CS 261 Lab #5

Is it really midterm time already?

Oregon State University OSU

Mix of multiple choice, matching,
short answer, true/false, and code

Code needn't be *perfect*, but should clearly be C

(missing a semi-colon is fine; only writing pseudo-code is not)

Will cover weeks 1 – 4

(everything up to and including binary search)

Header code will be provided

Be able to determine **Big O runtime** by
examining an algorithm
(in pseudocode and C) or equation

Know the Big O runtimes
of **common algorithms**
(e.g., *binary search*)

What's the **Big O runtime** for:

A method that takes $3n^2+6n+50$ steps?

What's the **Big O runtime** for:

A method that takes $3n^2+6n+50$ steps?

$O(n^2)$ because the n^2 term dominates

What's the **Big O runtime** for:

A method that takes $3n^2+6n+50$ steps?

$O(n^2)$ because the n^2 term dominates

Binary search?

What's the **Big O runtime** for:

A method that takes $3n^2+6n+50$ steps?

$O(n^2)$ because the n^2 term dominates

Binary search?

O(log n) because it halves the search space on each iteration

What's the **Big O runtime** for:

A method that takes $3n^2+6n+50$ steps?

$O(n^2)$ because the n^2 term dominates

Binary search?

O(log n) because it halves the search space on each iteration

```
for (int i = n; i > 0; i = i / 2) {
    // constant-time operations
}
```

What's the **Big O runtime** for:

A method that takes $3n^2+6n+50$ steps?

O(n^2) because the n^2 term dominates

Binary search?

O(log n) because it halves the search space on each iteration

```
for (int i = n; i > 0; i = i / 2) {
    // constant-time operations
}
```

O(log n) because the counter it halved on each iteration

What's the **Big O runtime** for:

```
for (int i = 0; i < n; i++) {
    for (int j = i; j < n; j++) {
        // constant-time operations
    }
}
```

What's the **Big O runtime** for:

```
for (int i = 0; i < n; i++) {
    for (int j = i; j < n; j++) {
        // constant-time operations
    }
}
```

$O(n^2)$ because the outer loop will run *n* times, and each time the inner loop can run up to *n* times

What's the **Big O runtime** for:

```
for (int i = 0; i < n; i++) {
    for (int j = i; j < n; j++) {
        // constant-time operations
    }
}
```

$O(n^2)$ because the outer loop will run *n* times, and each time the inner loop can run up to *n* times

```
iterator = list->frontSentinel->next;
while (iterator != list->backSentinel) {
    if (iterator->value == value)
        return 1;
    iterator = iterator->next;
}
```

What's the **Big O runtime** for:

```
for (int i = 0; i < n; i++) {
    for (int j = i; j < n; j++) {
        // constant-time operations
    }
}
```

O(n^2) because the outer loop will run *n* times, and each time the inner loop can run up to *n* times

```
iterator = list->frontSentinel->next;
while (iterator != list->backSentinel) {
    if (iterator->value == value)
        return 1;
    iterator = iterator->next;
}
```

O(n) because it needs to check each element

Know the **properties** and **operations** of the data types we've covered (e.g., *stack*, *queue*, *dynamic array*, etc.)

Be able to **compare the Big O runtimes** of common operations on different data types

Understand situations when **one data type is preferable** to another

What are the three operations of a **stack** ADT?

Which ADT would be good for **finite-length undo**?

What are the three operations of a **stack** ADT?

push, pop, & top

Which ADT would be good for **finite-length undo**?

What are the three operations of a **stack** ADT?

push, pop, & top

Which ADT would be good for **finite-length undo**?

dequeue
(need to remove old entries to have finite length)

What's the **ordering** property of a **stack**?

What's the **ordering** property of a **stack**?

last in, first out

What's the **ordering** property of a **stack**?

last in, first out

What about a **queue**?

What's the **ordering** property of a **stack**?

last in, first out

What about a **queue**?

first in, first out

What's the **ordering** property of a **stack**?

last in, first out

What about a **queue**?

first in, first out

Does the **bag** ADT have an **ordering** property?

What's the **ordering** property of a **stack**?

last in, first out

What about a **queue**?

first in, first out

Does the **bag** ADT have an **ordering** property?

nope, but the **ordered bag** does

What are the **average** and **worst-case** Big O runtimes for the the **deque** and **bag** interfaces on a **dynamic array** versus a **linked list**?

	dynamic array		linked list	
	average	worst	average	worst
add				
contains				
addFirst				
addLast				

What are the **average** and **worst-case** Big O runtimes for the the **deque** and **bag** interfaces on a **dynamic array** versus a **linked list**?

	dynamic array		linked list	
	average	worst	average	worst
add	**O(1+)**			
contains				
addFirst				
addLast				

What are the **average** and **worst-case** Big O runtimes for the the **deque** and **bag** interfaces on a **dynamic array** versus a **linked list**?

	dynamic array		linked list	
	average	worst	average	worst
add	**O(1+)**	**O(n)**		
contains				
addFirst				
addLast				

What are the **average** and **worst-case** Big O runtimes for the the **deque** and **bag** interfaces on a **dynamic array** versus a **linked list**?

	dynamic array		linked list	
	average	worst	average	worst
add	**O(1+)**	**O(n)**	**O(1)**	
contains				
addFirst				
addLast				

What are the **average** and **worst-case** Big O runtimes for the the **deque** and **bag** interfaces on a **dynamic array** versus a **linked list**?

	dynamic array		linked list	
	average	worst	average	worst
add	**O(1+)**	**O(n)**	**O(1)**	**O(1)**
contains				
addFirst				
addLast				

What are the **average** and **worst-case** Big O runtimes for the the **deque** and **bag** interfaces on a **dynamic array** versus a **linked list**?

	dynamic array		linked list	
	average	worst	average	worst
add	**O(1+)**	**O(n)**	**O(1)**	**O(1)**
contains	**O(n)**			
addFirst				
addLast				

What are the **average** and **worst-case** Big O runtimes for the the **deque** and **bag** interfaces on a **dynamic array** versus a **linked list**?

	dynamic array		linked list	
	average	worst	average	worst
add	**O(1+)**	**O(n)**	**O(1)**	**O(1)**
contains	**O(n)**	**O(n)**		
addFirst				
addLast				

What are the **average** and **worst-case** Big O runtimes for the the **deque** and **bag** interfaces on a **dynamic array** versus a **linked list**?

	dynamic array		linked list	
	average	worst	average	worst
add	**O(1+)**	**O(n)**	**O(1)**	**O(1)**
contains	**O(n)**	**O(n)**	**O(n)**	
addFirst				
addLast				

What are the **average** and **worst-case** Big O runtimes for the the **deque** and **bag** interfaces on a **dynamic array** versus a **linked list**?

	dynamic array		linked list	
	average	worst	average	worst
add	**O(1+)**	**O(n)**	**O(1)**	**O(1)**
contains	**O(n)**	**O(n)**	**O(n)**	**O(n)**
addFirst				
addLast				

What are the **average** and **worst-case** Big O runtimes for the the **deque** and **bag** interfaces on a **dynamic array** versus a **linked list**?

	dynamic array		linked list	
	average	worst	average	worst
add	**O(1+)**	**O(n)**	**O(1)**	**O(1)**
contains	**O(n)**	**O(n)**	**O(n)**	**O(n)**
addFirst	**O(1+)**			
addLast				

What are the **average** and **worst-case** Big O runtimes for the the **deque** and **bag** interfaces on a **dynamic array** versus a **linked list**?

	dynamic array		linked list	
	average	worst	average	worst
add	**O(1+)**	**O(n)**	**O(1)**	**O(1)**
contains	**O(n)**	**O(n)**	**O(n)**	**O(n)**
addFirst	**O(1+)**	**O(n)**		
addLast				

What are the **average** and **worst-case** Big O runtimes for the the **deque** and **bag** interfaces on a **dynamic array** versus a **linked list**?

	dynamic array		linked list	
	average	worst	average	worst
add	**O(1+)**	**O(n)**	**O(1)**	**O(1)**
contains	**O(n)**	**O(n)**	**O(n)**	**O(n)**
addFirst	**O(1+)**	**O(n)**	**O(1)**	
addLast				

What are the **average** and **worst-case** Big O runtimes for the the **deque** and **bag** interfaces on a **dynamic array** versus a **linked list**?

	dynamic array		linked list	
	average	worst	average	worst
add	**O(1+)**	**O(n)**	**O(1)**	**O(1)**
contains	**O(n)**	**O(n)**	**O(n)**	**O(n)**
addFirst	**O(1+)**	**O(n)**	**O(1)**	**O(1)**
addLast				

What are the **average** and **worst-case** Big O runtimes for the the **deque** and **bag** interfaces on a **dynamic array** versus a **linked list**?

	dynamic array		linked list	
	average	worst	average	worst
add	**O(1+)**	**O(n)**	**O(1)**	**O(1)**
contains	**O(n)**	**O(n)**	**O(n)**	**O(n)**
addFirst	**O(1+)**	**O(n)**	**O(1)**	**O(1)**
addLast	**O(1+)**			

What are the **average** and **worst-case** Big O runtimes for the the **deque** and **bag** interfaces on a **dynamic array** versus a **linked list**?

	dynamic array		linked list	
	average	worst	average	worst
add	**O(1+)**	**O(n)**	**O(1)**	**O(1)**
contains	**O(n)**	**O(n)**	**O(n)**	**O(n)**
addFirst	**O(1+)**	**O(n)**	**O(1)**	**O(1)**
addLast	**O(1+)**	**O(n)**		

What are the **average** and **worst-case** Big O runtimes for the the **deque** and **bag** interfaces on a **dynamic array** versus a **linked list**?

	dynamic array		linked list	
	average	worst	average	worst
add	**O(1+)**	**O(n)**	**O(1)**	**O(1)**
contains	**O(n)**	**O(n)**	**O(n)**	**O(n)**
addFirst	**O(1+)**	**O(n)**	**O(1)**	**O(1)**
addLast	**O(1+)**	**O(n)**	**O(1)**	

What are the **average** and **worst-case** Big O runtimes for the the **deque** and **bag** interfaces on a **dynamic array** versus a **linked list**?

	dynamic array		linked list	
	average	worst	average	worst
add	**O(1+)**	**O(n)**	**O(1)**	**O(1)**
contains	**O(n)**	**O(n)**	**O(n)**	**O(n)**
addFirst	**O(1+)**	**O(n)**	**O(1)**	**O(1)**
addLast	**O(1+)**	**O(n)**	**O(1)**	**O(1)**

Code

Be able to **write and understand C code** that uses or builds upon our ADTs

Be able to **show the state of an ADT** after a series of operations have been performed on it

Show the state of a dynamic array after each of the following operations:

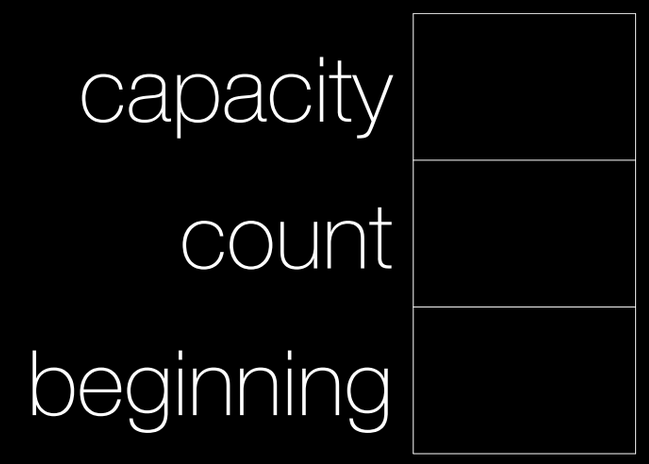
```
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```

Show the state of a dynamic array after each of the following operations:

```
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```

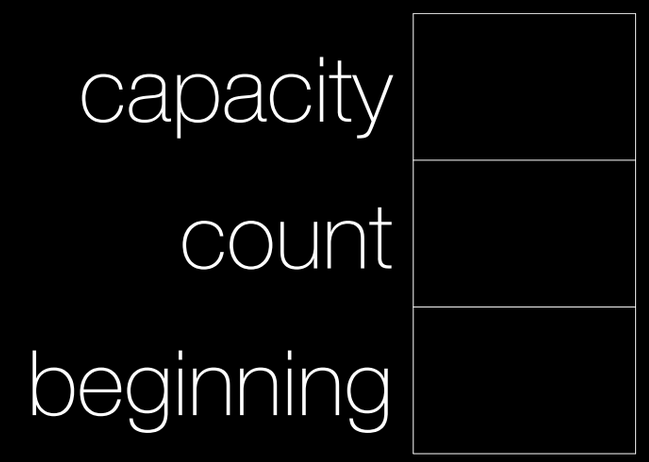
Show the state of a dynamic array after each of the following operations:

```
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```



Show the state of a dynamic array after each of the following operations:

```
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```



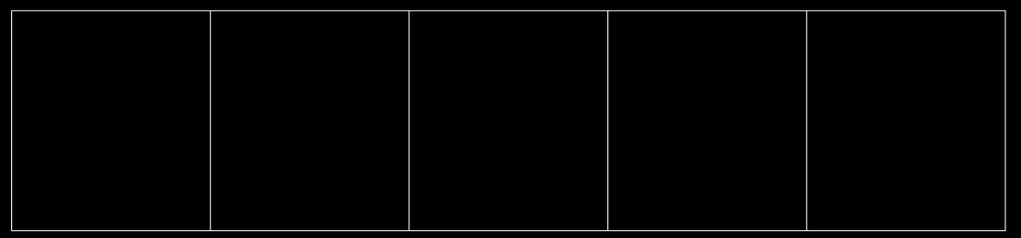
Show the state of a dynamic array after each of the following operations:

```
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```

capacity	5
count	0
beginning	0

Show the state of a dynamic array after each of the following operations:

```
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```

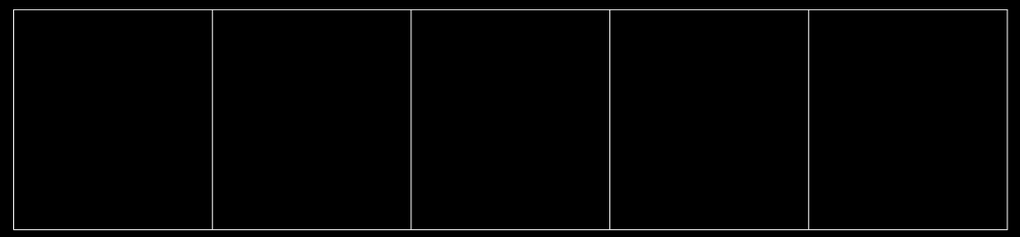


0 1 2 3 4

capacity	5
count	0
beginning	0

Show the state of a dynamic array after each of the following operations:

```
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```



capacity	5
count	0
beginning	0

Show the state of a dynamic array after each of the following operations:

```
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```

3.0				
0	1	2	3	4

capacity	5
count	0
beginning	0

Show the state of a dynamic array after each of the following operations:

```
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```

3.0				
0	1	2	3	4

capacity	5
count	1
beginning	0

Show the state of a dynamic array after each of the following operations:

```
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```

3.0				
0	1	2	3	4

capacity	5
count	1
beginning	0

Show the state of a dynamic array after each of the following operations:

```
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```

3.0	5.0			
0	1	2	3	4

capacity	5
count	1
beginning	0

Show the state of a dynamic array after each of the following operations:

```
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```

capacity	5
count	2
beginning	0

Show the state of a dynamic array after each of the following operations:

```
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```

3.0	5.0			
0	1	2	3	4

capacity	5
count	2
beginning	0

Show the state of a dynamic array after each of the following operations:

```
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```

3.0	5.0	1.0		
0	1	2	3	4

capacity	5
count	2
beginning	0

Show the state of a dynamic array after each of the following operations:

```
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```

3.0	5.0	1.0		
0	1	2	3	4

capacity	5
count	3
beginning	0

Show the state of a dynamic array after each of the following operations:

```
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```

3.0	5.0	1.0		
0	1	2	3	4

capacity	5
count	3
beginning	0

Show the state of a dynamic array after each of the following operations:

```
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```

3.0	5.0	1.0		
0	1	2	3	4

capacity	5
count	3
beginning	1

Show the state of a dynamic array after each of the following operations:

```
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```

3.0	5.0	1.0		
0	1	2	3	4

capacity	5
count	2
beginning	1

Show the state of a dynamic array after each of the following operations:

```
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```

3.0	5.0	1.0		
0	1	2	3	4

capacity	5
count	2
beginning	1

Show the state of a dynamic array after each of the following operations:

```
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```

3.0	5.0	1.0	2.0	
0	1	2	3	4

capacity	5
count	2
beginning	1

Show the state of a dynamic array after each of the following operations:

```
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```

3.0	5.0	1.0	2.0	
0	1	2	3	4

capacity	5
count	3
beginning	1

Show the state of a dynamic array after each of the following operations:

```
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```

3.0	5.0	1.0	2.0	
0	1	2	3	4

capacity	5
count	3
beginning	1

Show the state of a dynamic array after each of the following operations:

```
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```

3.0	5.0	1.0	2.0	
0	1	2	3	4

capacity	5
count	3
beginning	0

Show the state of a dynamic array after each of the following operations:

```
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```

8.0	5.0	1.0	2.0	
0	1	2	3	4

capacity	5
count	3
beginning	0

Show the state of a dynamic array after each of the following operations:

```
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```

8.0	5.0	1.0	2.0	
0	1	2	3	4

capacity	5
count	4
beginning	0

Show the state of a dynamic array after each of the following operations:

```
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```

8.0	5.0	1.0	2.0	
0	1	2	3	4

capacity	5
count	4
beginning	0

Show the state of a dynamic array after each of the following operations:

```
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```

8.0	5.0	1.0	2.0	
0	1	2	3	4

capacity	5
count	3
beginning	0

Show the state of a dynamic array after each of the following operations:

```
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```

8.0	5.0	1.0	2.0	
0	1	2	3	4

capacity	5
count	3
beginning	0

Show the state of a dynamic array after each of the following operations:

```
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```

8.0	5.0	1.0	2.0	
0	1	2	3	4

capacity	5
count	3
beginning	1

Show the state of a dynamic array after each of the following operations:

```
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```

8.0	5.0	1.0	2.0	
0	1	2	3	4

capacity	5
count	2
beginning	1

Show the state of a dynamic array after each of the following operations:

```
struct dynArrDeque d;
initDynArrDeque(&d, 5);
addBackArrDeque(&d, 3.0);
addBackArrDeque(&d, 5.0);
addBackArrDeque(&d, 1.0);
removeFrontArrDeque(&d);
addBackArrDeque(&d, 2.0);
addFrontArrDeque(&d, 8.0);
removeBackArrDeque(&d);
removeFrontArrDeque(&d);
```

capacity	5
count	2
beginning	1

Code

After the following instructions execute, how many times would you need to **pop** to get the value **8** off of the stack?

```
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```

Code

After the following instructions execute, how many times would you need to **pop** to get the value **8** off of the stack?

```
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```

Code

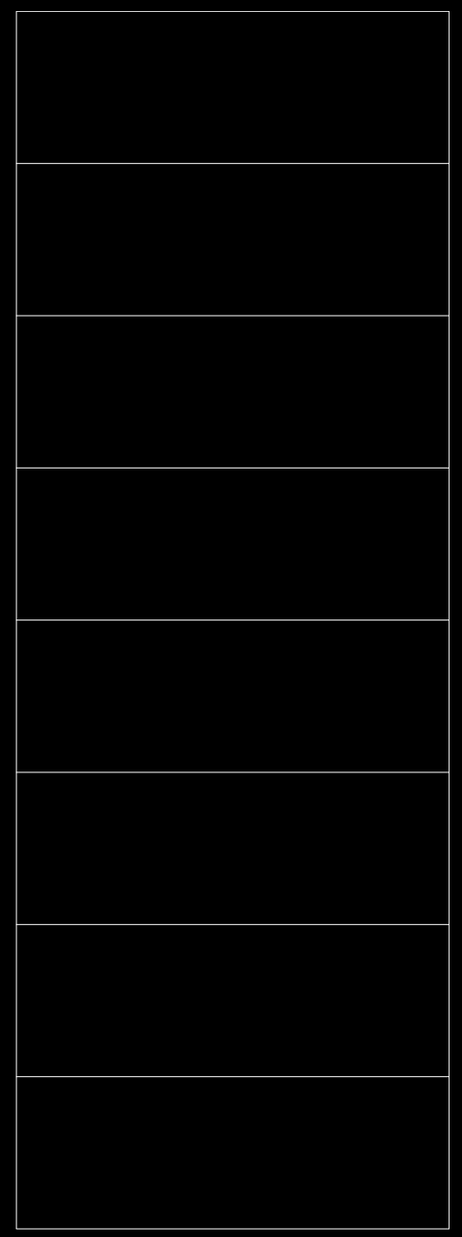
After the following instructions execute, how many times would you need to **pop** to get the value **8** off of the stack?

```
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```

Code

After the following instructions execute, how many times would you need to **pop** to get the value **8** off of the stack?

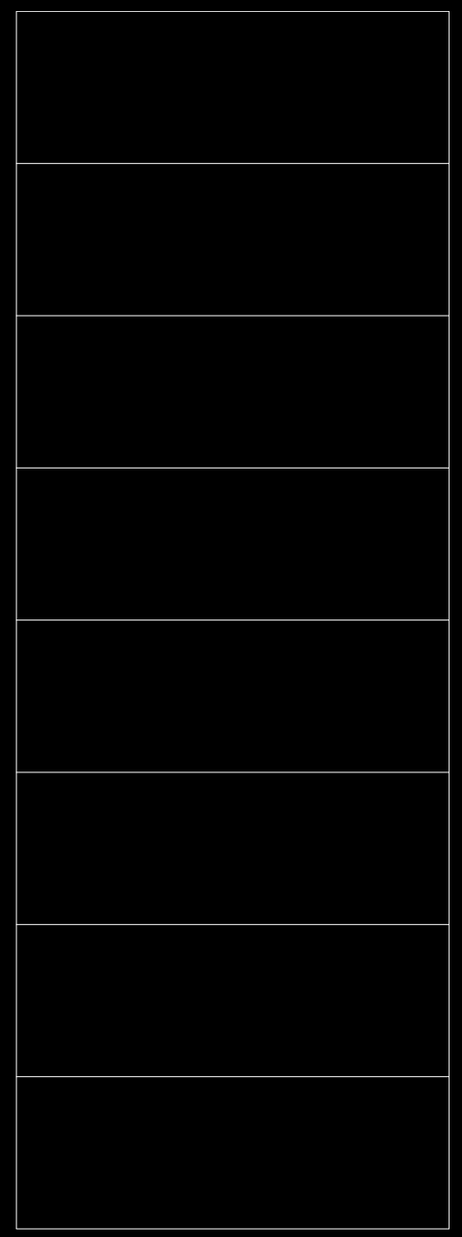
```
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```



Code

After the following instructions execute, how many times would you need to **pop** to get the value **8** off of the stack?

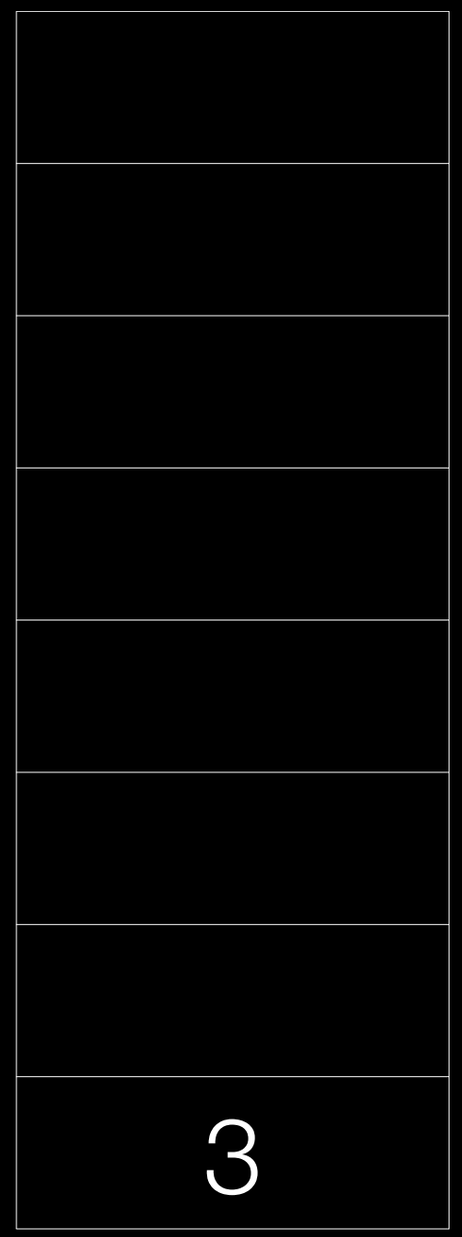
```
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```



Code

After the following instructions execute, how many times would you need to **pop** to get the value **8** off of the stack?

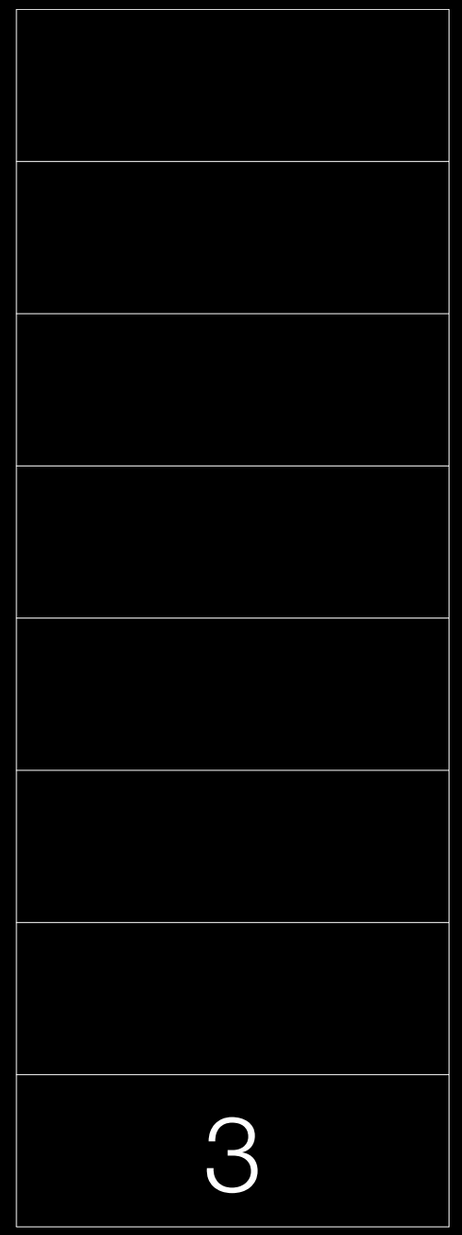
```
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```



Code

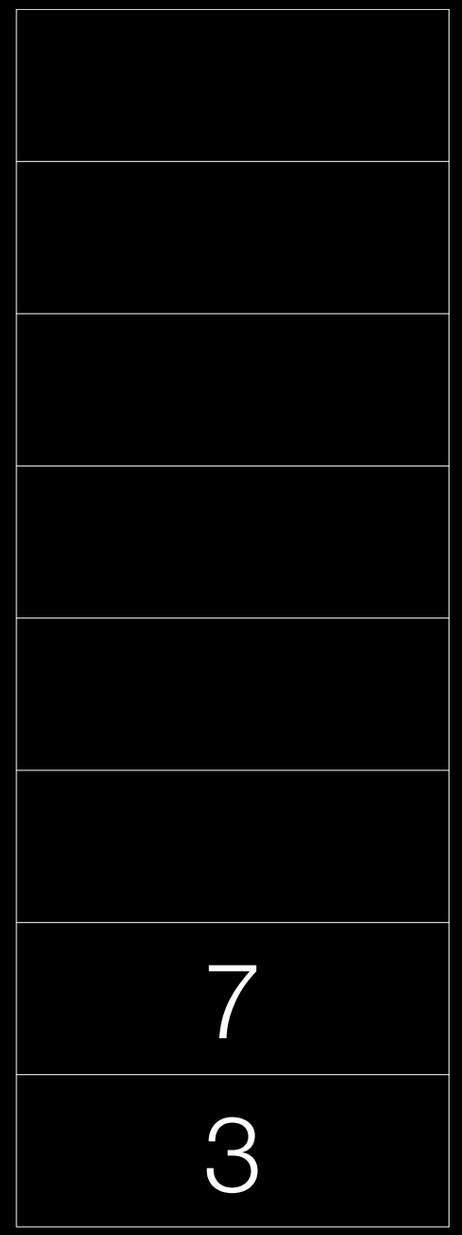
After the following instructions execute, how many times would you need to **pop** to get the value **8** off of the stack?

```
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```



After the following instructions execute, how many times would you need to **pop** to get the value **8** off of the stack?

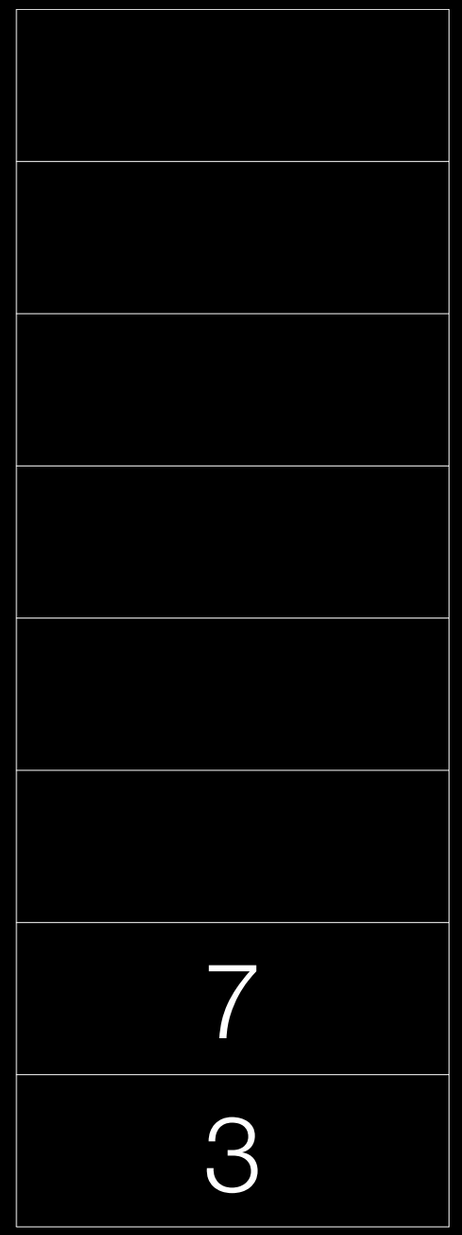
```
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```



Code

After the following instructions execute, how many times would you need to **pop** to get the value **8** off of the stack?

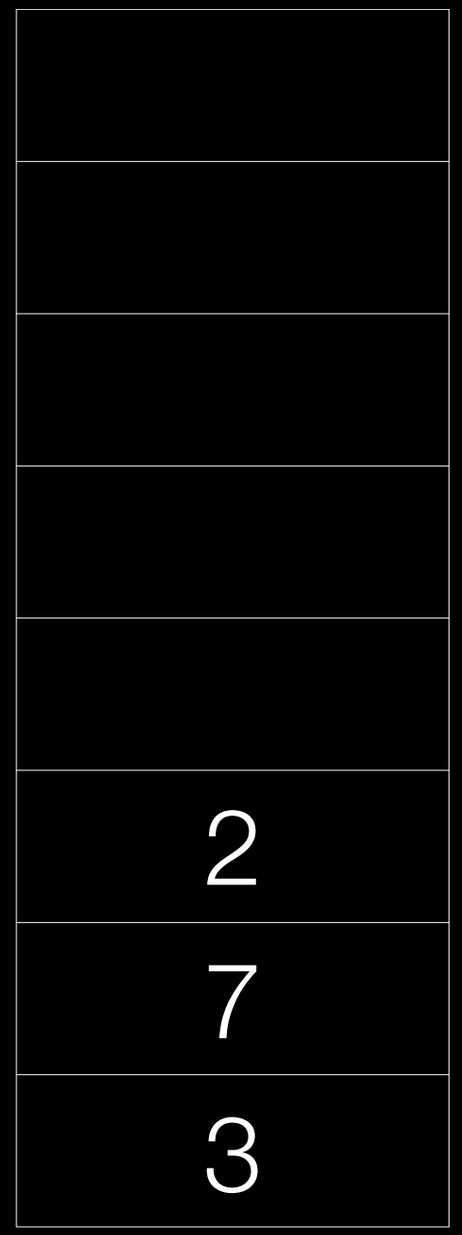
```
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```



Code

After the following instructions execute, how many times would you need to **pop** to get the value **8** off of the stack?

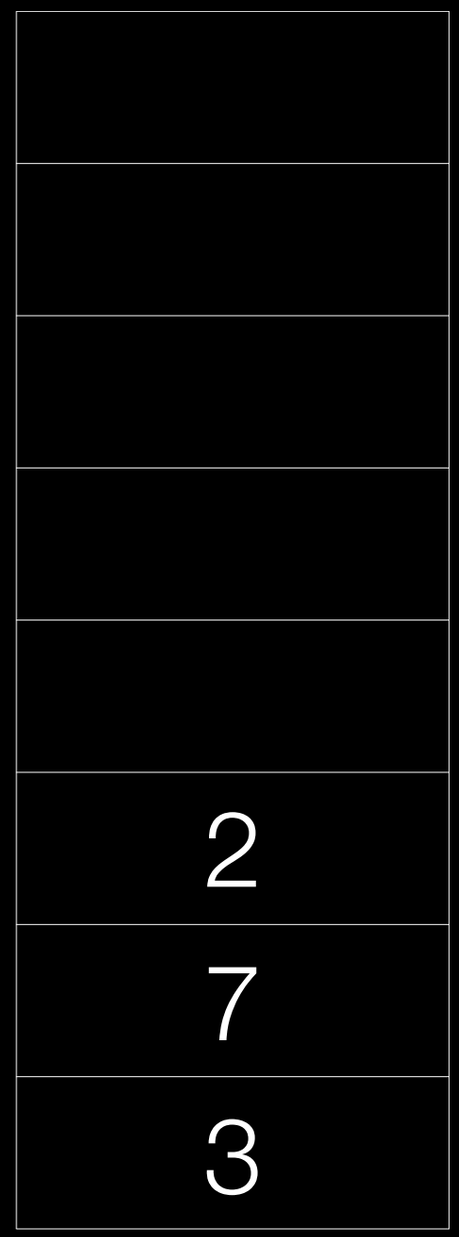
```
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```



Code

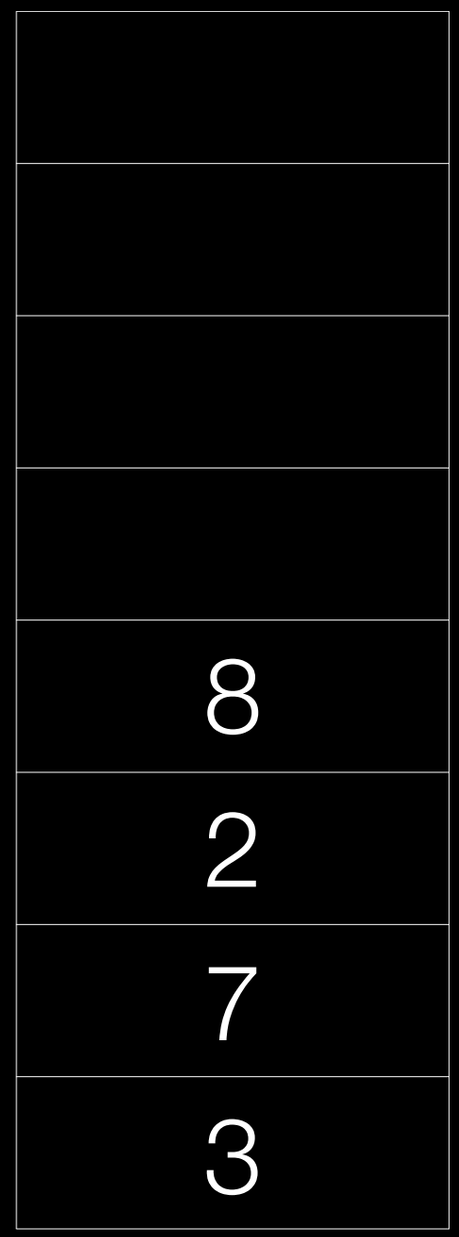
After the following instructions execute, how many times would you need to **pop** to get the value **8** off of the stack?

```
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```



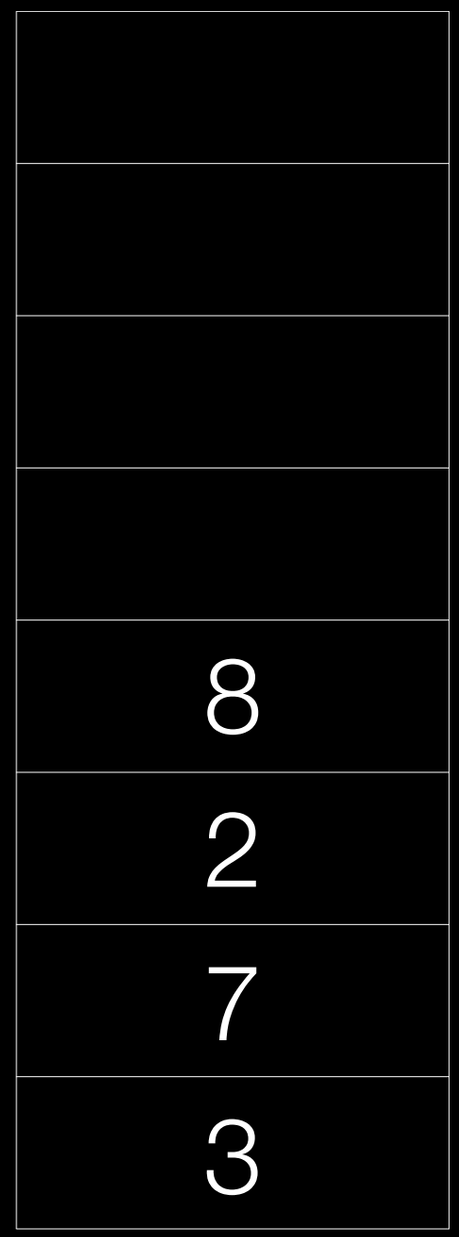
After the following instructions execute, how many times would you need to **pop** to get the value **8** off of the stack?

```
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```



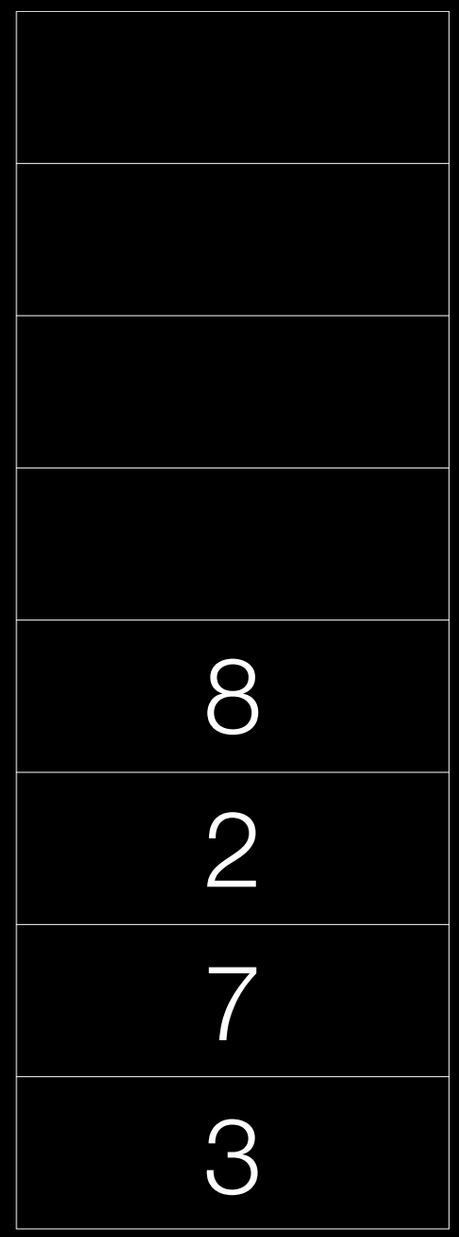
After the following instructions execute, how many times would you need to **pop** to get the value **8** off of the stack?

```
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```



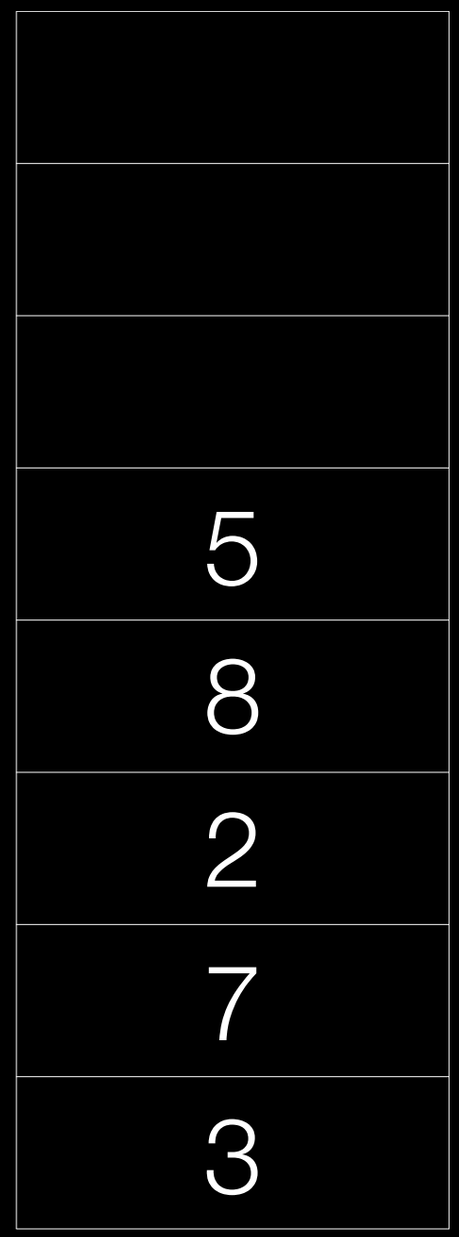
After the following instructions execute, how many times would you need to **pop** to get the value **8** off of the stack?

```
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```



After the following instructions execute, how many times would you need to **pop** to get the value **8** off of the stack?

```
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```



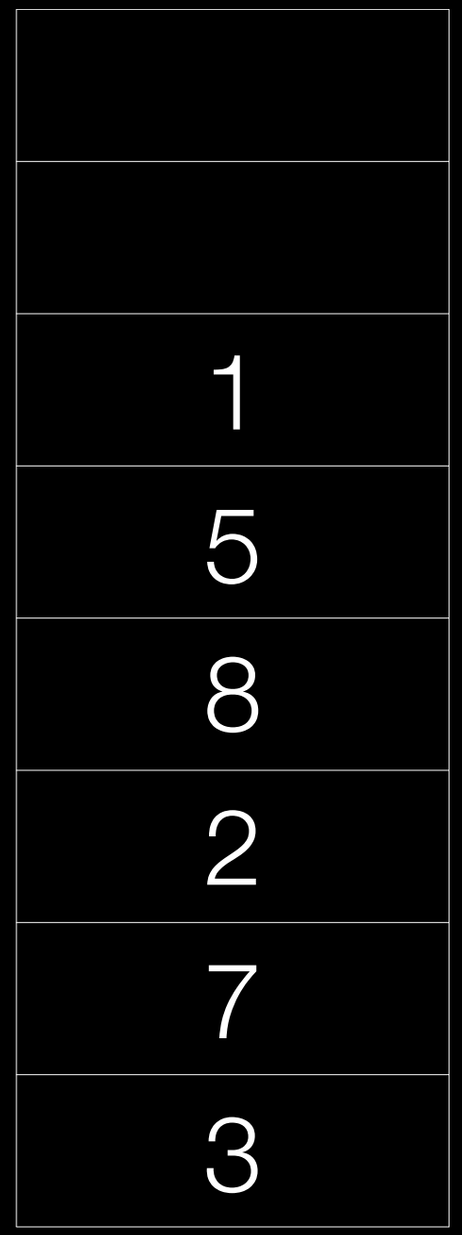
After the following instructions execute, how many times would you need to **pop** to get the value **8** off of the stack?

```
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```

Code

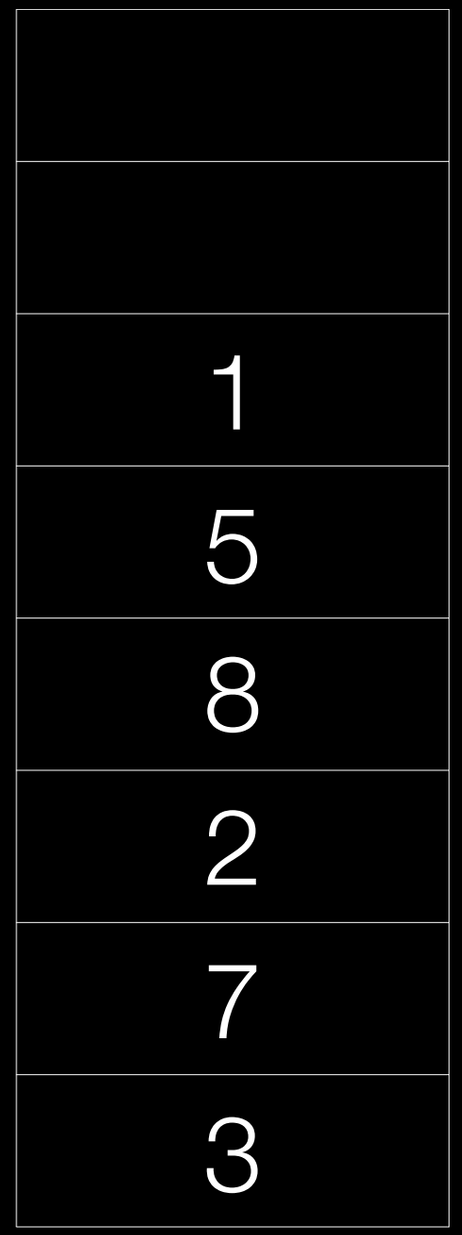
After the following instructions execute, how many times would you need to **pop** to get the value **8** off of the stack?

```
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```



After the following instructions execute, how many times would you need to **pop** to get the value **8** off of the stack?

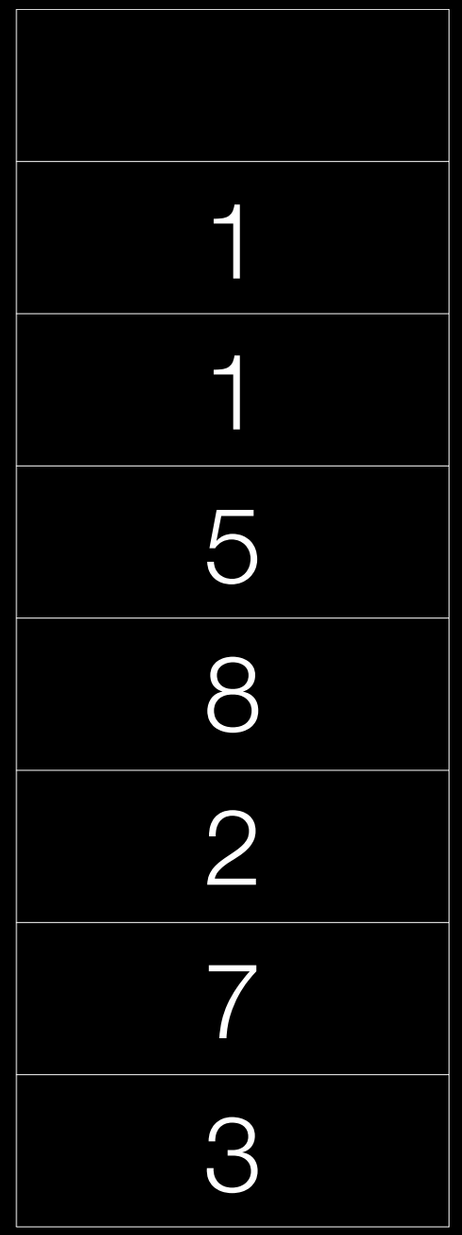
```
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```



Code

After the following instructions execute, how many times would you need to **pop** to get the value **8** off of the stack?

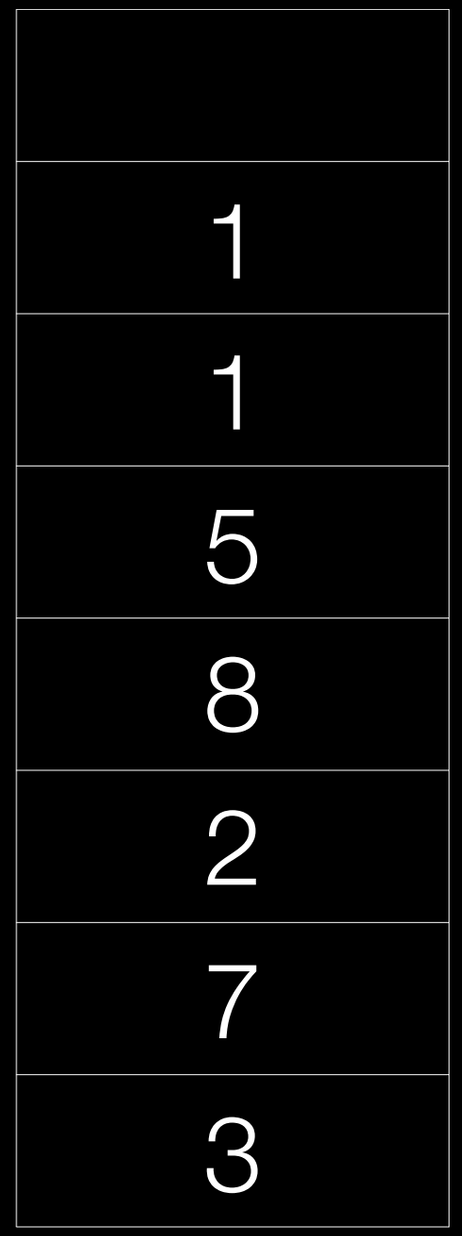
```
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```



Code

After the following instructions execute, how many times would you need to **pop** to get the value **8** off of the stack?

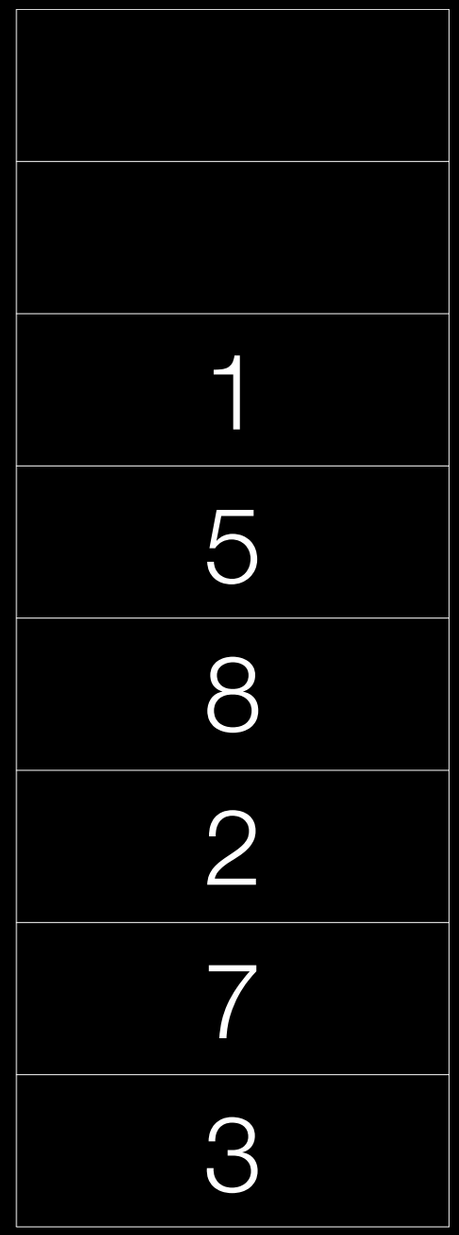
```
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```



Code

After the following instructions execute, how many times would you need to **pop** to get the value **8** off of the stack?

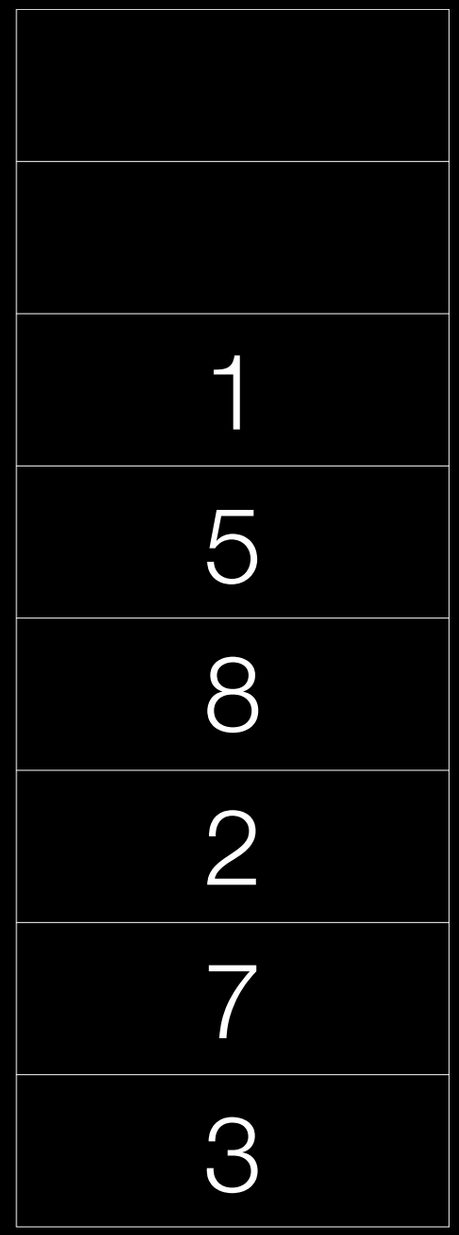
```
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```



Code

After the following instructions execute, how many times would you need to **pop** to get the value **8** off of the stack?

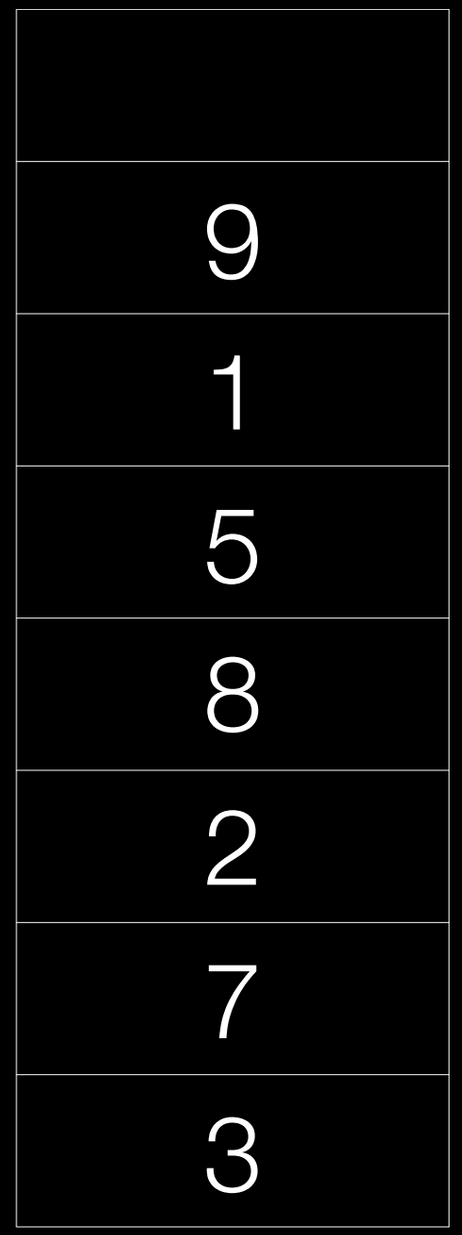
```
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```



Code

After the following instructions execute, how many times would you need to **pop** to get the value **8** off of the stack?

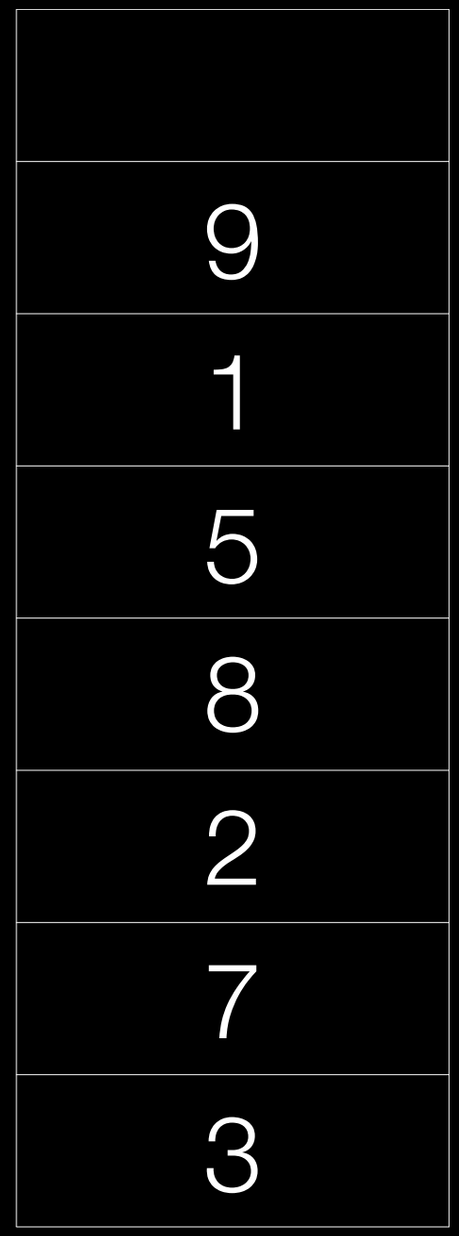
```
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```



Code

After the following instructions execute, how many times would you need to **pop** to get the value **8** off of the stack?

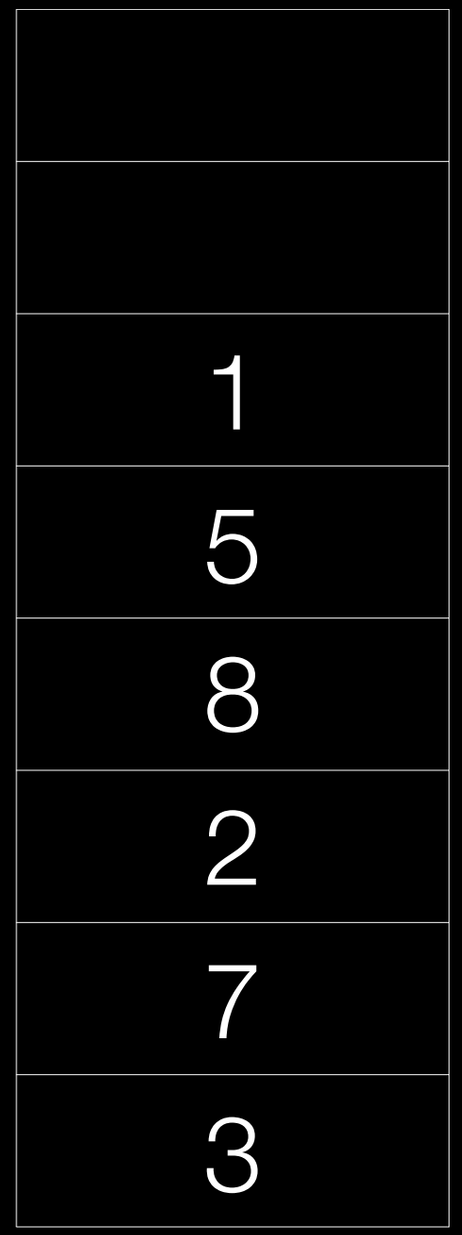
```
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```



Code

After the following instructions execute, how many times would you need to **pop** to get the value **8** off of the stack?

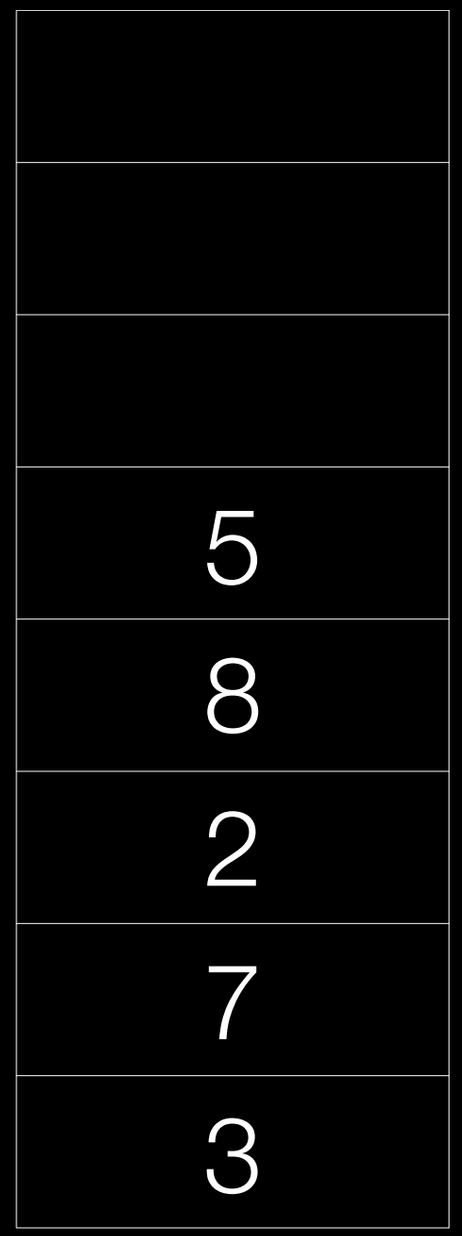
```
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```



Code

After the following instructions execute, how many times would you need to **pop** to get the value **8** off of the stack?

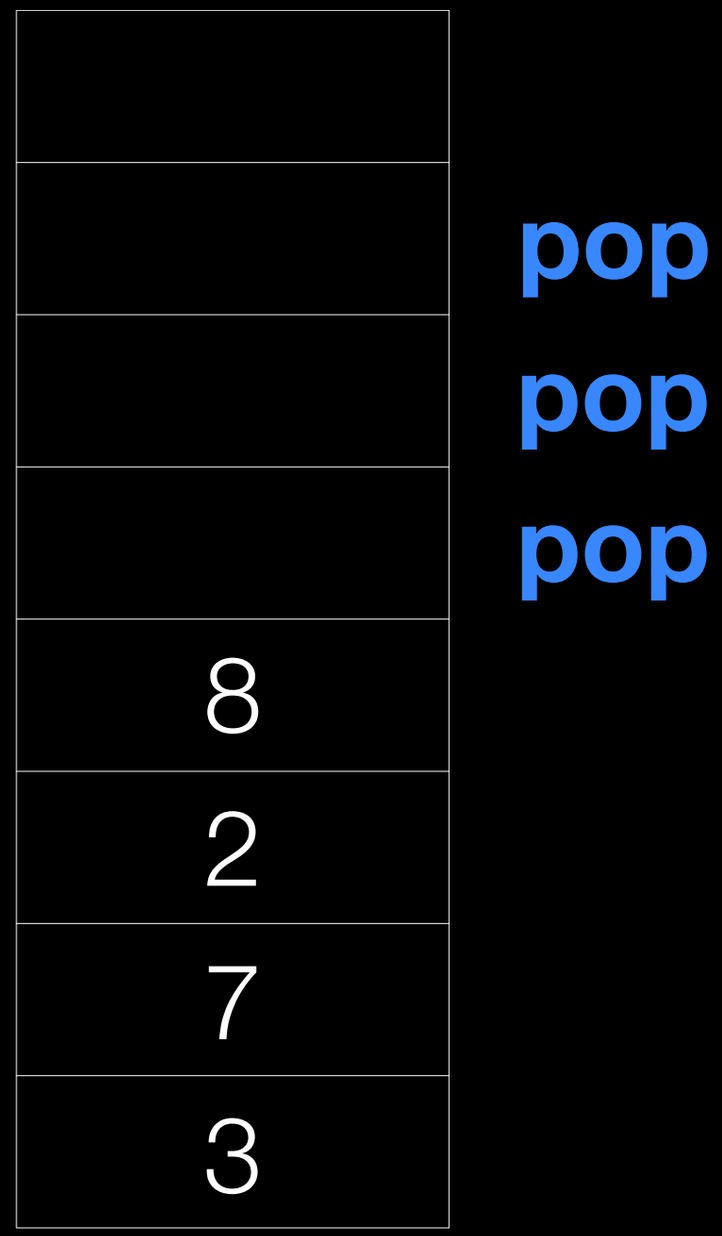
```
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```



Code

After the following instructions execute, how many times would you need to **pop** to get the value **8** off of the stack?

```
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```



Code

After the following instructions execute, how many times would you need to **pop** to get the value **8** off of the stack?

```
struct dynArrStack s;
dynArrStackInit(&s, 8);
dynArrStackPush(&s, 3);
dynArrStackPush(&s, 7);
dynArrStackPush(&s, 2);
dynArrStackPush(&s, 8);
dynArrStackTop(&s);
dynArrStackPush(&s, 5);
dynArrStackPush(&s, 1);
dynArrStackPush(&s, 1);
dynArrStackPop(&s);
dynArrStackPush(&s, 9);
```

Write a function to **print each value** of a **linked list**:

```
struct SLink {
  TYPE value;
  struct SLink *next;
};
```

```
struct List {
  struct SLink *frontSntl;
  struct SLink *backSntl;
};
```

```
void _printList(struct List *list) {
}
```

Write a function to **print each value** of a **linked list**:

```
struct SLink {
  TYPE value;
  struct SLink *next;
};
```

```
struct List {
  struct SLink *frontSntl;
  struct SLink *backSntl;
};
```

```
void _printList(struct List *list) {
  struct SLink *current;



}
```

Write a function to **print each value** of a **linked list**:

```
struct SLink {
  TYPE value;
  struct SLink *next;
};
```

```
struct List {
  struct SLink *frontSntl;
  struct SLink *backSntl;
};
```

```
void _printList(struct List *list) {
  struct SLink *current;
  current = list->frontSntl->next;



}
```

Write a function to **print each value** of a **linked list**:

```
struct SLink {
  TYPE value;
  struct SLink *next;
};
```

```
struct List {
  struct SLink *frontSntl;
  struct SLink *backSntl;
};
```

```
void _printList(struct List *list) {
    struct SLink *current;
    current = list->frontSntl->next;
    while (current != list->backSntl) {



    }
}
```

Write a function to **print each value** of a **linked list**:

```
struct SLink {
  TYPE value;
  struct SLink *next;
};
```

```
struct List {
  struct SLink *frontSntl;
  struct SLink *backSntl;
};
```

```
void _printList(struct List *list) {
    struct SLink *current;
    current = list->frontSntl->next;
    while (current != list->backSntl) {

        printf(“Value = %d\n”, current->value);



    }
}
```

Write a function to **print each value** of a **linked list**:

```
struct SLink {
  TYPE value;
  struct SLink *next;
};
```

```
struct List {
  struct SLink *frontSntl;
  struct SLink *backSntl;
};
```

```
void _printList(struct List *list) {
    struct SLink *current;
    current = list->frontSntl->next;
    while (current != list->backSntl) {

        printf("Value = %d\n", current->value);
        current = current->next;


    }
}
```

Write the Iterator functions **next()** and **hasNext()** for a linked list:

```
struct ListIterator {
  struct List *list;
  struct SLink *current;
};

int hasNext(struct ListIterator *itr) {



}
TYPE next (struct ListIterator *itr) {

}
```

Write the Iterator functions **next()** and **hasNext()** for a linked list:

```
struct ListIterator {
  struct List *list;
  struct SLink *current;
};

int hasNext(struct ListIterator *itr) {
    if (itr->current->next != itr->list->backSntl) {

    }

}
TYPE next (struct ListIterator *itr) {

}
```

Write the Iterator functions **next()** and **hasNext()** for a linked list:

```
struct ListIterator {
  struct List *list;
  struct SLink *current;
};

int hasNext(struct ListIterator *itr) {
    if (itr->current->next != itr->list->backSntl) {
        itr->current = itr->current->next;

    }

}
TYPE next (struct ListIterator *itr) {

}
```

Write the Iterator functions **next()** and **hasNext()** for a linked list:

```
struct ListIterator {
  struct List *list;
  struct SLink *current;
};

int hasNext(struct ListIterator *itr) {
    if (itr->current->next != itr->list->backSntl) {
        itr->current = itr->current->next;
        return 1;
    }


}
TYPE next (struct ListIterator *itr) {

}
```

Write the Iterator functions **next()** and **hasNext()** for a linked list:

```
struct ListIterator {
   struct List *list;
   struct SLink *current;
};

int hasNext(struct ListIterator *itr) {
    if (itr->current->next != itr->list->backSntl) {
        itr->current = itr->current->next;
        return 1;
    } else {
        return 0;
    }
}

TYPE next (struct ListIterator *itr) {

}
```

Write the Iterator functions **next()** and **hasNext()** for a linked list:

```
struct ListIterator {
   struct List *list;
   struct SLink *current;
};

int hasNext(struct ListIterator *itr) {
    if (itr->current->next != itr->list->backSntl) {
        itr->current = itr->current->next;
        return 1;
    } else {
        return 0;
    }
}

TYPE next (struct ListIterator *itr) {
    return itr->current->value;
}
```

Use the provided _binarySearch() function to **implement a contains() function** that runs in **O(log n)** time

```
int _binarySearch(TYPE *data, int count, TYPE value);

int contains (struct DynArr *da, TYPE value) {



}
```

Use the provided _binarySearch() function to **implement a contains() function** that runs in **O(log n)** time

```
int _binarySearch(TYPE *data, int count, TYPE value);

int contains (struct DynArr *da, TYPE value) {
    int index;



}
```

Use the provided _binarySearch() function to **implement a contains() function** that runs in **O(log n)** time

```
int _binarySearch(TYPE *data, int count, TYPE value);

int contains (struct DynArr *da, TYPE value) {
    int index;
    index = _binarySearch(da->data, da->size, value);



}
```

Use the provided _binarySearch() function to **implement a contains() function** that runs in **O(log n)** time

```
int _binarySearch(TYPE *data, int count, TYPE value);

int contains (struct DynArr *da, TYPE value) {
     int index;
     index = _binarySearch(da->data, da->size, value);
     if (index < da->size) {


     }

}
```

Code

Use the provided _binarySearch() function to **implement a contains() function** that runs in **O(log n)** time

```
int _binarySearch(TYPE *data, int count, TYPE value);

int contains (struct DynArr *da, TYPE value) {
    int index;
    index = _binarySearch(da->data, da->size, value);

    if (index < da->size) {
        if (da->data[index] == value) {

        }
    }

}
```

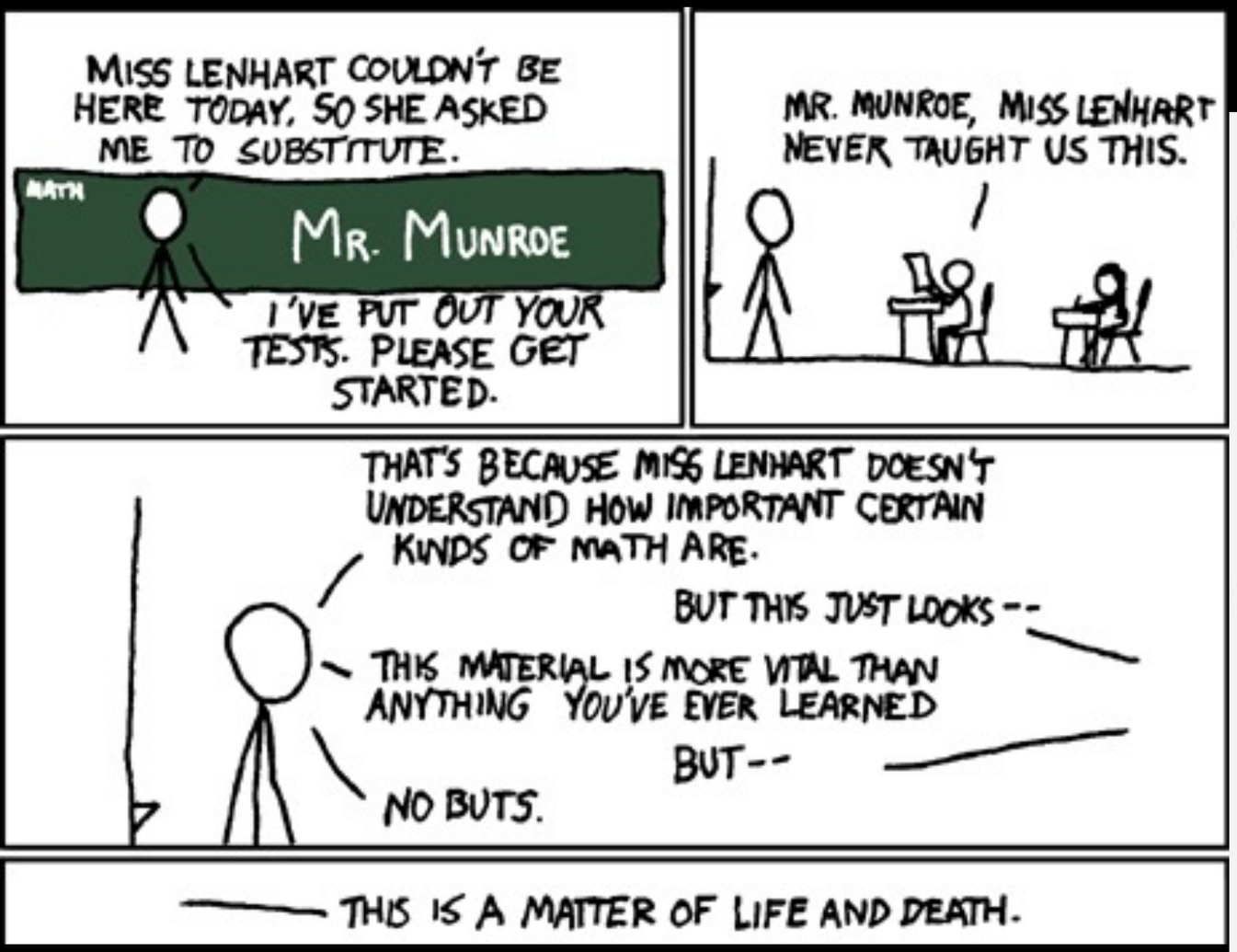
Use the provided _binarySearch() function to **implement a contains() function** that runs in **O(log n)** time

```
int _binarySearch(TYPE *data, int count, TYPE value);

int contains (struct DynArr *da, TYPE value) {
    int index;
    index = _binarySearch(da->data, da->size, value);

    if (index < da->size) {
        if (da->data[index] == value) {
            return 1;
        }
    }


}
```

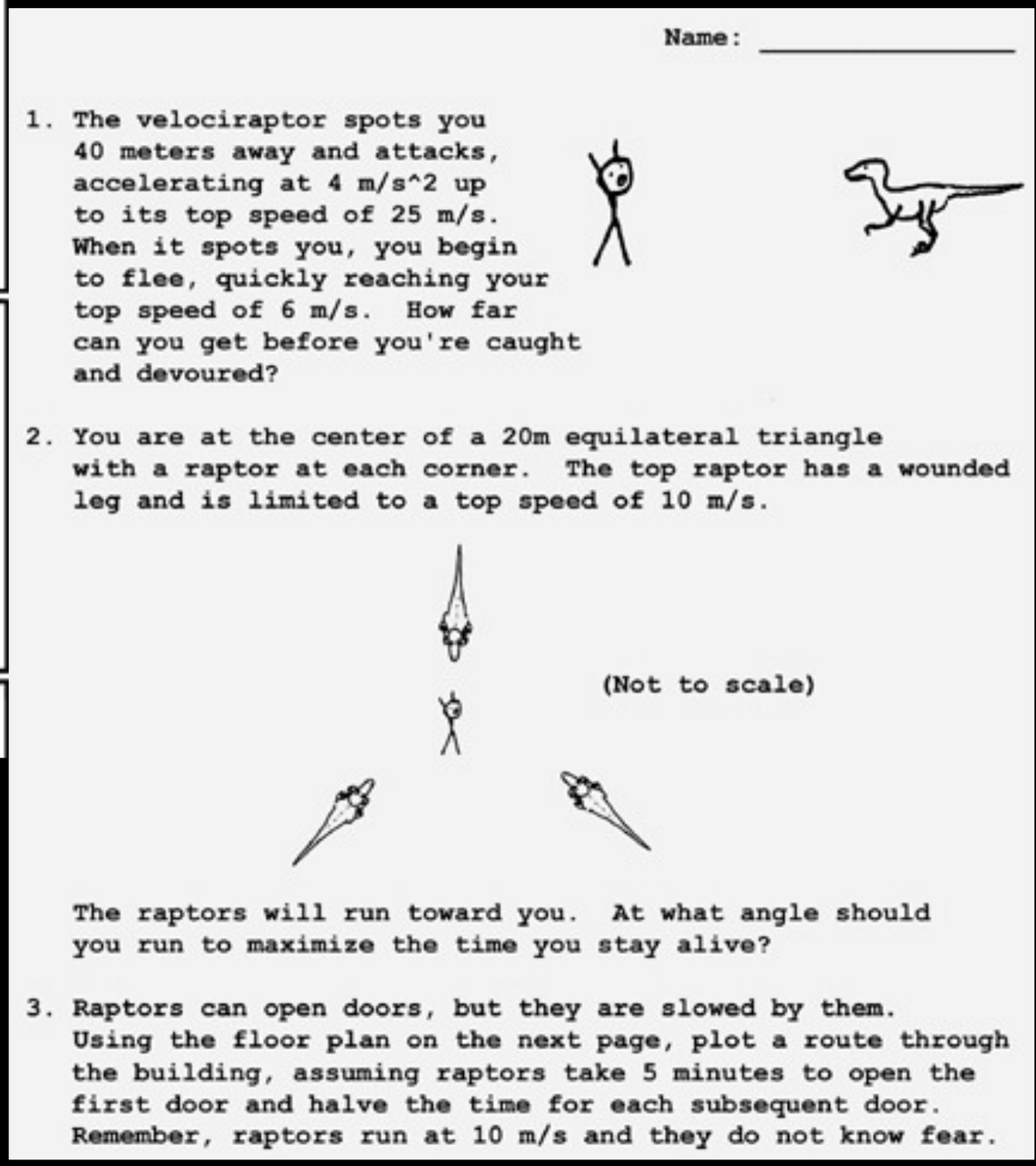
Use the provided _binarySearch() function to **implement a contains() function** that runs in **O(log n)** time

```
int _binarySearch(TYPE *data, int count, TYPE value);

int contains (struct DynArr *da, TYPE value) {
    int index;
    index = _binarySearch(da->data, da->size, value);

    if (index < da->size) {
        if (da->data[index] == value) {
            return 1;
        }
    }
    return 0;
}
```

This was practice…
midterm questions will be different!



xkcd #135



Name: _____

1. The velociraptor spots you 40 meters away and attacks, accelerating at 4 m/s^2 up to its top speed of 25 m/s. When it spots you, you begin to flee, quickly reaching your top speed of 6 m/s. How far can you get before you're caught and devoured?

2. You are at the center of a 20m equilateral triangle with a raptor at each corner. The top raptor has a wounded leg and is limited to a top speed of 10 m/s.

(Not to scale)

The raptors will run toward you. At what angle should you run to maximize the time you stay alive?

3. Raptors can open doors, but they are slowed by them. Using the floor plan on the next page, plot a route through the building, assuming raptors take 5 minutes to open the first door and halve the time for each subsequent door. Remember, raptors run at 10 m/s and they do not know fear.

That's all!

Any questions?