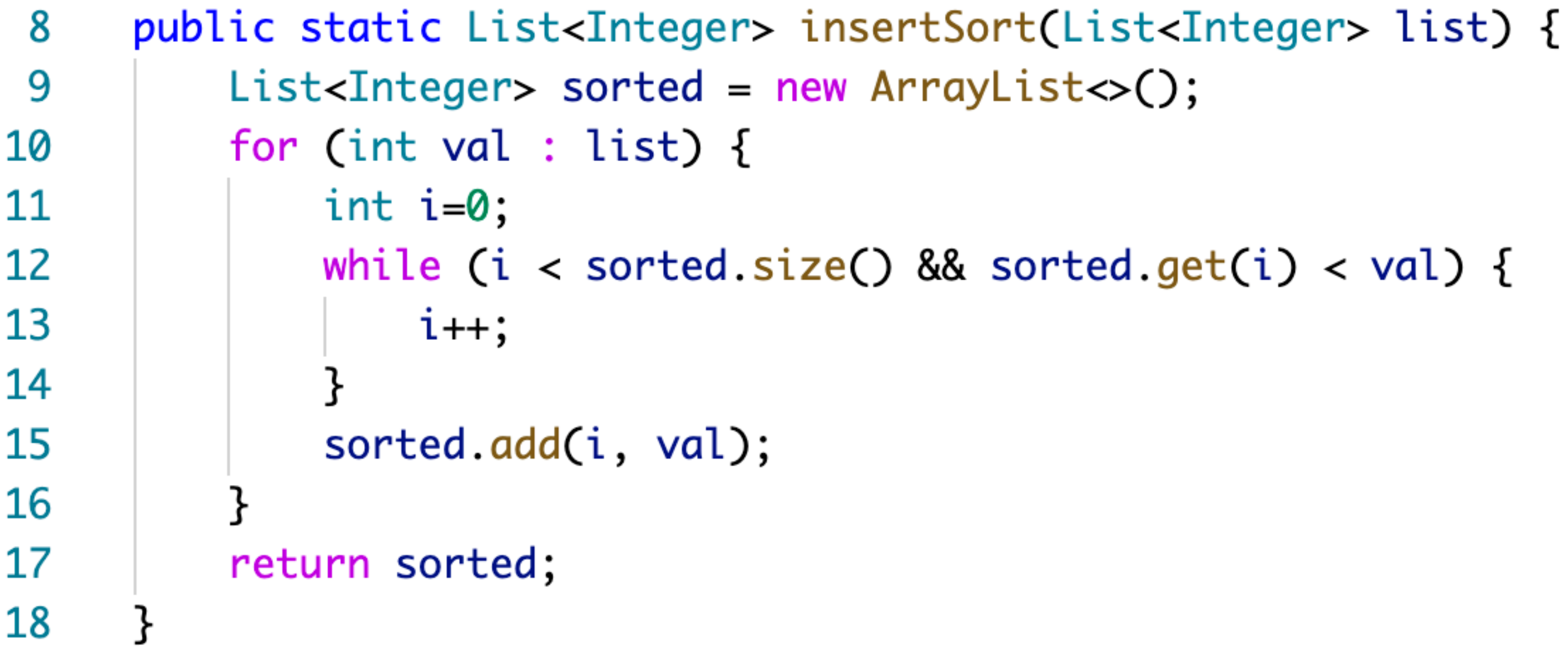
3	2	4	1

Sorted list

1	2	3	4

- Insert 1 at front

Simple insertion sort code



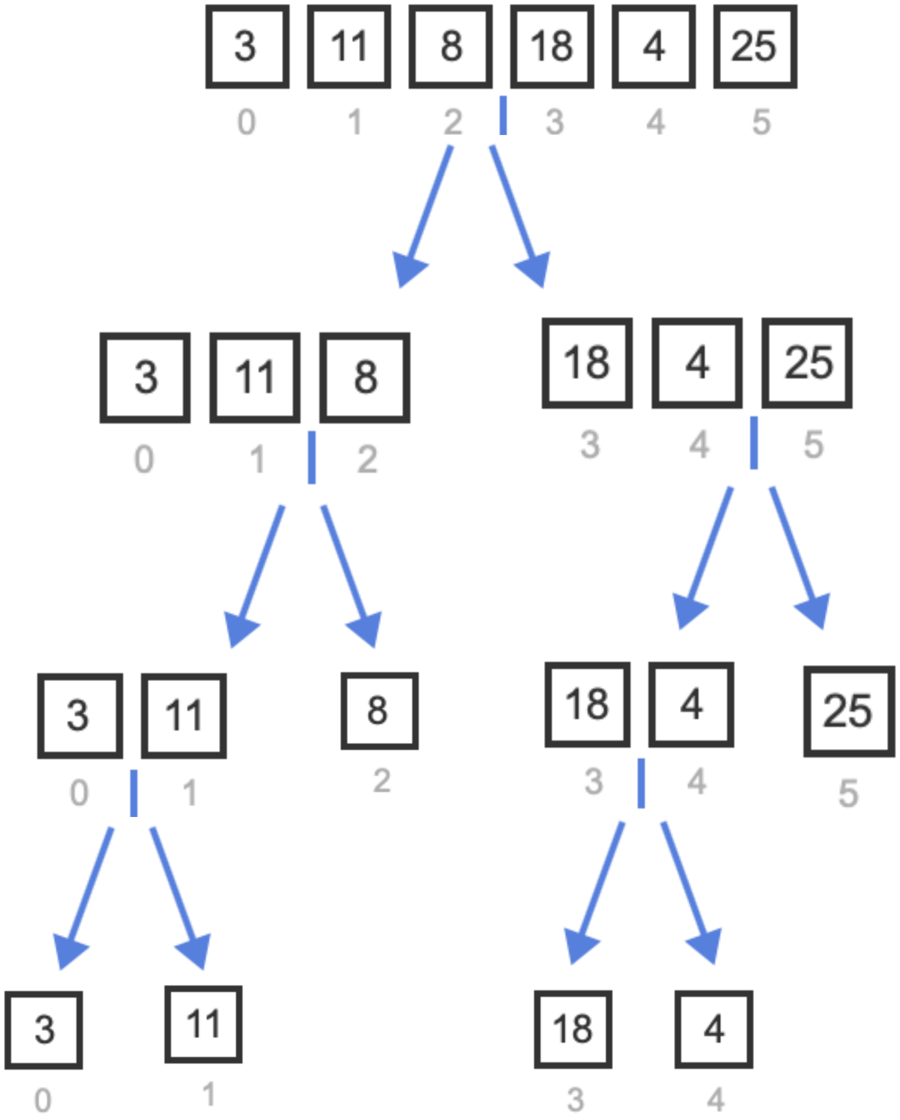
```java
public static List<Integer> insertSort(List<Integer> list) {
    List<Integer> sorted = new ArrayList<>();
    for (int val : list) {
        int i=0;
        while (i < sorted.size() && sorted.get(i) < val) {
            i++;
        }
        sorted.add(i, val);
    }
    return sorted;
}
```

- Unlike Collections.sort, creates new list, does not mutate input list.
- Runtime complexity? O(N^2). Anything faster?

Mergesort

High level idea:

- Base case: size 1
  - Return list
- Recursive case:
  - Mergesort(first half)
  - Mergesort(second half)
  - …



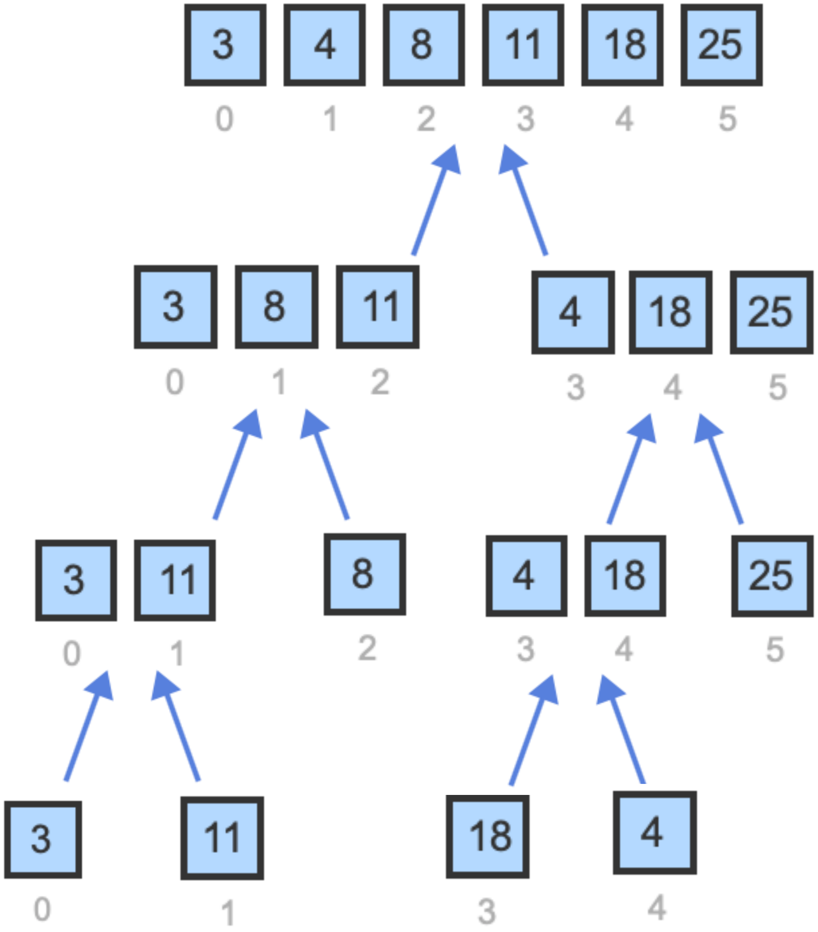
Zybook 18.4

Mergesort

High level idea:

- Base case: size 1
  - Return list
- Recursive case:
  - Mergesort(first half)
  - Mergesort(second half)
  - Merge the sorted halves
  - Return sorted

Helper method



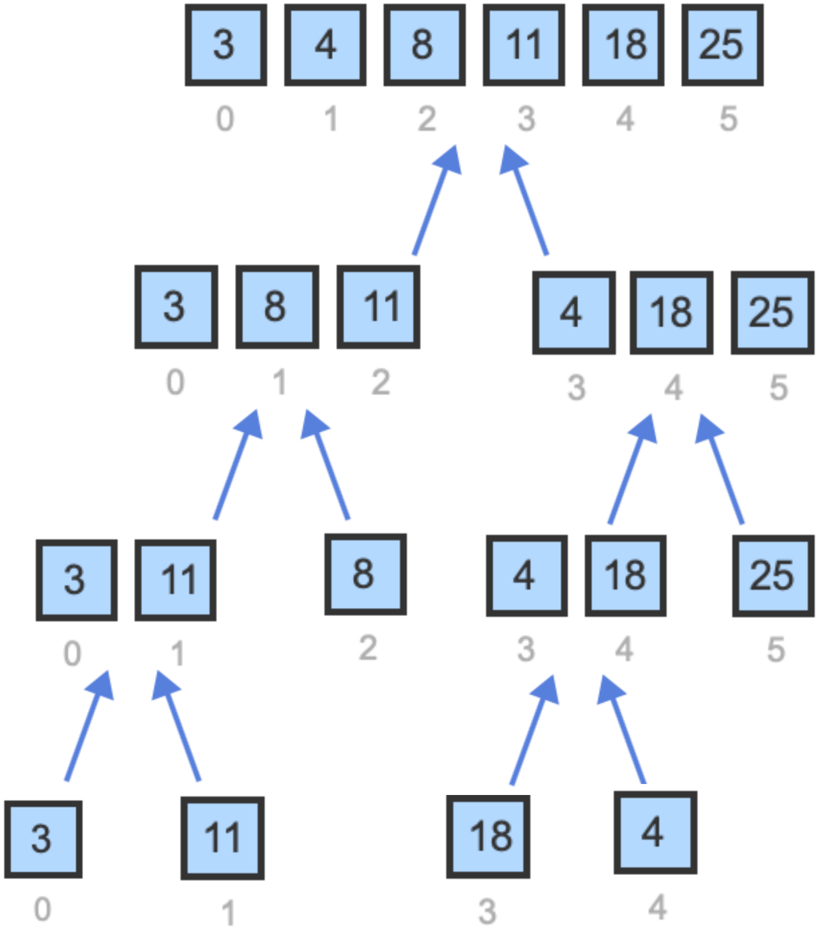
Zybook 18.4

Why mergesort is O(Nlog(N)), intuition

- Halves at each level, so just O(log(N)) levels.

- If we can do all of the merges at each level in O(N) time?

- Overall O(Nlog(N)).



Zybook 18.4

Comparing O(N^2) and O(Nlog(N)) sorts empirically

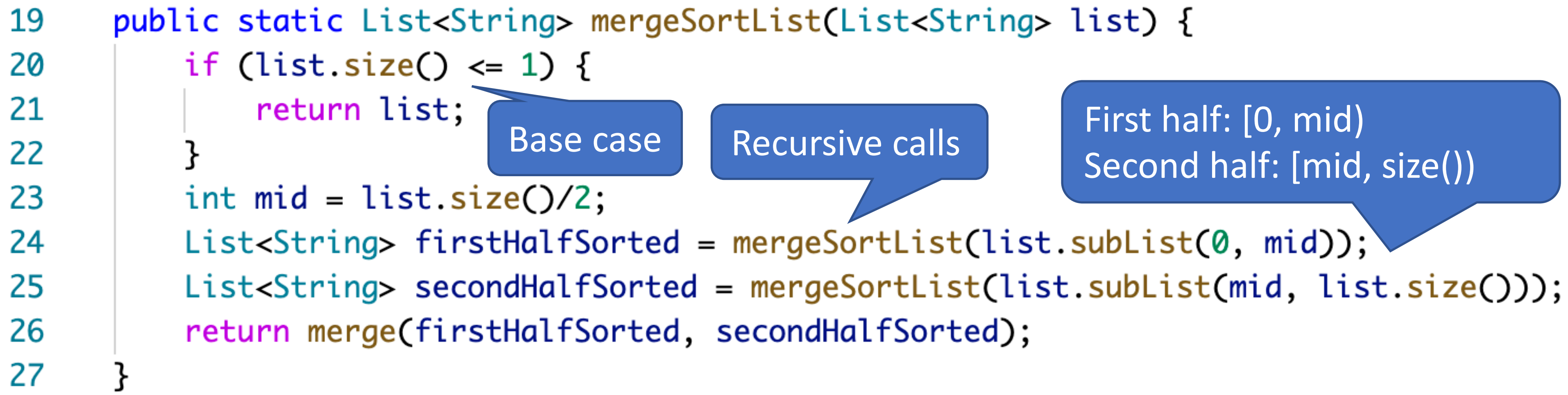
N, number of Strings sorting	insertionSort, O(N^2), in ms	Mergesort O(Nlog(N)), in ms
10,000	54	6
20,000	196	13
40,000	783	24
80,000	3040	51

O(Nlog(N)) sometimes referred to as *nearly linear*.

Double N and it doubles runtime “plus a little”

Mergesort in Code

- Written to sort Strings instead of integers.



```java
public static List<String> mergeSortList(List<String> list) {
    if (list.size() <= 1) {
        return list;
    }
    int mid = list.size()/2;
    List<String> firstHalfSorted = mergeSortList(list.subList(0, mid));
    List<String> secondHalfSorted = mergeSortList(list.subList(mid, list.size()));
    return merge(firstHalfSorted, secondHalfSorted);
}
```

Base case

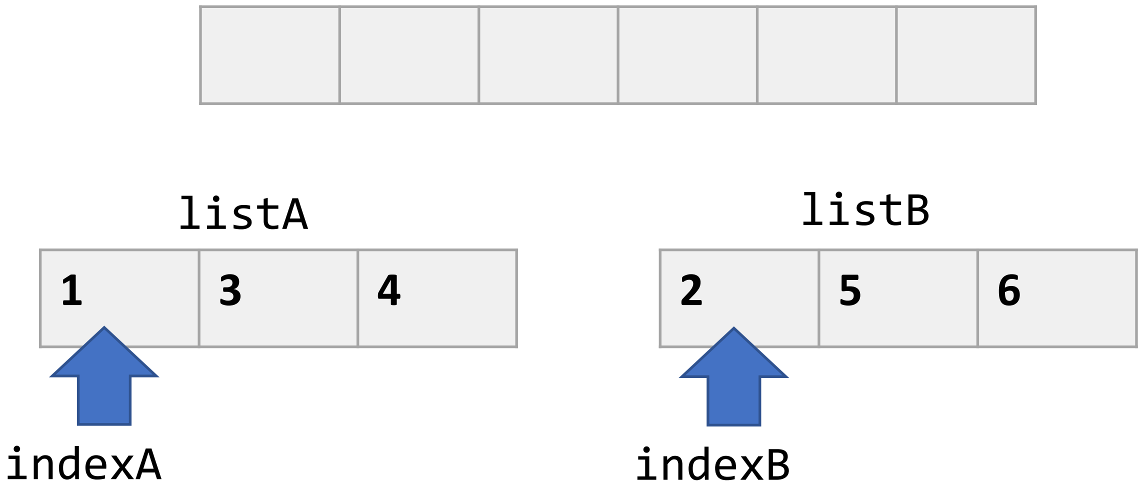
Recursive calls

First half: [0, mid)
Second half: [mid, size())

- Unlike Collections.sort, this implementation returns a new sorted list, does not mutate the input.
- Where are new lists created? merge helper method.

Merge method

- Given two sorted lists, `listA` and `listB`, want to return a new sorted list with all values from both.
- Need to keep track of **two** indices, `indexA` in `listA` and `indexB` in `listB`.



listA

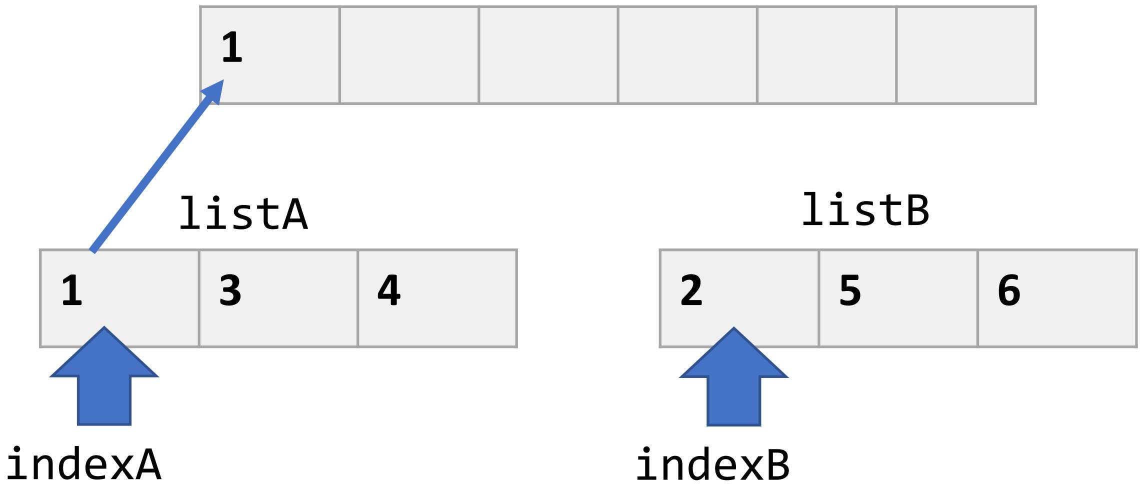
1	3	4

listB

2	5	6

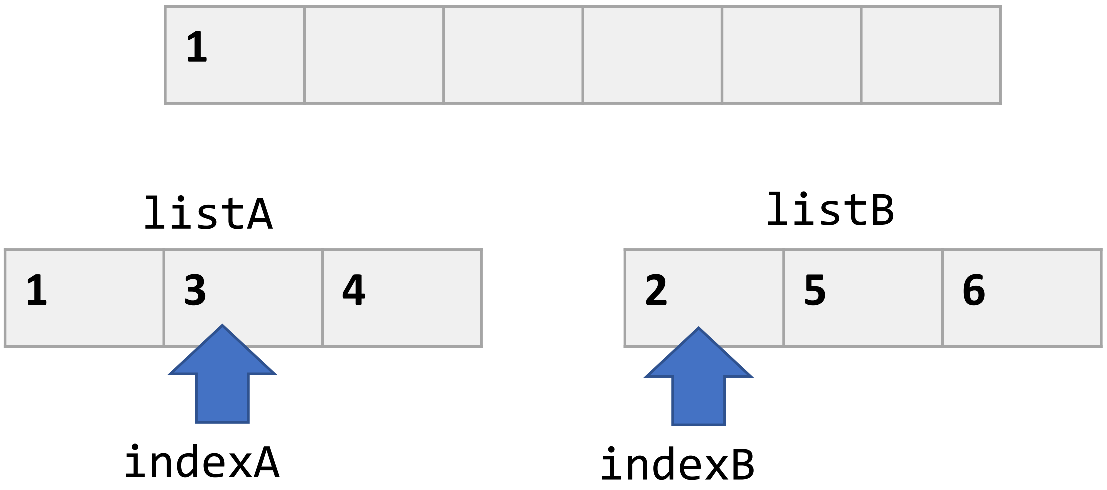
Merge method

- Given two sorted lists, `listA` and `listB`, want to return a new sorted list with all values from both.
- Need to keep track of **two** indices, `indexA` in `listA` and `indexB` in `listB`.



Merge method

- Given two sorted lists, `listA` and `listB`, want to return a new sorted list with all values from both.
- Need to keep track of **two** indices, `indexA` in `listA` and `indexB` in `listB`.



1					

listA

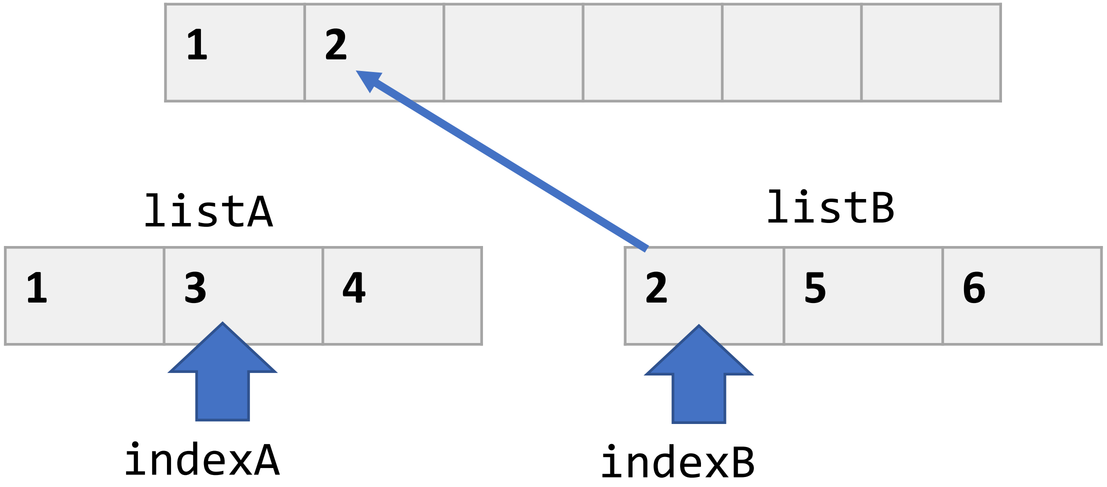
1	3	4

listB

2	5	6

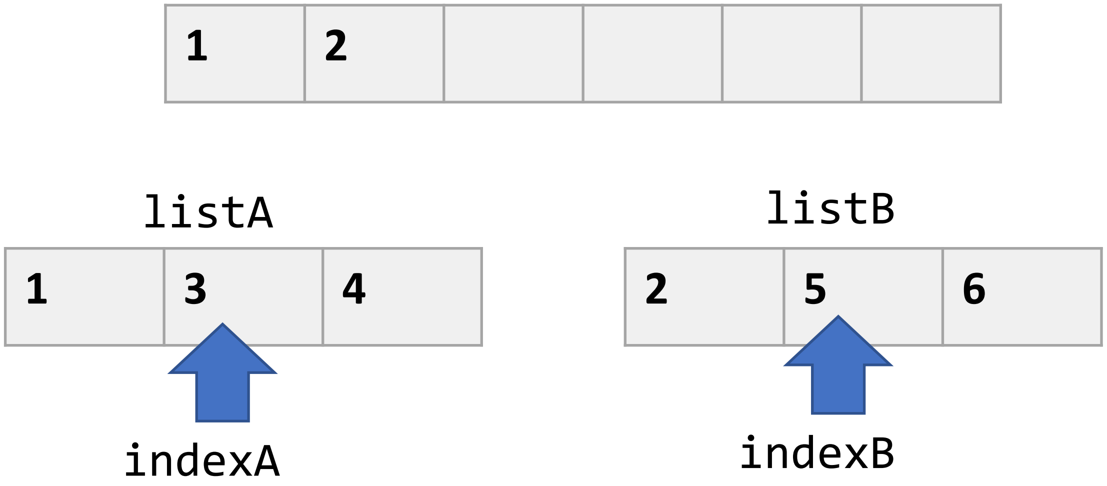
Merge method

- Given two sorted lists, `listA` and `listB`, want to return a new sorted list with all values from both.
- Need to keep track of **two** indices, `indexA` in `listA` and `indexB` in `listB`.



Merge method

- Given two sorted lists, `listA` and `listB`, want to return a new sorted list with all values from both.
- Need to keep track of **two** indices, `indexA` in `listA` and `indexB` in `listB`.



1	2				

listA

1	3	4

listB

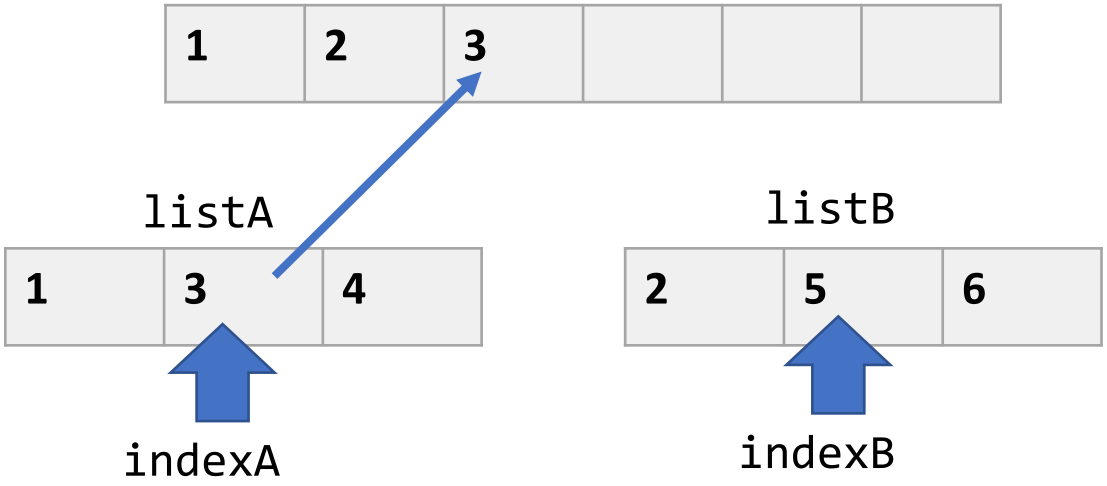
2	5	6

indexA

indexB

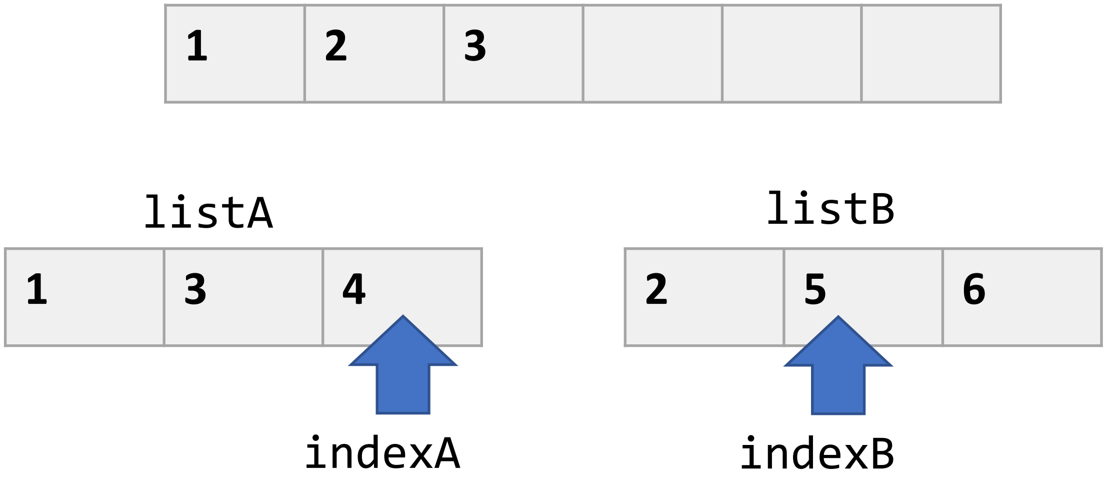
Merge method

- Given two sorted lists, `listA` and `listB`, want to return a new sorted list with all values from both.
- Need to keep track of **two** indices, `indexA` in `listA` and `indexB` in `listB`.



Merge method

- Given two sorted lists, `listA` and `listB`, want to return a new sorted list with all values from both.
- Need to keep track of **two** indices, `indexA` in `listA` and `indexB` in `listB`.



1	2	3			

listA

1	3	4

indexA

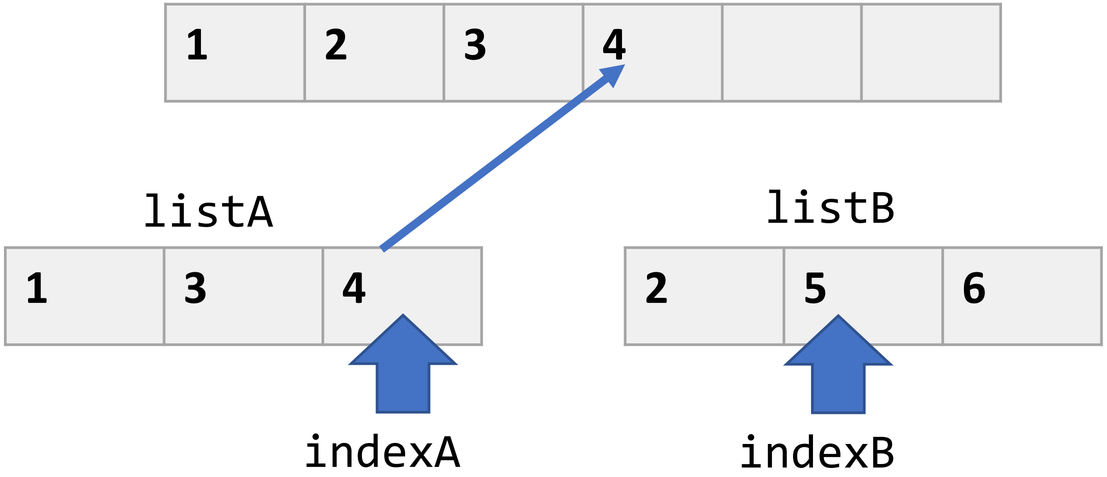
listB

2	5	6

indexB

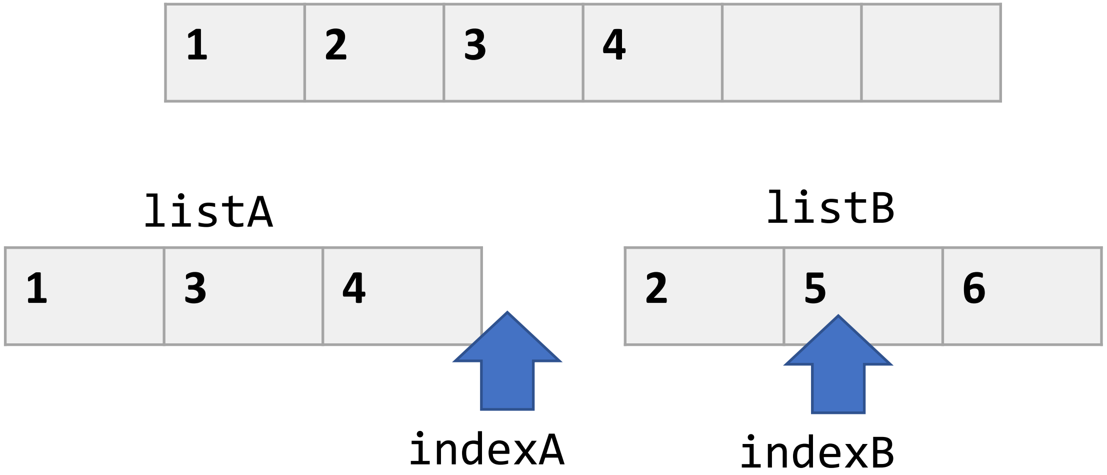
Merge method

- Given two sorted lists, `listA` and `listB`, want to return a new sorted list with all values from both.
- Need to keep track of **two** indices, `indexA` in `listA` and `indexB` in `listB`.



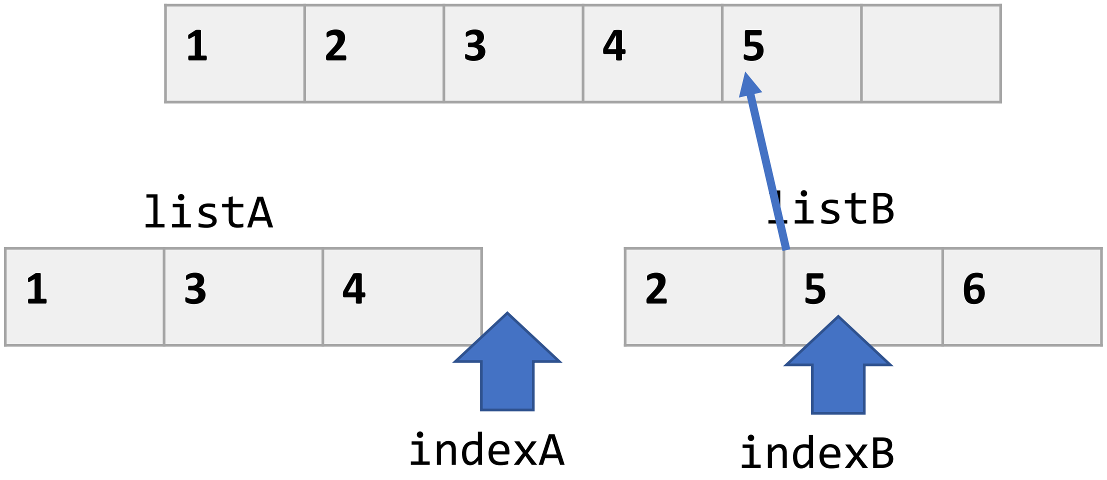
Merge method

- Given two sorted lists, `listA` and `listB`, want to return a new sorted list with all values from both.
- Need to keep track of **two** indices, `indexA` in `listA` and `indexB` in `listB`.



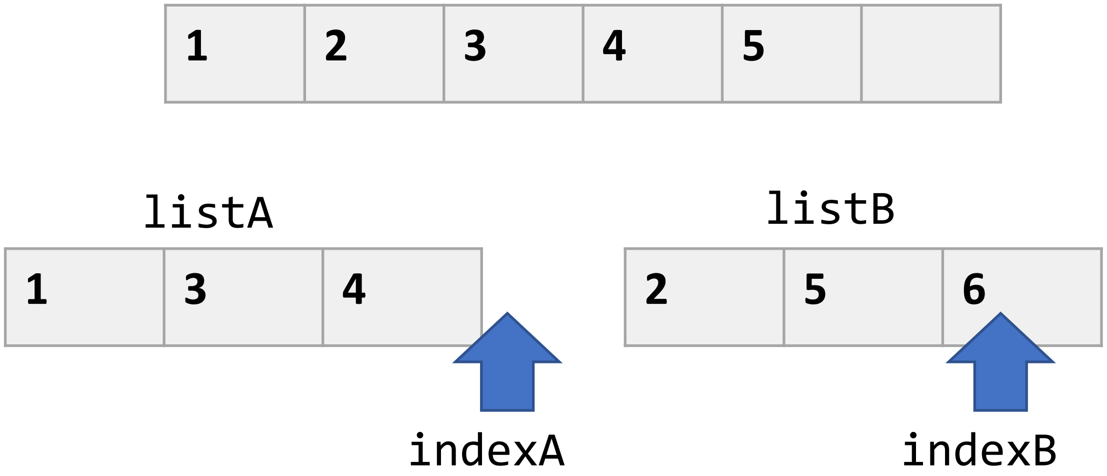
Merge method

- Given two sorted lists, `listA` and `listB`, want to return a new sorted list with all values from both.
- Need to keep track of **two** indices, `indexA` in `listA` and `indexB` in `listB`.



Merge method

- Given two sorted lists, `listA` and `listB`, want to return a new sorted list with all values from both.
- Need to keep track of **two** indices, `indexA` in `listA` and `indexB` in `listB`.



1	2	3	4	5	

listA

1	3	4

listB

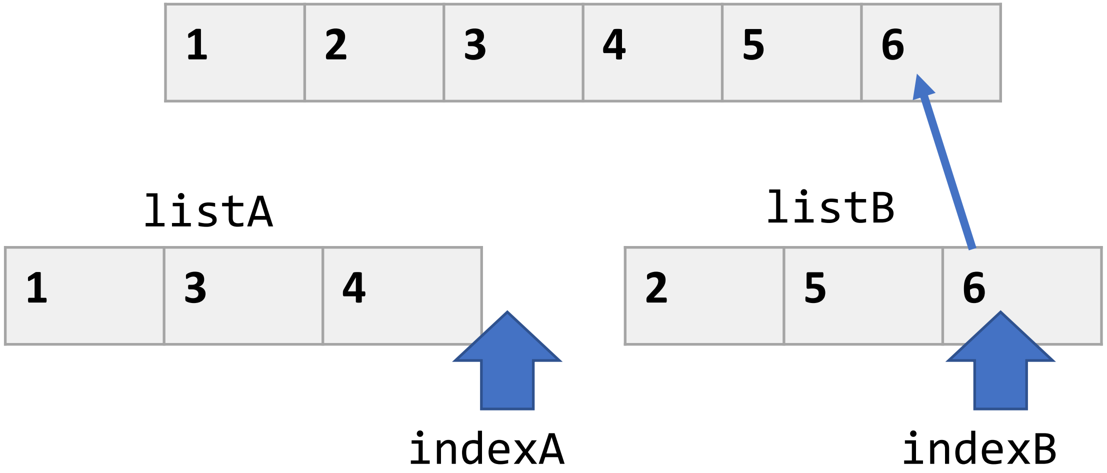
2	5	6

indexA

indexB

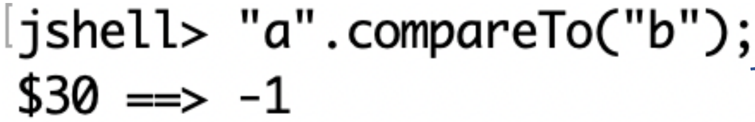
Merge method

- Given two sorted lists, `listA` and `listB`, want to return a new sorted list with all values from both.
- Need to keep track of **two** indices, `indexA` in `listA` and `indexB` in `listB`.



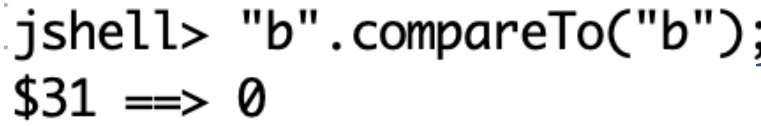
How to compare Strings?

- What is the equivalent of < for Strings?
- Use the `compareTo` method for the natural lexicographic (dictionary/sorted) ordering.



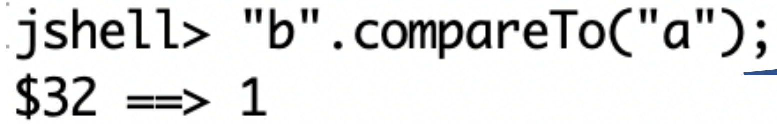
```
jshell> "a".compareTo("b");
$30 ==> -1
```

Negative for “less than”



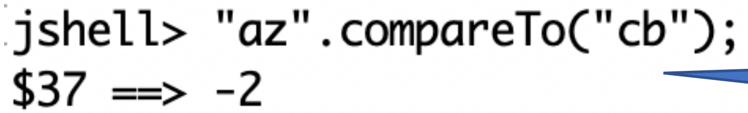
```
jshell> "b".compareTo("b");
$31 ==> 0
```

Zero for “equal”



```
jshell> "b".compareTo("a");
$32 ==> 1
```

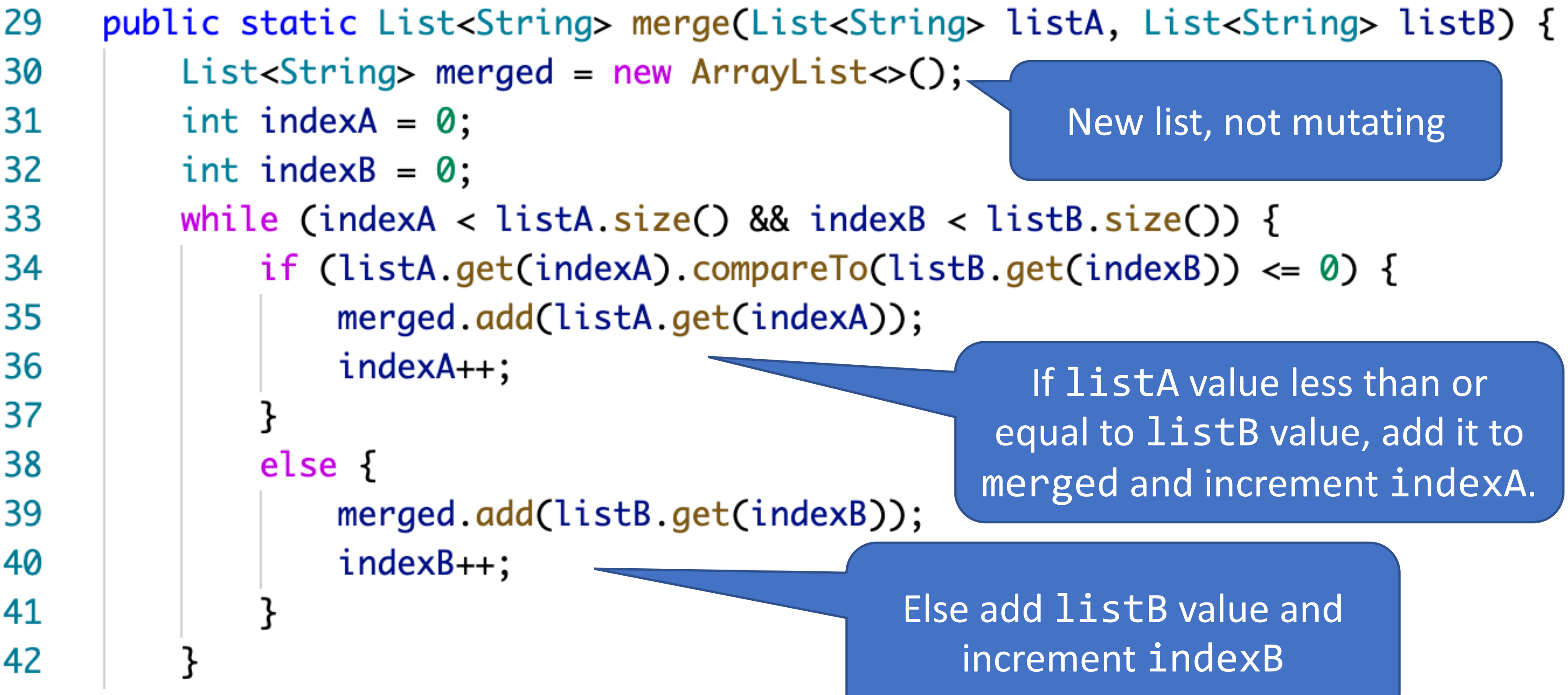
Positive for “greater than”



```
jshell> "az".compareTo("cb");
$37 ==> -2
```

Lexicographic, check first character, second if equal, third if still equal, …

Merge method in code



```java
public static List<String> merge(List<String> listA, List<String> listB) {
    List<String> merged = new ArrayList<>();
    int indexA = 0;
    int indexB = 0;
    while (indexA < listA.size() && indexB < listB.size()) {
        if (listA.get(indexA).compareTo(listB.get(indexB)) <= 0) {
            merged.add(listA.get(indexA));
            indexA++;
        }
        else {
            merged.add(listB.get(indexB));
            indexB++;
        }
    }
```

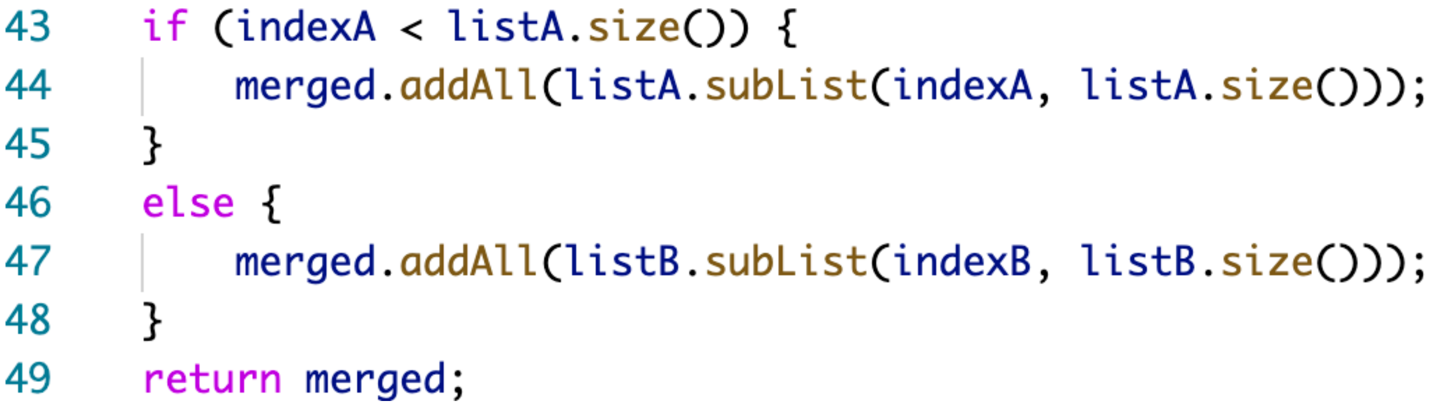
New list, not mutating

If `listA` value less than or equal to `listB` value, add it to `merged` and increment `indexA`.

Else add `listB` value and increment `indexB`

- Not recursive! Just looping.
- Code shown incomplete. How to finish?

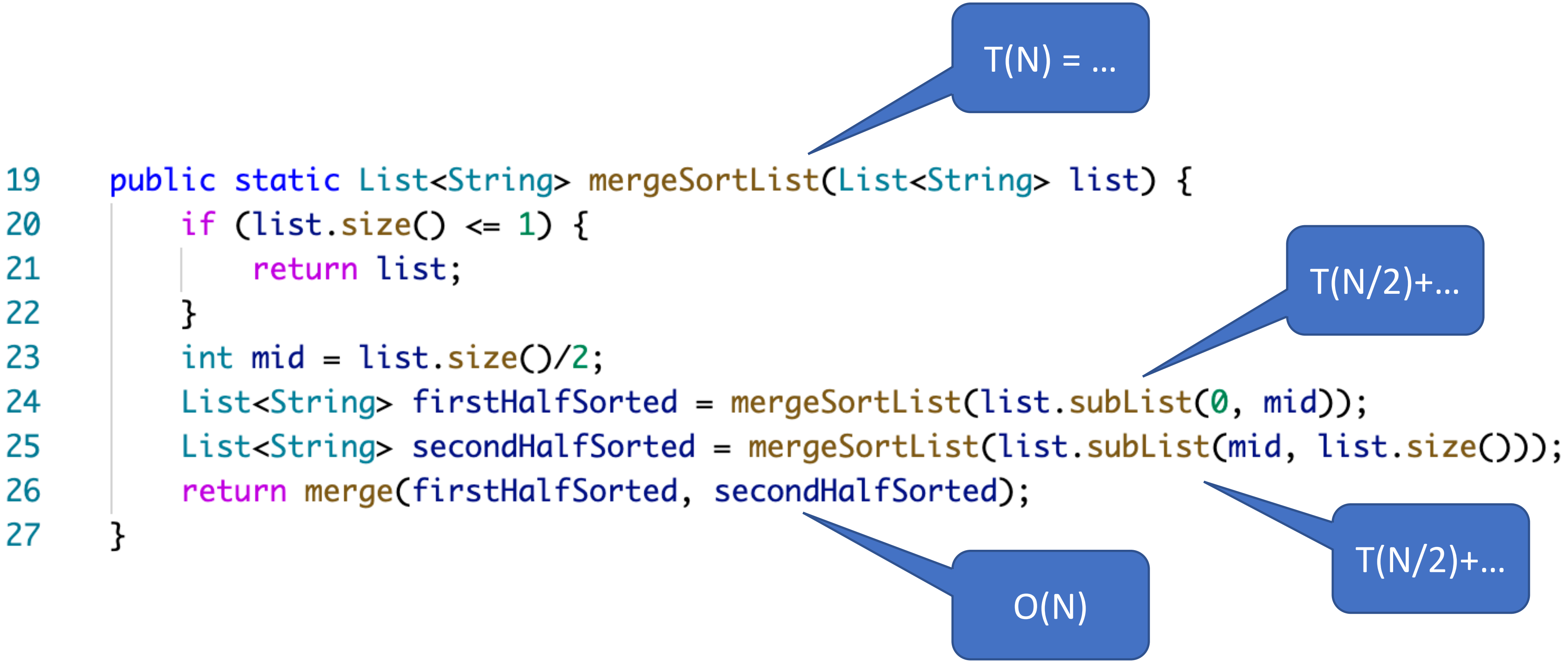
Finishing merge method



```
if (indexA < listA.size()) {
    merged.addAll(listA.subList(indexA, listA.size()));
}
else {
    merged.addAll(listB.subList(indexB, listB.size()));
}
return merged;
```

- Overall runtime complexity?
- Need to iterate through every element in `listA` and `listB`, constant time operations on each.
- If nA = `listA.size()` and nB = `listB.size()`, runtime complexity is O(nA+nB), **linear**.

Runtime complexity of mergesort?



```java
public static List<String> mergeSortList(List<String> list) {
    if (list.size() <= 1) {
        return list;
    }
    int mid = list.size()/2;
    List<String> firstHalfSorted = mergeSortList(list.subList(0, mid));
    List<String> secondHalfSorted = mergeSortList(list.subList(mid, list.size()));
    return merge(firstHalfSorted, secondHalfSorted);
}
```

T(N) = …

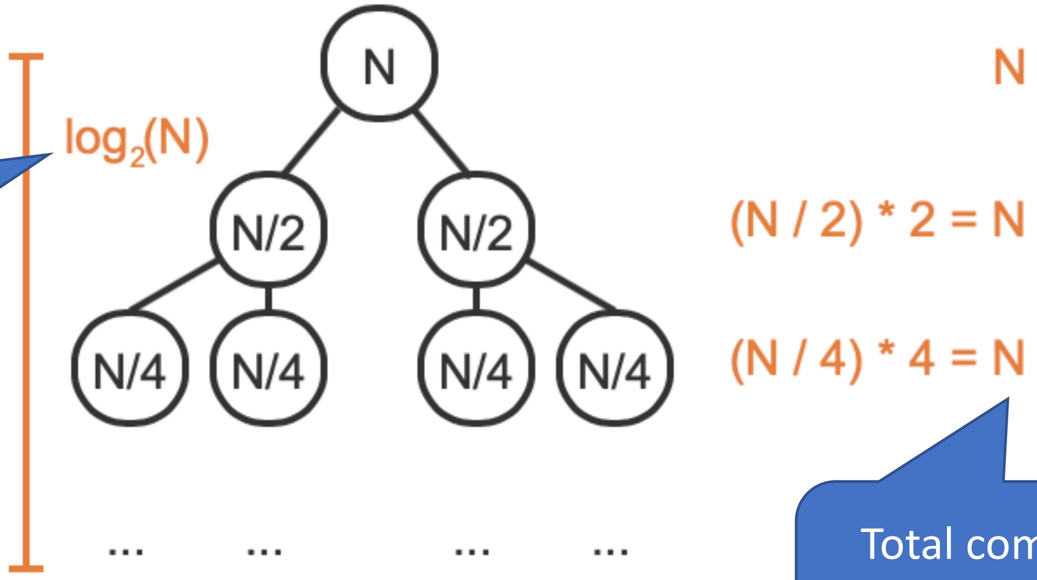
T(N/2)+…

T(N/2)+…

O(N)

Recursion tree

$$T(N) = N + T(N / 2) + T(N / 2)$$



Depth of the recursion tree: Number of recursive calls before base case.

Total complexity of each level across all of the recursive calls.

```
T(N) = O(N log N)
```

Visualization from the Zybook

Person in CS: Ellen Ochoa

- BS physics (‘75), PhD EE (‘85).
- Starting working on software for optical recognition systems in ‘80s.
- 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