L16: Queues and Binary Trees

Alex Steiger

CompSci 201: Spring 2024

3/6/2024

Announcements, Coming up

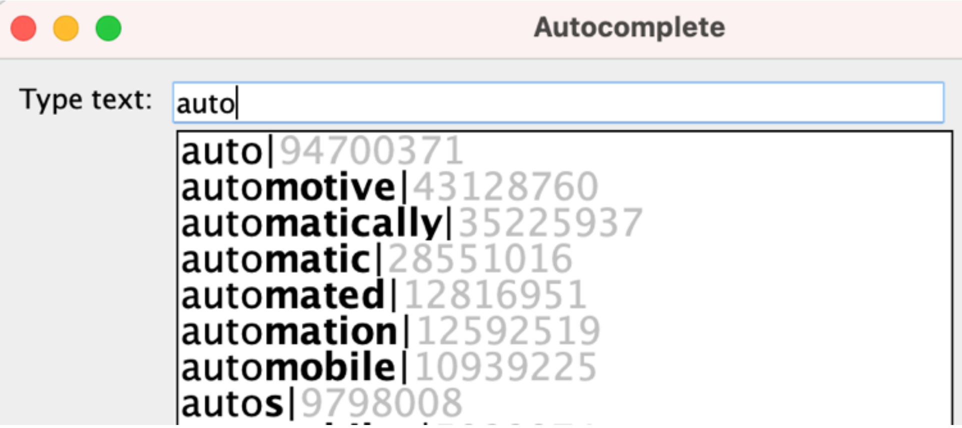
- Today, Wednesday 3/6
  - APT 6 (sorting problems) due
  - Project P4: Autocomplete released
  - APT 7 out soon, **due 3/29** (week after exam)

- Friday 3/8
  - Fill out the **midsemester course survey**
  - **No discussion, enjoy spring break!**

- Wednesday 3/20
  - Midterm 2
  - Practice exams available this evening on Canvas

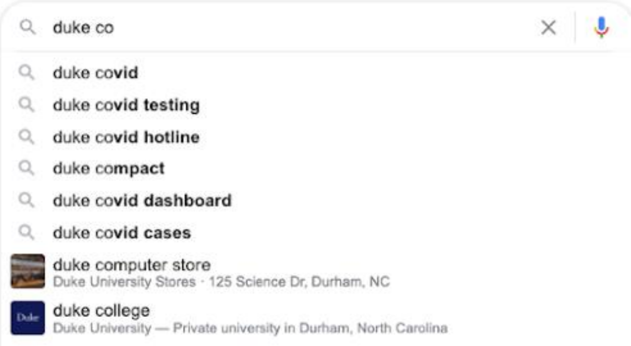
Project 4 Autocomplete

- How to create something like:



Type text: auto

```
auto|94700371
automotive|43128760
automatically|35225937
automatic|28551016
automated|12816951
automation|12592519
automobile|10939225
autos|9798008
```



duke co

- duke covid
- duke covid testing
- duke covid hotline
- duke compact
- duke covid dashboard
- duke covid cases
- duke computer store
  Duke University Stores · 125 Science Dr, Durham, NC
- duke college
  Duke University — Private university in Durham, North Carolina

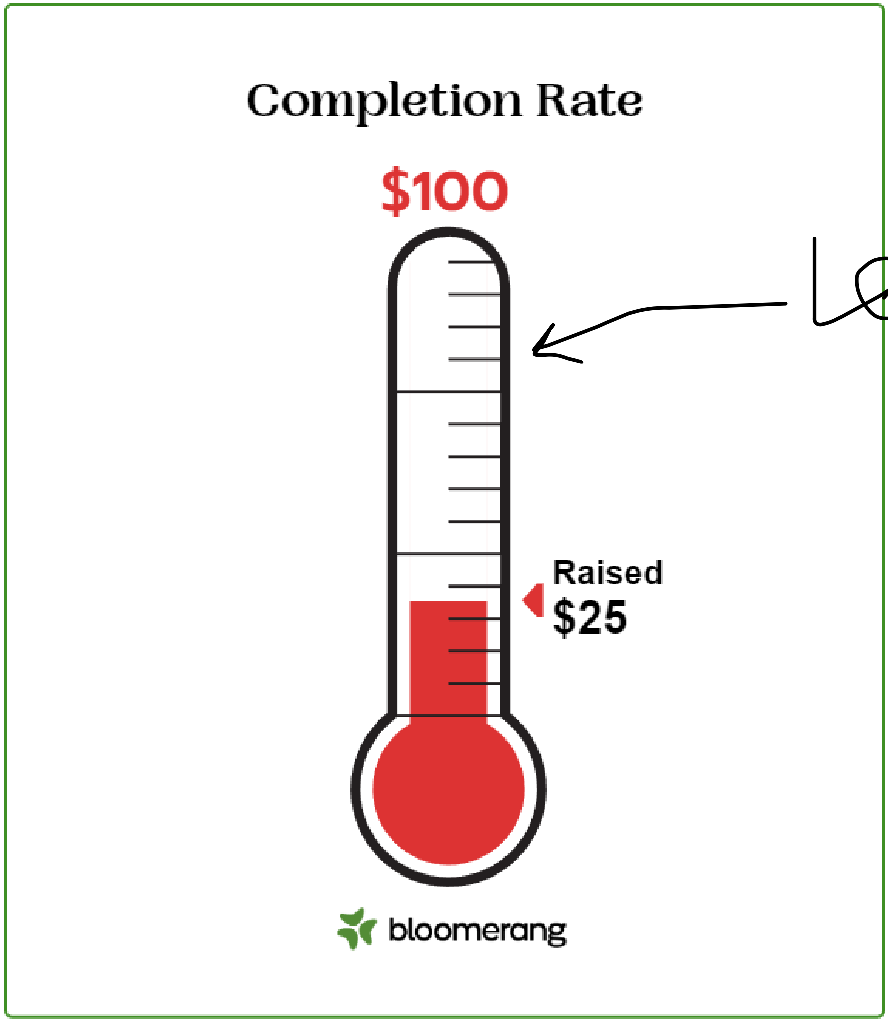
- All about two things:
  - Searching for all words that match on a prefix, and…
  - Sorting them by how common they are,
  - Return these words to show in the GUI above

Midterm 2

- 60 minutes, in-class
- Multiple choice + short answer
- 1 double-sided reference sheet (8.5”x11”)
- Extra credit if >70% midsemester survey completion rate
- Grade replaced by Final Exam Part 2

- Lectures up to **Monday + Binary Search today**
    - Stacks/queues/trees not on exam
- All projects and APTs through this week

Midsemester Survey



lets get here (and more)

Today’s Agenda

1. Binary Search

2. Stack, Queue, PriorityQueue: API perspective
   - Stack/Queue we already know how to implement
   - PriorityQueue later

3. Binary (Search) Tree

Binary Search

Binary Search

- Given a ***sorted*** **`list`** of N elements and a **`target`** value, return:
    - Index **`i`** such that **`list.get(i)`** equals **`target`**, or
    - -1 if **`target`** not in **`list`**

- Example:
    - If we search for ‘h’, should return 4
    - If we search for ‘c’, should return -1

value	‘a’	‘b’	‘d’	‘g’	‘h’	‘j’	‘k’	‘m’	‘p’
index	0	1	2	3	4	5	6	7	8

Java API Binary Search

`Arrays.binarySearch` (for arrays) and **`Collections.binarySearch`** (for Lists).

```
String[] ar = {"ape", "bird", "cat", "dog", "elephant", "ferret",
"gecko", "hippo"};
int index = Arrays.binarySearch(ar, "cat");
```

Returns 2

Careful, assumes input is sorted (and does not verify)!

```
String[] ar = {"cat", "ape", "bird",…
int index = Arrays.binarySearch(ar, "cat");
```

Returns -4

Java API Binary Search with Comparator

Can pass a comparator **`comp`**, in which case:

1. Array/List should be sorted by that **`comp`**, and
2. Want an index **`i`** with **`i`**'th element e_i has `comp.compare(`e_i`, target)==0`.

Sorted by length

[ape, cat, dog, bird, gecko, hippo, ferret, elephant]

```
Comparator<String> comp =
    Comparator.comparing(String::length);

index = Arrays.binarySearch(ar, "dog", comp);
```

Returns 1. `comp.compare (“cat”, “dog”)==0`

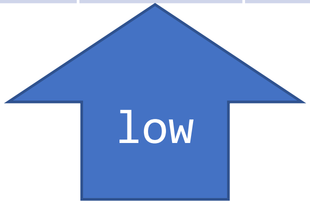
How is Binary Search O(log(N))?

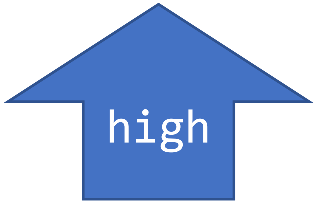
- How to find something in a list of N elements without looping over the list?
- Let `low` (initially 0) and `high` (initially N-1) mark the limits of the active search space.
- Want to cut down the search space by half at each step:

N
N/2
N/4
N/8
…
1

log_2(N) steps!

value	‘a’	‘b’	‘d’	‘g’	‘h’	‘j’	‘k’	‘m’	‘p’
index	0	1	2	3	4	5	6	7	8

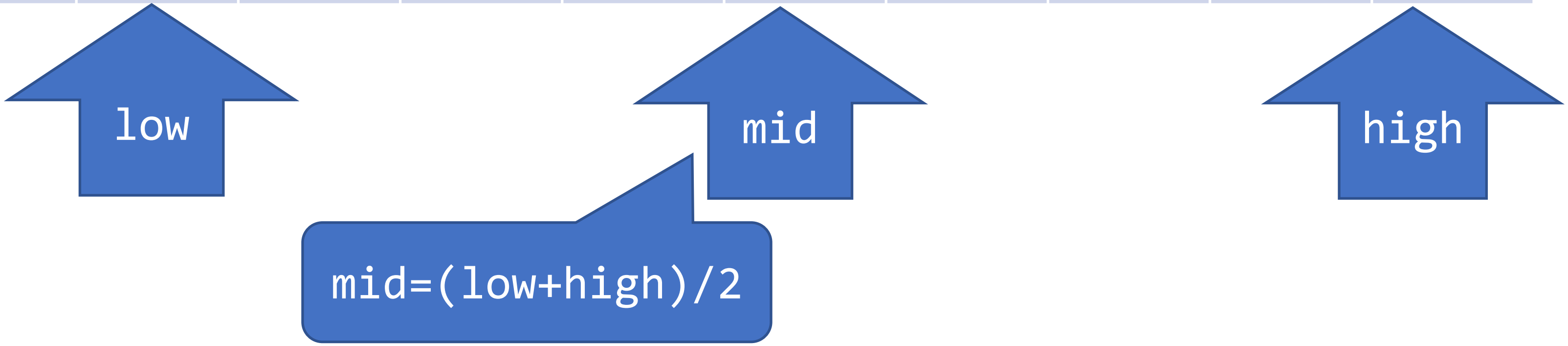




Binary Search in Pictures

- Searching for ‘d’ in

value	‘a’	‘b’	‘d’	‘g’	‘h’	‘j’	‘k’	‘m’	‘p’
index	0	1	2	3	4	5	6	7	8

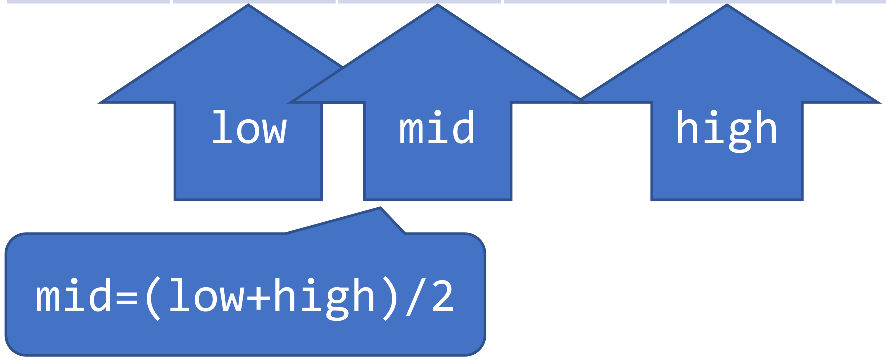


- ‘h’ > ‘d’, so need to keep searching in the *lower* half.
- Set **`high = mid-1;`**

Binary Search in Pictures

- Searching for ‘d’ in

value	‘a’	‘b’	‘d’	‘g’	‘h’	‘j’	‘k’	‘m’	‘p’
index	0	1	2	3	4	5	6	7	8



low

mid

high

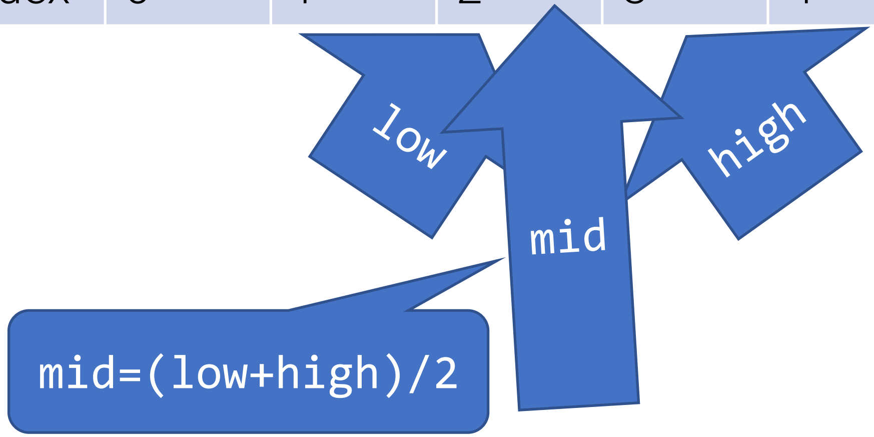
`mid=(low+high)/2`

- ‘b’ < ‘d’, so need to keep searching in the *upper* half.
- Set `low = mid+1;`

Binary Search in Pictures

- Searching for ‘d’ in

value	‘a’	‘b’	‘d’	‘g’	‘h’	‘j’	‘k’	‘m’	‘p’
index	0	1	2	3	4	5	6	7	8



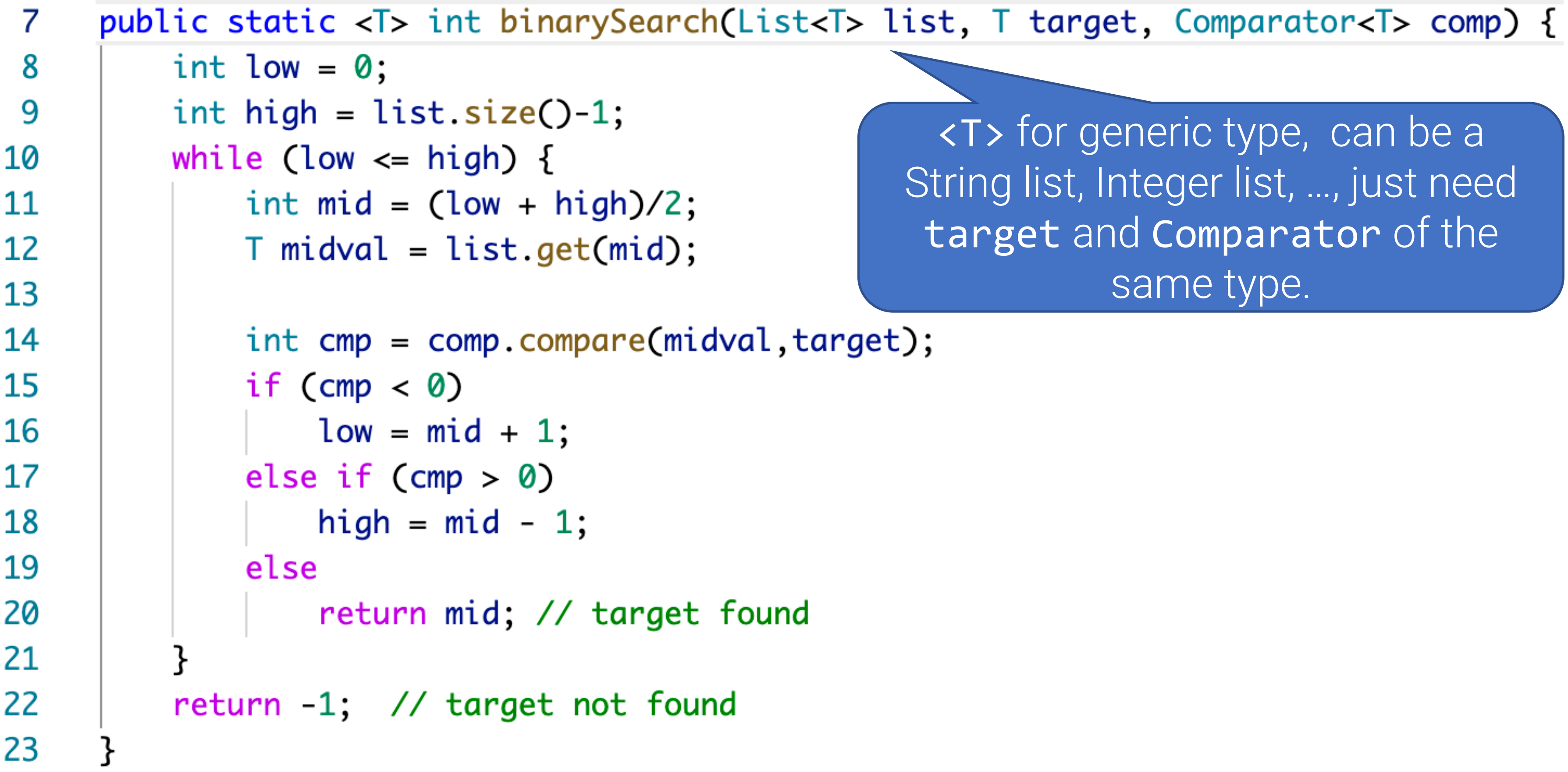
`mid=(low+high)/2`

- ‘d’ equals ‘d’, return **`mid (2)`**

Reasoning about Coding Binary Search

- Going to loop `while` (`low <= high)`
  - Looping while there is anything left to search
- For correctness, want to maintain the following **loop invariant:**
  - If the target is in the array/list, it is in the range [low, high]
- At each step, either find the target and return, or…cut [low, high] in half without losing the target
  - Needs sortedness

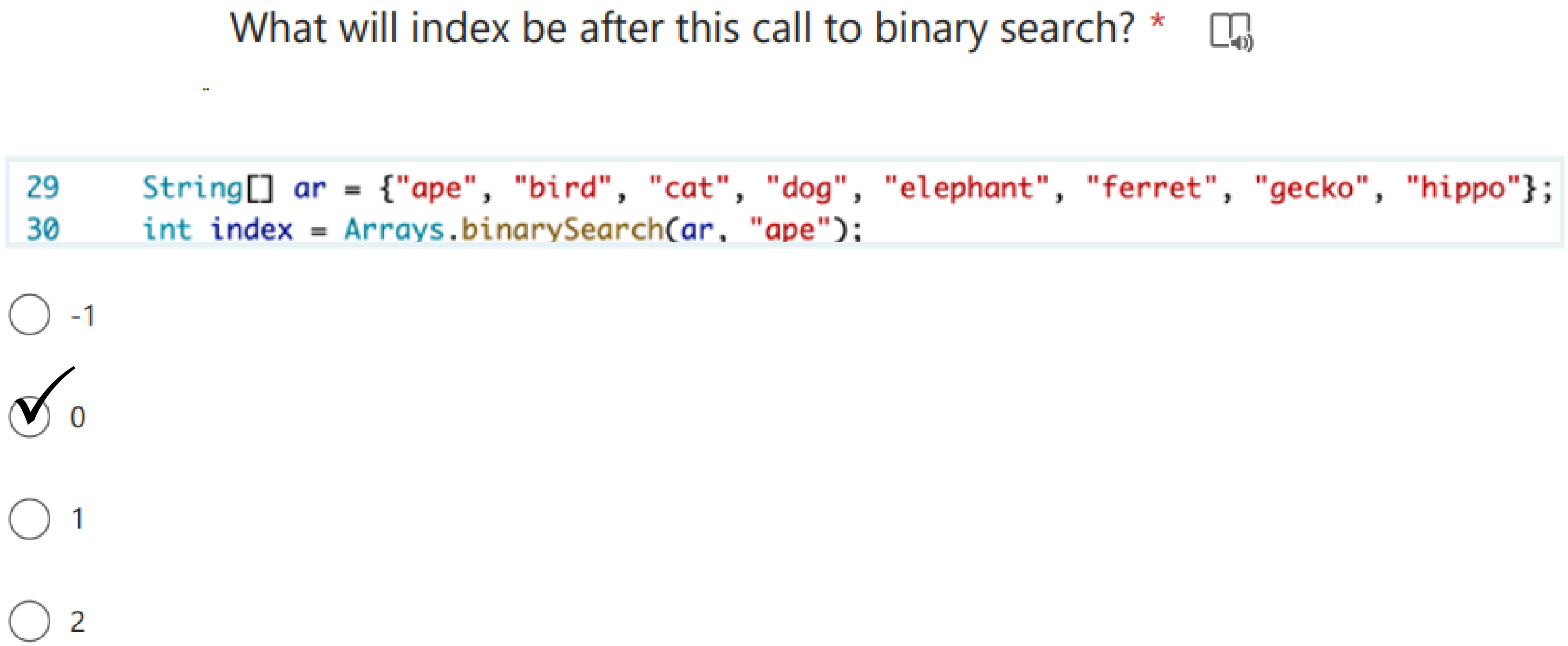
Iterative Code for DIY Binary Search?



```java
public static <T> int binarySearch(List<T> list, T target, Comparator<T> comp) {
    int low = 0;
    int high = list.size()-1;
    while (low <= high) {
        int mid = (low + high)/2;
        T midval = list.get(mid);

        int cmp = comp.compare(midval,target);
        if (cmp < 0)
            low = mid + 1;
        else if (cmp > 0)
            high = mid - 1;
        else
            return mid; // target found
    }
    return -1;  // target not found
}
```

`<T>` for generic type, can be a String list, Integer list, …, just need `target` and `Comparator` of the same type.



What will index be after this call to binary search? *

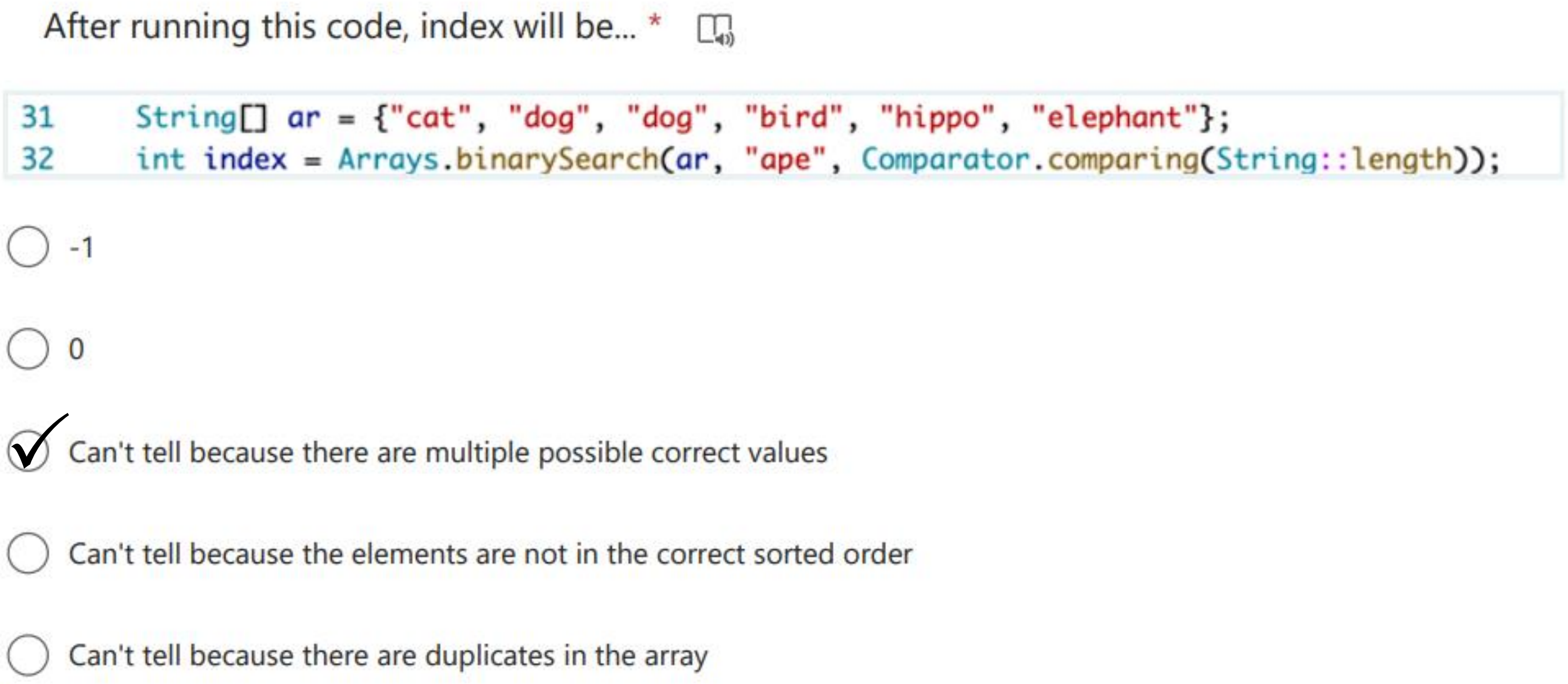
```
String[] ar = {"ape", "bird", "cat", "dog", "elephant", "ferret", "gecko", "hippo"};
int index = Arrays.binarySearch(ar, "ape");
```

○ -1

○ [✓ 0]

○ 1

○ 2



After running this code, index will be... *

```java
String[] ar = {"cat", "dog", "dog", "bird", "hippo", "elephant"};
int index = Arrays.binarySearch(ar, "ape", Comparator.comparing(String::length));
```

○ -1

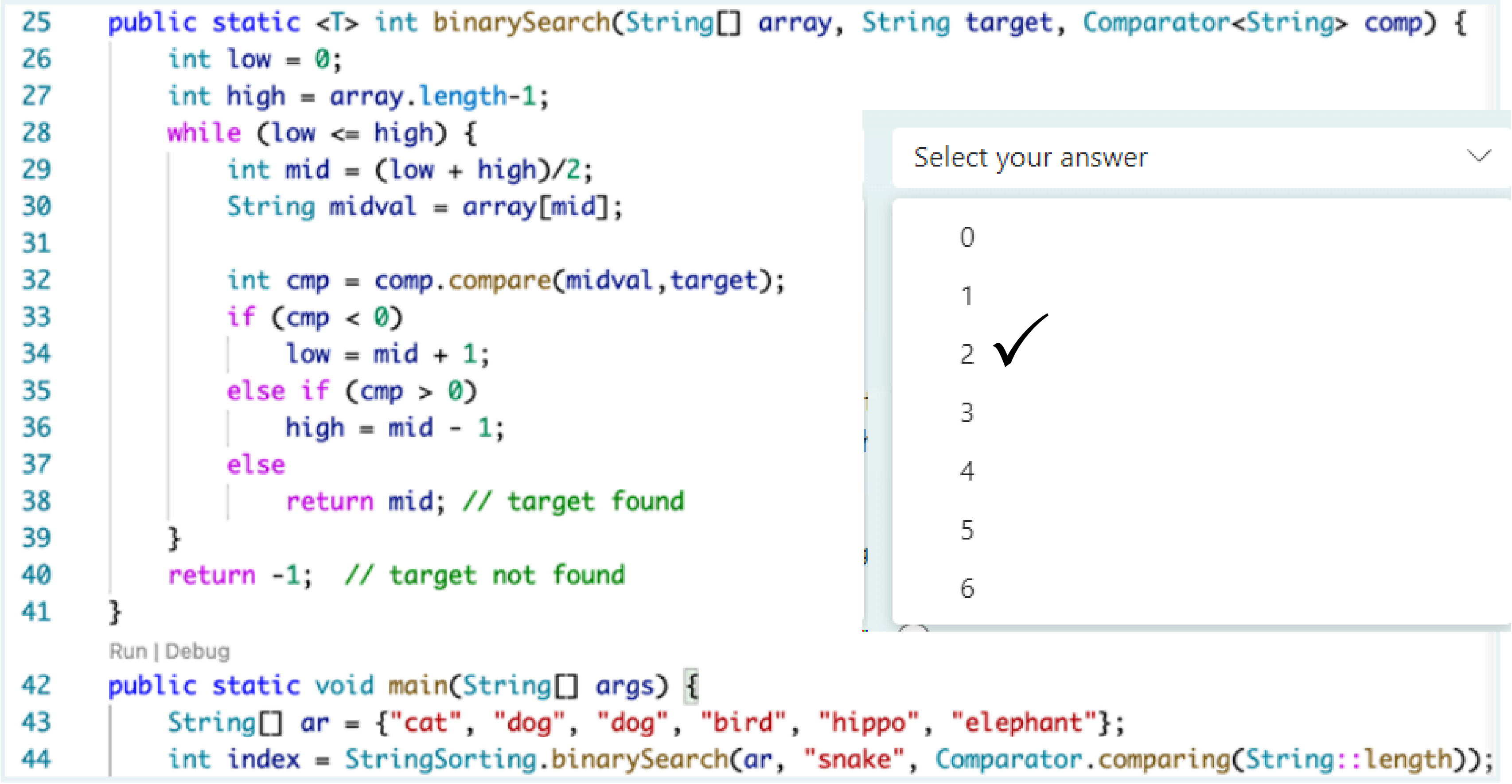
○ 0

✓ Can't tell because there are multiple possible correct values

○ Can't tell because the elements are not in the correct sorted order

○ Can't tell because there are duplicates in the array

How many calls to the compare method will result from the call to binary search in the main method on line 44? *



```
25    public static <T> int binarySearch(String[] array, String target, Comparator<String> comp) {
26        int low = 0;
27        int high = array.length-1;
28        while (low <= high) {
29            int mid = (low + high)/2;
30            String midval = array[mid];
31
32            int cmp = comp.compare(midval,target);
33            if (cmp < 0)
34                low = mid + 1;
35            else if (cmp > 0)
36                high = mid - 1;
37            else
38                return mid; // target found
39        }
40        return -1;  // target not found
41    }
      Run | Debug
42    public static void main(String[] args) {
43        String[] ar = {"cat", "dog", "dog", "bird", "hippo", "elephant"};
44        int index = StringSorting.binarySearch(ar, "snake", Comparator.comparing(String::length));
```

Select your answer

0

1

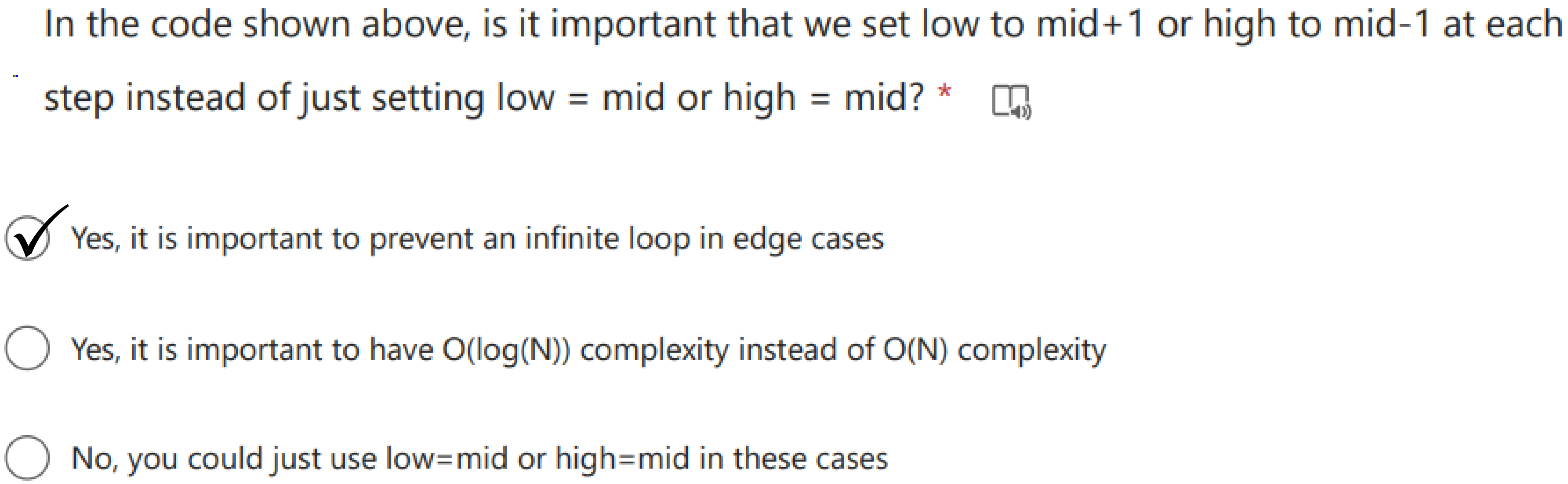
[✓ 2]

3

4

5

6



In the code shown above, is it important that we set low to mid+1 or high to mid-1 at each step instead of just setting low = mid or high = mid? *

✓ Yes, it is important to prevent an infinite loop in edge cases

○ Yes, it is important to have O(log(N)) complexity instead of O(N) complexity

○ No, you could just use low=mid or high=mid in these cases

If `low == mid` or `high == mid` before reassignment, then low/high may not change ⇒ infinite loop

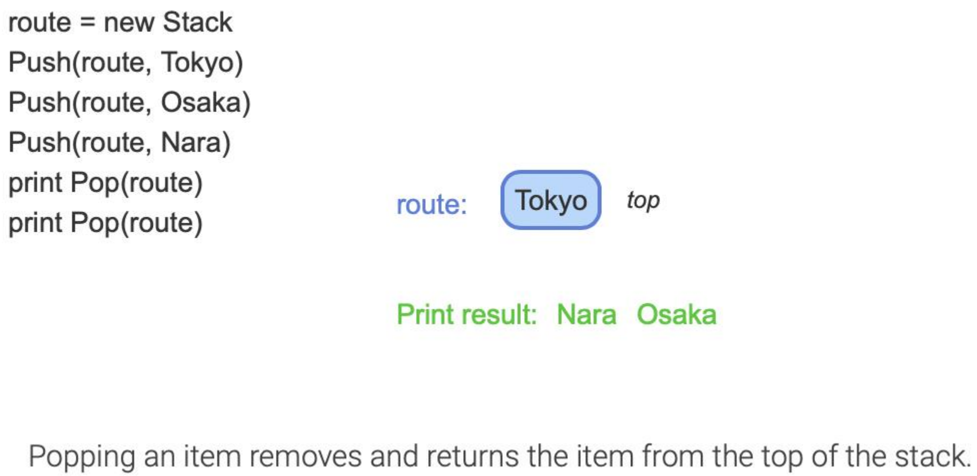
Finding the first or last?

- Algorithm we have shown does ***not*** guarantee to find the first or last such index if there are multiple.

- You will develop versions of binary search in Project 4: Autocomplete that find such indices.

Stacks, Queues, PriorityQueue: API Perspective

Stack Abstract Data Structure: LIFO List



```
route = new Stack
Push(route, Tokyo)
Push(route, Osaka)
Push(route, Nara)
print Pop(route)
print Pop(route)
```

```
Print result: Nara Osaka
```

Popping an item removes and returns the item from the top of the stack.

Zybook

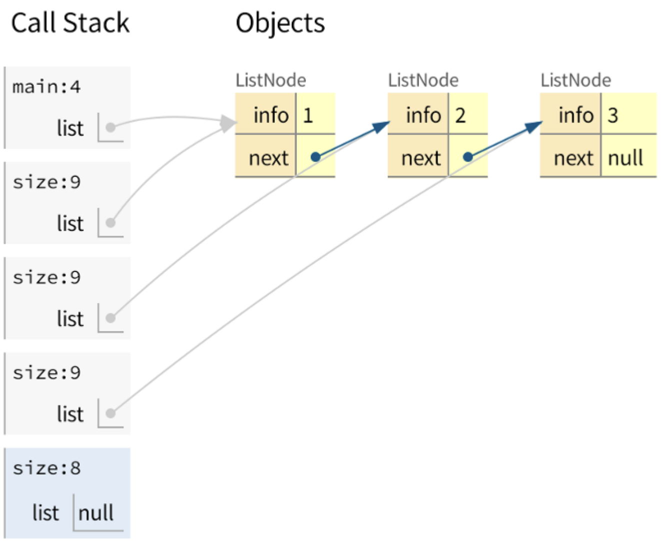
LIFO = Last In First Out

Push: Add element to stack

Pop: Get last element in

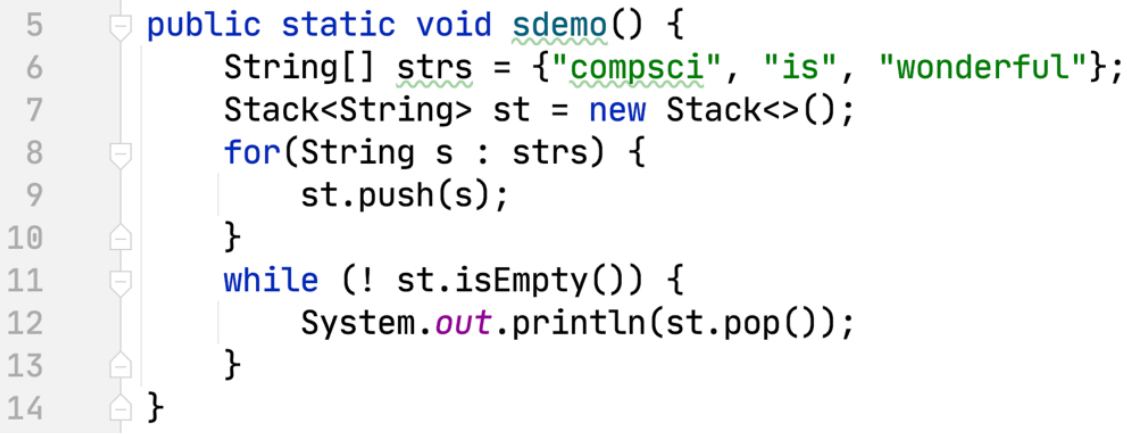
Applications?
Stack in the real world?

- Remember the call stack?
- History on your web browser / back button?
- Depth-first search in a graph (more coming soon!)



`java.util.Stack` class

- both push and pop are O(1)
  - Adds and removes from end of **`ArrayList*`**
  - Could also use **`LinkedList`**

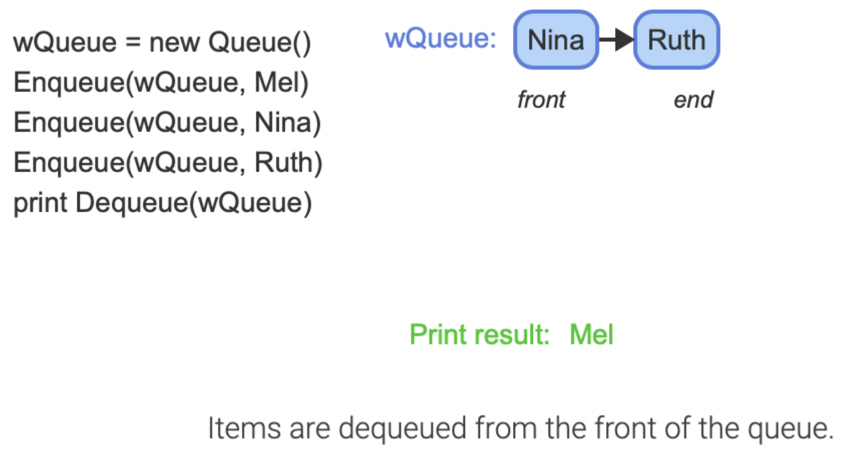


```java
public static void sdemo() {
    String[] strs = {"compsci", "is", "wonderful"};
    Stack<String> st = new Stack<>();
    for(String s : strs) {
        st.push(s);
    }
    while (! st.isEmpty()) {
        System.out.println(st.pop());
    }
}
```

```
wonderful
is
compsci
```

*Actually uses the Vector class (see docs),
but for 201 imagine ArrayList

Queue Abstract Data Structure: FIFO List



```
wQueue = new Queue()
Enqueue(wQueue, Mel)
Enqueue(wQueue, Nina)
Enqueue(wQueue, Ruth)
print Dequeue(wQueue)
```

Print result: Mel

Items are dequeued from the front of the queue.

Zybook

FIFO = **First** In First Out

Enqueue: Add element to queue

Dequeue: Remove first in element

Applications?
Queue in the real world?

- Operating system keeps track of which program should get processor time next.

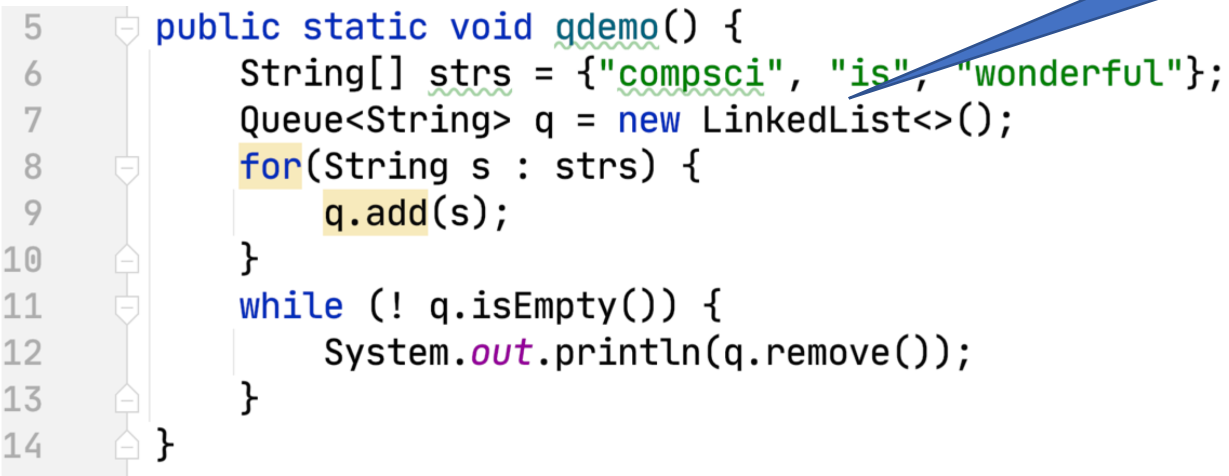
- Waitlist for class registration on DukeHub?

- Many “shortest way to get from X to Y” problems, e.g., breadth-first search in a graph (more coming soon!)

`java.util.Queue` interface

- Both add and remove are O(1)
  - Add at end of LinkedList
  - Remove from front of LinkedList

LinkedList implements the Queue interface.

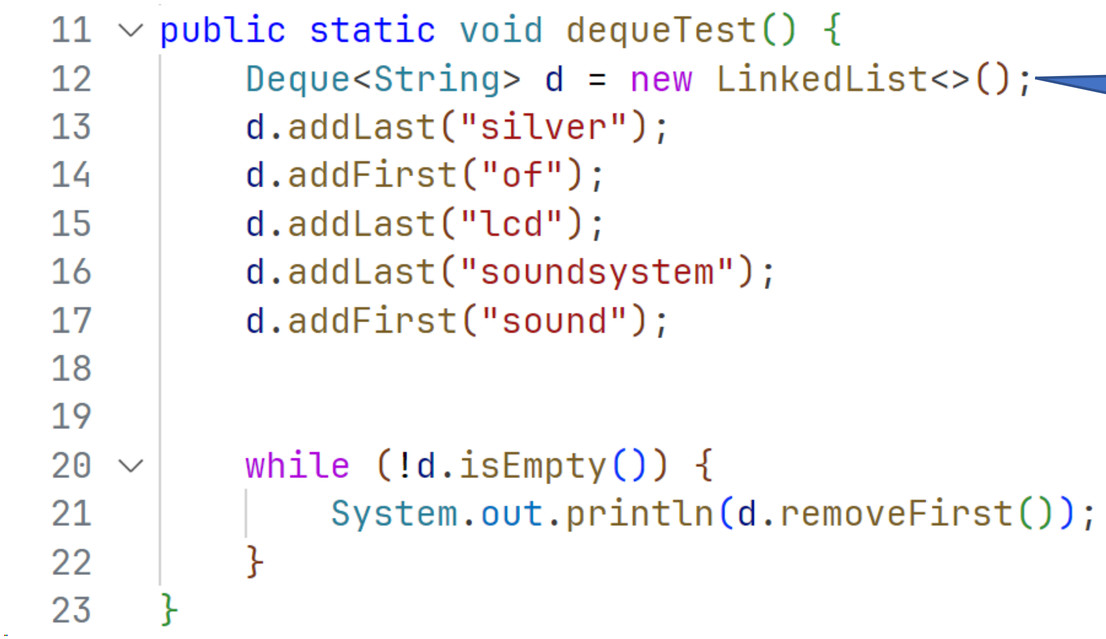


```java
public static void qdemo() {
    String[] strs = {"compsci", "is", "wonderful"};
    Queue<String> q = new LinkedList<>();
    for(String s : strs) {
        q.add(s);
    }
    while (! q.isEmpty()) {
        System.out.println(q.remove());
    }
}
```

```
compsci
is
wonderful
```

java.util.Deque interface

- “Double-ended queue”, pronounced “deck”
  - Implemented by LinkedList , which is doubly-linked
  - Add/remove to front/end (head/tail) in O(1) time



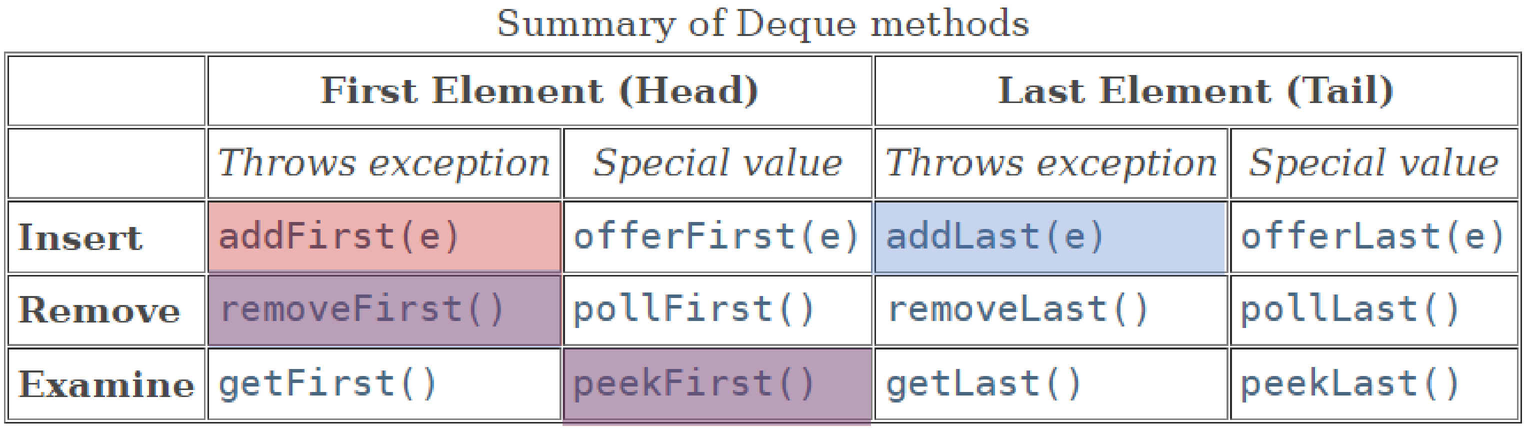
```java
public static void dequeTest() {
    Deque<String> d = new LinkedList<>();
    d.addLast("silver");
    d.addFirst("of");
    d.addLast("lcd");
    d.addLast("soundsystem");
    d.addFirst("sound");


    while (!d.isEmpty()) {
        System.out.println(d.removeFirst());
    }
}
```

LinkedList implements the Deque interface – it’s doubly linked!

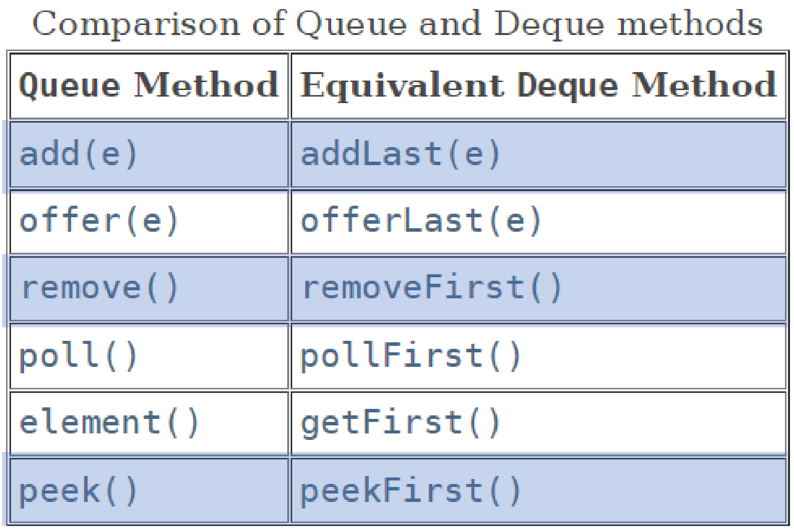
```
sound
of
silver
lcd
soundsystem
```

`java.util.Deque` interface



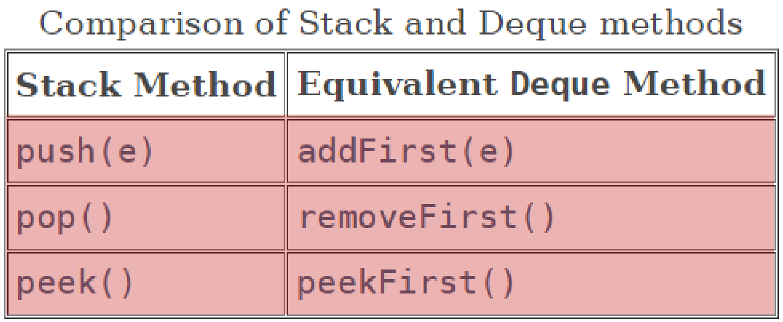
Summary of Deque methods

	First Element (Head)		Last Element (Tail)	
	Throws exception	*Special value*	*Throws exception*	*Special value*
Insert	`addFirst(e)`	`offerFirst(e)`	`addLast(e)`	`offerLast(e)`
Remove	`removeFirst()`	`pollFirst()`	`removeLast()`	`pollLast()`
Examine	`getFirst()`	`peekFirst()`	`getLast()`	`peekLast()`



Comparison of Queue and Deque methods

Queue Method	**Equivalent Deque Method**
`add(e)`	`addLast(e)`
`offer(e)`	`offerLast(e)`
`remove()`	`removeFirst()`
`poll()`	`pollFirst()`
`element()`	`getFirst()`
`peek()`	`peekFirst()`

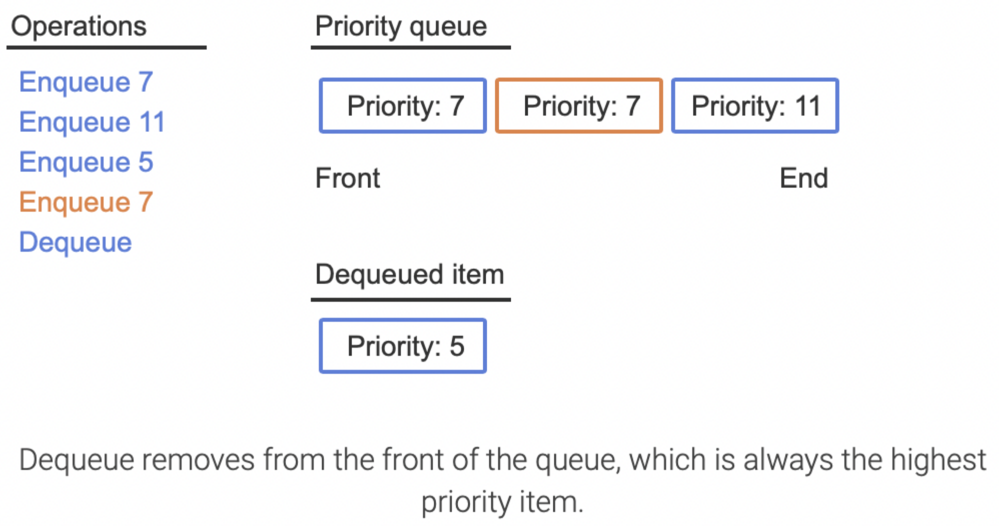


Comparison of Stack and Deque methods

Stack Method	**Equivalent Deque Method**
`push(e)`	`addFirst(e)`
`pop()`	`removeFirst()`
`peek()`	`peekFirst()`

https://docs.oracle.com/javase/8/docs/api/java/util/Deque.html

Priority Queue in the Abstract



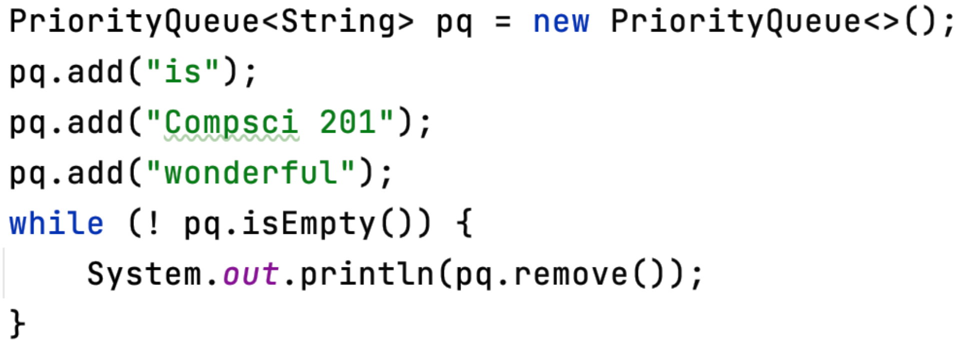
Dequeue removes from the front of the queue, which is always the highest priority item.

Queue sorted by *priority* instead of insertion order.

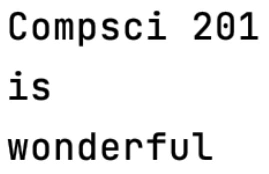
Zybook

java.util.PriorityQueue Class

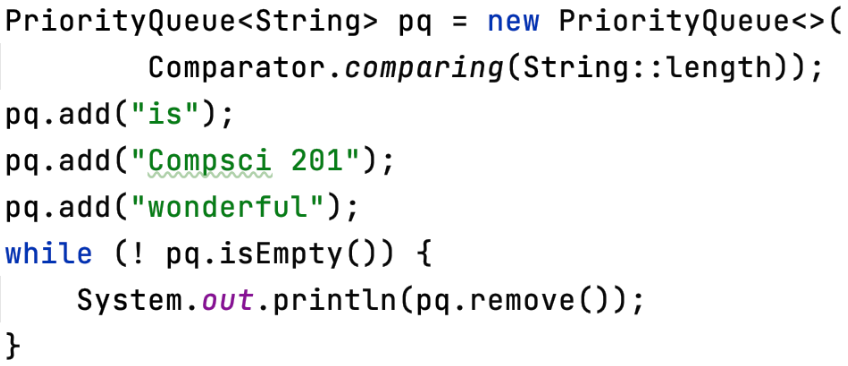
- Kept in sorted order, smallest out first
    - Objects must be Comparable OR provide Comparator to priority queue



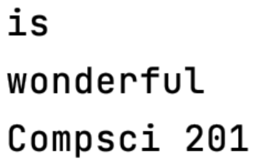
```java
PriorityQueue<String> pq = new PriorityQueue<>();
pq.add("is");
pq.add("Compsci 201");
pq.add("wonderful");
while (! pq.isEmpty()) {
    System.out.println(pq.remove());
}
```



```
Compsci 201
is
wonderful
```



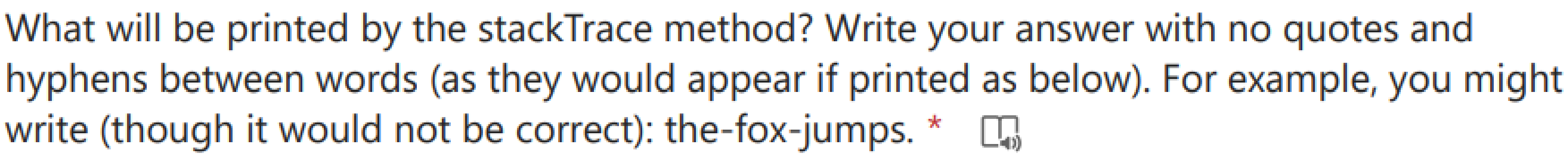
```java
PriorityQueue<String> pq = new PriorityQueue<>(
        Comparator.comparing(String::length));
pq.add("is");
pq.add("Compsci 201");
pq.add("wonderful");
while (! pq.isEmpty()) {
    System.out.println(pq.remove());
}
```



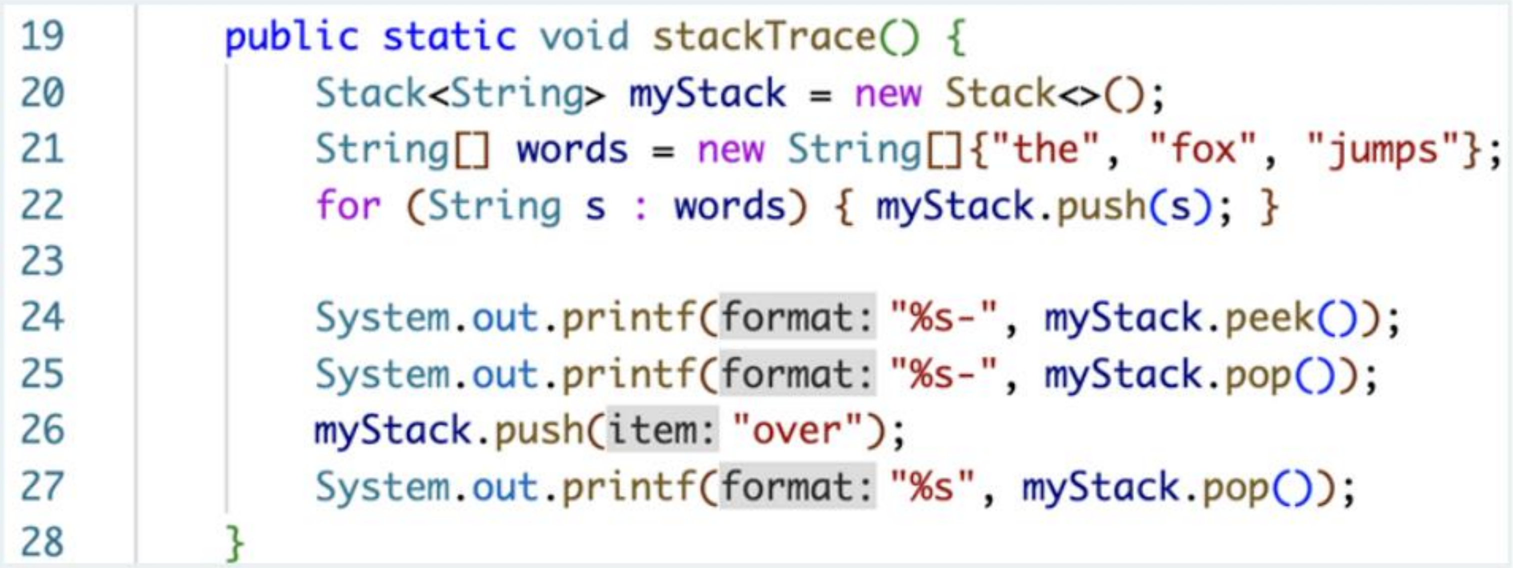
```
is
wonderful
Compsci 201
```

Complexity of Java's Priority Queue

Method	Behavior	Runtime Complexity
`add(element)`	Add an element to the priority queue	O(log(N)) *comparisons*
`remove()`	Remove and return the minimal element	O(log(N)) *comparisons*
`peek()`	Return (do *not* remove) the minimal element	O(1)
`size()`	Return number of elements	O(1)



What will be printed by the stackTrace method? Write your answer with no quotes and hyphens between words (as they would appear if printed as below). For example, you might write (though it would not be correct): the-fox-jumps. *

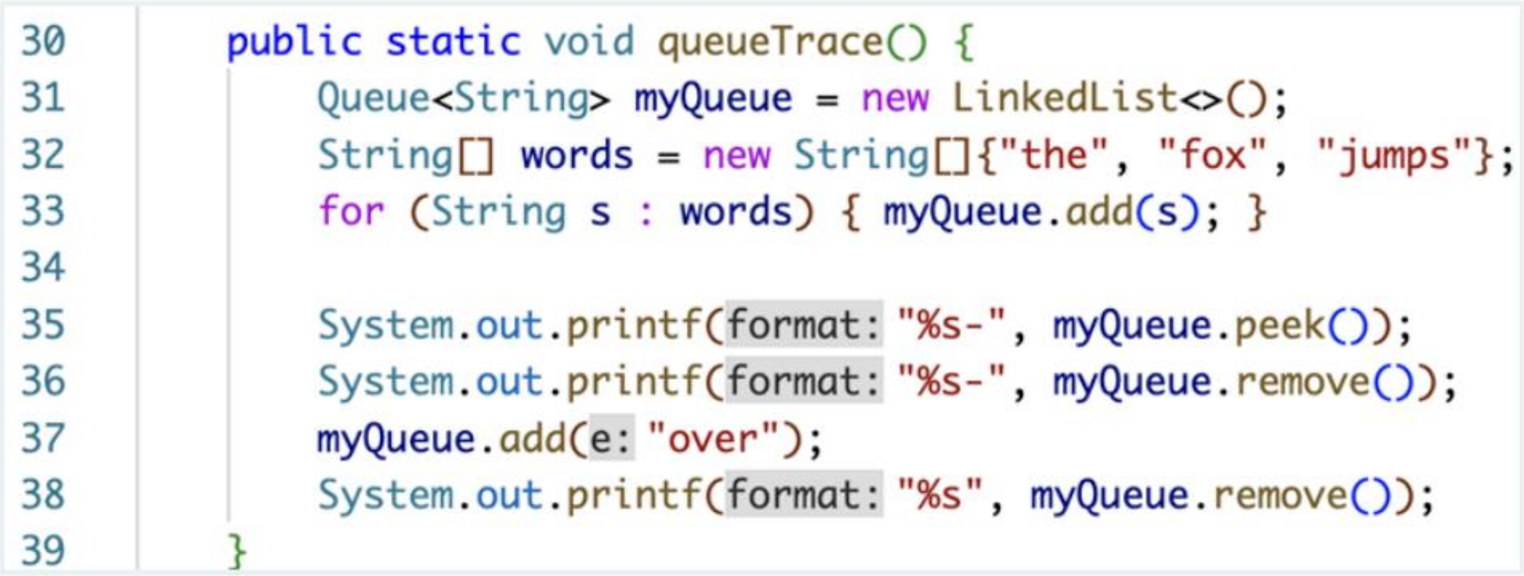


```java
    public static void stackTrace() {
        Stack<String> myStack = new Stack<>();
        String[] words = new String[]{"the", "fox", "jumps"};
        for (String s : words) { myStack.push(s); }

        System.out.printf(format: "%s-", myStack.peek());
        System.out.printf(format: "%s-", myStack.pop());
        myStack.push(item: "over");
        System.out.printf(format: "%s", myStack.pop());
    }
```

jumps-jumps-over

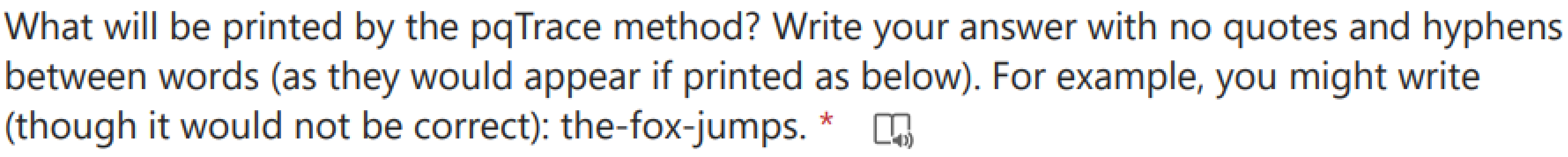
What will be printed by the queueTrace method? Write your answer with no quotes and hyphens between words (as they would appear if printed as below). For example, you might write (though it would not be correct): the-fox-jumps. *



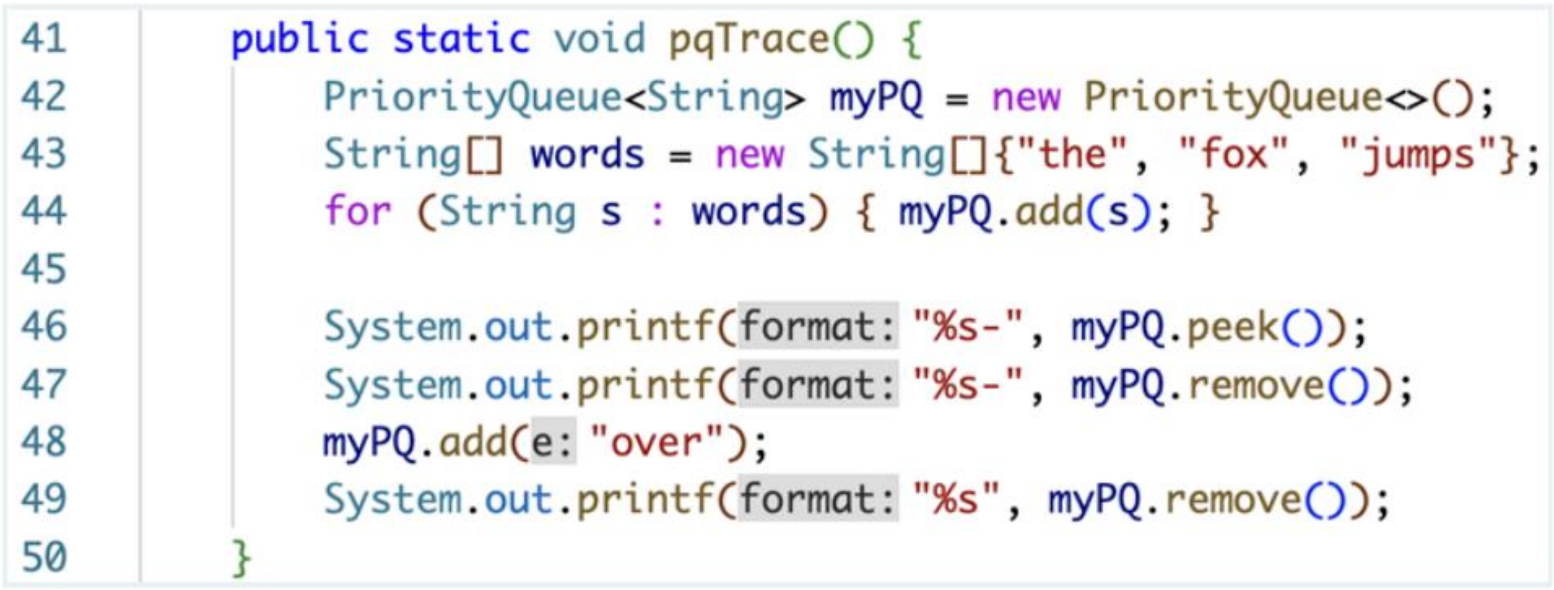
```
    public static void queueTrace() {
        Queue<String> myQueue = new LinkedList<>();
        String[] words = new String[]{"the", "fox", "jumps"};
        for (String s : words) { myQueue.add(s); }

        System.out.printf(format: "%s-", myQueue.peek());
        System.out.printf(format: "%s-", myQueue.remove());
        myQueue.add(e: "over");
        System.out.printf(format: "%s", myQueue.remove());
    }
```

the-the-fox



What will be printed by the pqTrace method? Write your answer with no quotes and hyphens between words (as they would appear if printed as below). For example, you might write (though it would not be correct): the-fox-jumps. *

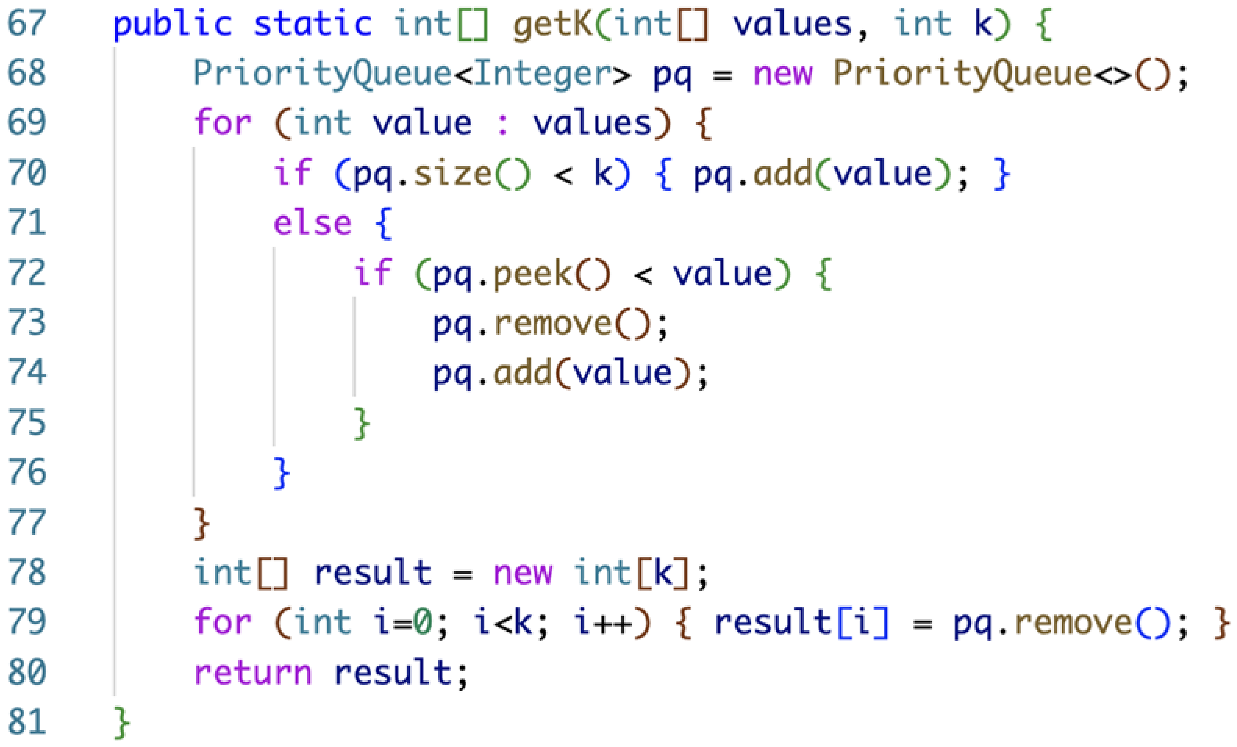


```java
    public static void pqTrace() {
        PriorityQueue<String> myPQ = new PriorityQueue<>();
        String[] words = new String[]{"the", "fox", "jumps"};
        for (String s : words) { myPQ.add(s); }

        System.out.printf(format: "%s-", myPQ.peek());
        System.out.printf(format: "%s-", myPQ.remove());
        myPQ.add(e: "over");
        System.out.printf(format: "%s", myPQ.remove());
    }
```

fox-fox-jumps

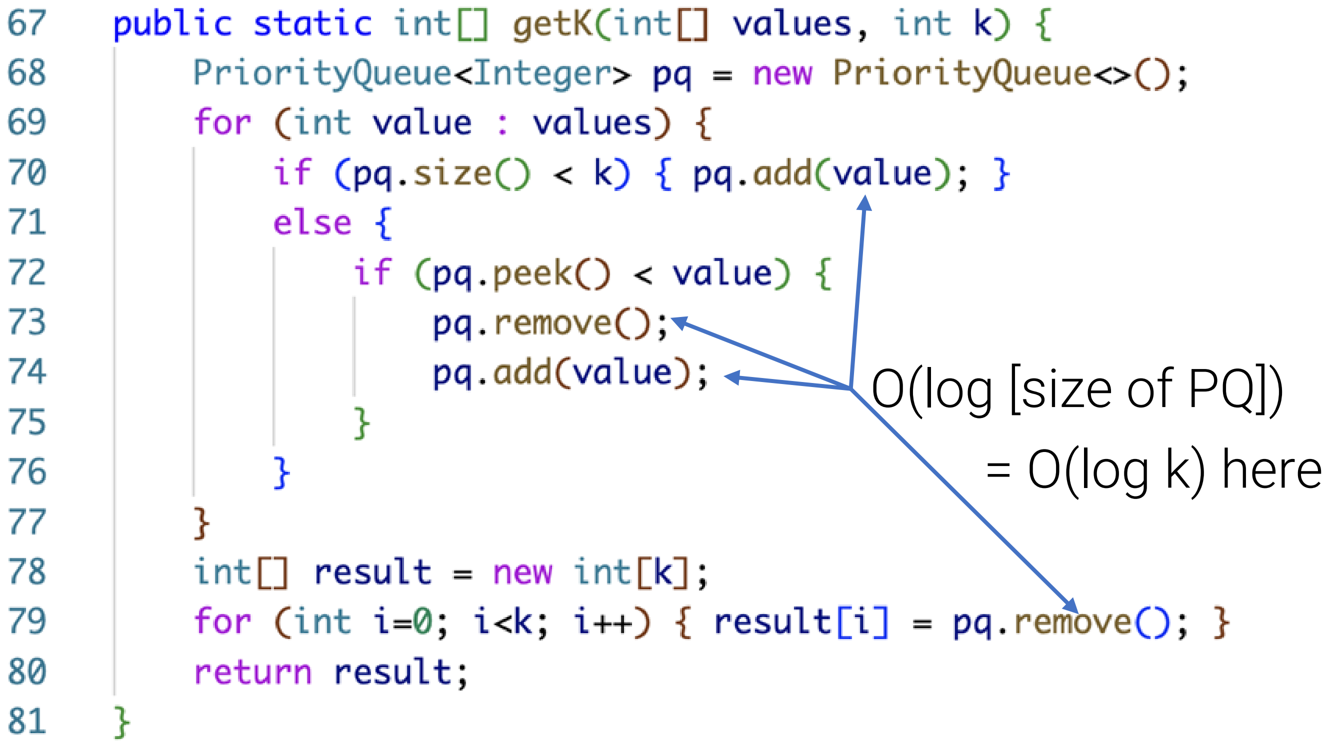
The getK method will return…



```java
public static int[] getK(int[] values, int k) {
    PriorityQueue<Integer> pq = new PriorityQueue<>();
    for (int value : values) {
        if (pq.size() < k) { pq.add(value); }
        else {
            if (pq.peek() < value) {
                pq.remove();
                pq.add(value);
            }
        }
    }
    int[] result = new int[k];
    for (int i=0; i<k; i++) { result[i] = pq.remove(); }
    return result;
}
```

The k **largest** elements of `values`

What is the asymptotic runtime complexity of the getK method as a function of N = values.length and k?



```java
public static int[] getK(int[] values, int k) {
    PriorityQueue<Integer> pq = new PriorityQueue<>();
    for (int value : values) {
        if (pq.size() < k) { pq.add(value); }
        else {
            if (pq.peek() < value) {
                pq.remove();
                pq.add(value);
            }
        }
    }
    int[] result = new int[k];
    for (int i=0; i<k; i++) { result[i] = pq.remove(); }
    return result;
}
```

O(log [size of PQ])

= O(log k) here

N iters, O(log k) time/iter. ⇒O(N log k)

How else might you find k-smallest without PQ? Sort then return first k items ⇒ O(N log N) time. PQ helps!

Binary Trees

Comparing TreeSet/Map with HashSet/Map

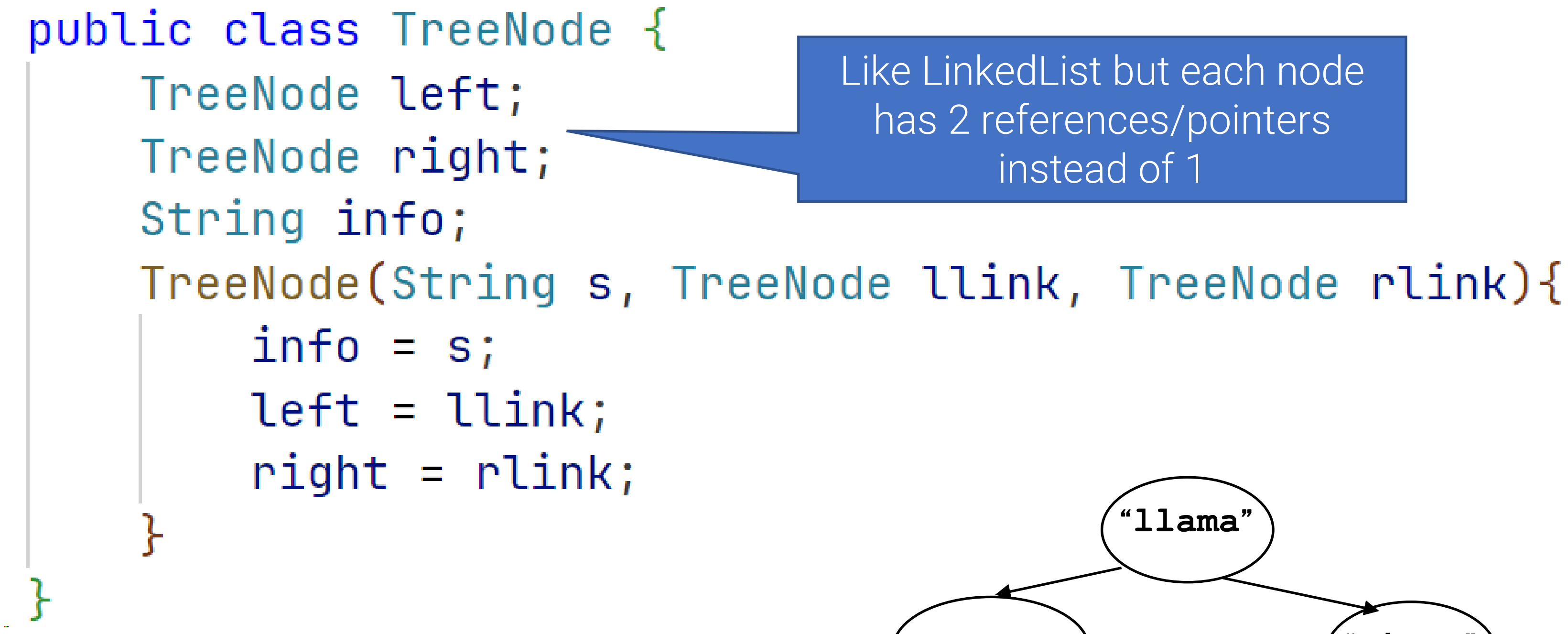
TreeSet/Map

- O(log(N)) add, contains, put, get ***are <u>not</u> amortized***.
- Stored in sorted order
  - Natural ordering by default; can provide Comparator
- Can get range of values in sorted order efficiently

HashSet/Map

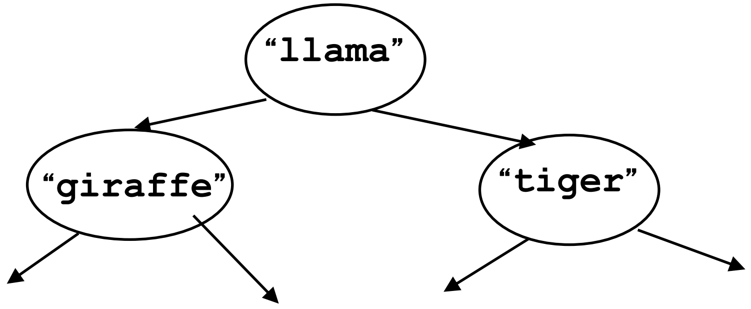
- O(1) add, contains, put, get, are ***amortized***.
- Unordered data structures
- Cannot get range efficiently, stored stored unordered

TreeNode to store Strings



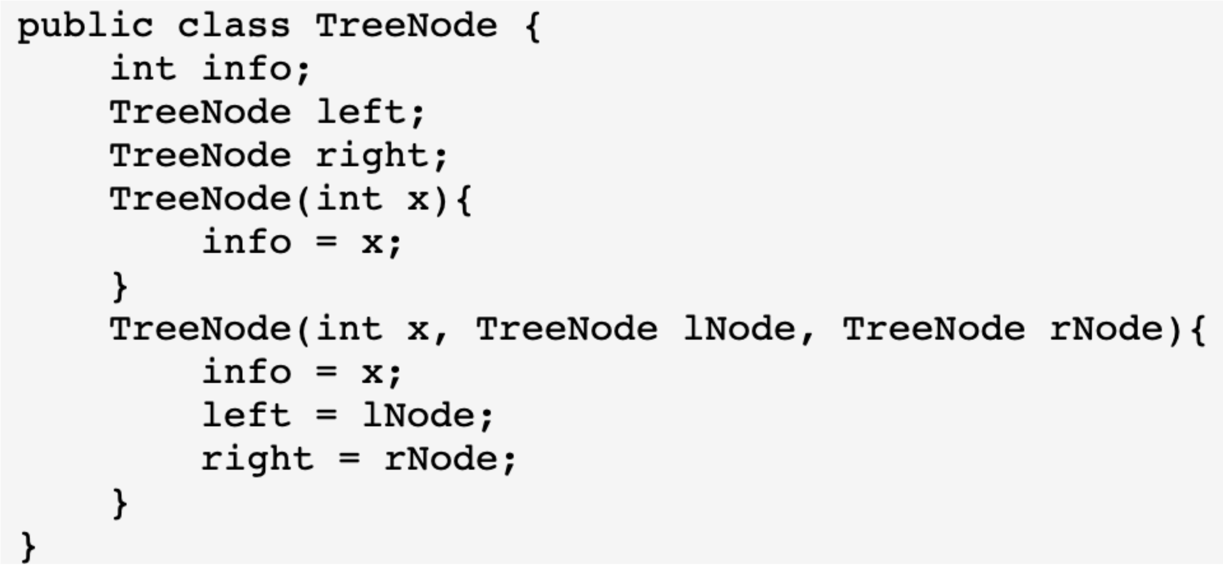
```
public class TreeNode {
    TreeNode left;
    TreeNode right;
    String info;
    TreeNode(String s, TreeNode llink, TreeNode rlink){
        info = s;
        left = llink;
        right = rlink;
    }
}
```

Like LinkedList but each node has 2 references/pointers instead of 1



APT TreeNode to store ints

APT TreeNode will only hold integer. Would need to create another class to hold Strings? Another for…?



```
public class TreeNode {
    int info;
    TreeNode left;
    TreeNode right;
    TreeNode(int x){
        info = x;
    }
    TreeNode(int x, TreeNode lNode, TreeNode rNode){
        info = x;
        left = lNode;
        right = rNode;
    }
}
```