CS 240

Data Structures and Algorithms I

Alex Vondrak

ajvondrak@csupomona.edu

October 26, 2011

Java API

- For the sake of discussion, we've been implementing stacks by hand
- Java has many data structures included with its libraries
- Because it's so extensive, it's important to be familiar with the Java libraries' APIs
- `http://download.oracle.com/javase/7/docs/api/java/util/Stack.html`

One Final Talking Point...

Q: How efficient are the stack operations?

A: Depends on the implementation!

Let's take a look at our `ArrayStack`:

Operation	**Worst-Case Running Time**
`size`	
`isEmpty`	
`top`	
`pop`	
`push`	

One Final Talking Point...

Q: How efficient are the stack operations?

A: Depends on the implementation!

Let's take a look at our `ArrayStack`:

Operation	**Worst-Case Running Time**
`size`	$O(1)$
`isEmpty`	
`top`	
`pop`	
`push`	

One Final Talking Point...

Q: How efficient are the stack operations?

A: Depends on the implementation!

Let's take a look at our `ArrayStack`:

Operation	**Worst-Case Running Time**
`size`	$O(1)$
`isEmpty`	$O(1)$
`top`	
`pop`	
`push`	

One Final Talking Point...

Q: How efficient are the stack operations?

A: Depends on the implementation!

Let's take a look at our `ArrayStack`:

Operation	**Worst-Case Running Time**
`size`	$O(1)$
`isEmpty`	$O(1)$
`top`	$O(1)$
`pop`	
`push`	

One Final Talking Point...

Q: How efficient are the stack operations?

A: Depends on the implementation!

Let's take a look at our `ArrayStack`:

Operation	**Worst-Case Running Time**
`size`	$O(1)$
`isEmpty`	$O(1)$
`top`	$O(1)$
`pop`	$O(1)$
`push`	

One Final Talking Point...

Q: How efficient are the stack operations?

A: Depends on the implementation!

Let's take a look at our `ArrayStack`:

Operation	**Worst-Case Running Time**
`size`	$O(1)$
`isEmpty`	$O(1)$
`top`	$O(1)$
`pop`	$O(1)$
`push`	$O(n)$

Amortized Analysis

- At worst, `push` takes $O(n)$ time to resize the array...
- ...But how often does that happen?
- Idea: average out the cost of n operations performed in sequence

```
public void push(E value) {
   if (size() == data.length)
      grow();
   data[++top] = value;
}
```

Q: What if `grow()` increases the size of the array by 1?

Q: What if `grow()` doubles the size of the array?

Growing One At A Time

Suppose, for simplicity:

- The internal array starts with a capacity of 1
- The cost of writing/copying an array element is 1 operation

Q: What is the average cost of a sequence of n `pushes`?

A:

$$\underbrace{1}_{\text{store}} +$$

Growing One At A Time

Suppose, for simplicity:

- The internal array starts with a capacity of 1
- The cost of writing/copying an array element is 1 operation

Q: What is the average cost of a sequence of n `pushes`?

A:

$$\underbrace{1}_{\text{store}} + \underbrace{2}_{\text{grow \& store}} +$$

Growing One At A Time

Suppose, for simplicity:

- The internal array starts with a capacity of 1
- The cost of writing/copying an array element is 1 operation

Q: What is the average cost of a sequence of n `pushes`?

A:

$$\underbrace{1}_{\text{store}} + \underbrace{2}_{\text{grow \& store}} + \underbrace{3}_{\text{grow \& store}} +$$

Growing One At A Time

Suppose, for simplicity:

- The internal array starts with a capacity of 1
- The cost of writing/copying an array element is 1 operation

Q: What is the average cost of a sequence of n `pushes`?

A:

$$\underbrace{1}_{\text{store}} + \underbrace{2}_{\text{grow \& store}} + \underbrace{3}_{\text{grow \& store}} + \ldots +$$

Growing One At A Time

Suppose, for simplicity:

- The internal array starts with a capacity of 1
- The cost of writing/copying an array element is 1 operation

Q: What is the average cost of a sequence of n `pushes`?

A:

$$\underbrace{1}_{\text{store}} + \underbrace{2}_{\text{grow \& store}} + \underbrace{3}_{\text{grow \& store}} + \ldots + \underbrace{n}_{\text{grow \& store}}$$

Growing One At A Time

Suppose, for simplicity:

- The internal array starts with a capacity of 1
- The cost of writing/copying an array element is 1 operation

Q: What is the average cost of a sequence of n `pushes`?

A:

$$\underbrace{1}_{\text{store}} + \underbrace{2}_{\text{grow \& store}} + \underbrace{3}_{\text{grow \& store}} + \ldots + \underbrace{n}_{\text{grow \& store}} \in O(n^2)$$

Averaged out, that's $O(n)$

Doubling The Size Each Time

Suppose, for simplicity:

- The internal array starts with a capacity of 1
- The cost of writing/copying an array element is 1 operation

Q: What is the average cost of a sequence of n `pushes`?

A: The growths will happen less frequently, but will cost

$$\underbrace{1}_{\text{first growth}} +$$

Doubling The Size Each Time

Suppose, for simplicity:

- The internal array starts with a capacity of 1
- The cost of writing/copying an array element is 1 operation

Q: What is the average cost of a sequence of n `pushes`?

A: The growths will happen less frequently, but will cost

$$\underbrace{1}_{\text{first growth}} + \underbrace{2}_{\text{second growth}} +$$

Doubling The Size Each Time

Suppose, for simplicity:

- The internal array starts with a capacity of 1
- The cost of writing/copying an array element is 1 operation

Q: What is the average cost of a sequence of n `pushes`?

A: The growths will happen less frequently, but will cost

$$\underbrace{1}_{\text{first growth}} + \underbrace{2}_{\text{second growth}} + \underbrace{4}_{\text{third growth}} +$$

Doubling The Size Each Time

Suppose, for simplicity:

- The internal array starts with a capacity of 1
- The cost of writing/copying an array element is 1 operation

Q: What is the average cost of a sequence of n `pushes`?

A: The growths will happen less frequently, but will cost

$$\underbrace{1}_{\text{first growth}} + \underbrace{2}_{\text{second growth}} + \underbrace{4}_{\text{third growth}} + \underbrace{8}_{\text{fourth growth}} +$$

Doubling The Size Each Time

Suppose, for simplicity:

- The internal array starts with a capacity of 1
- The cost of writing/copying an array element is 1 operation

Q: What is the average cost of a sequence of n `pushes`?

A: The growths will happen less frequently, but will cost

$$\underbrace{1}_{\text{first growth}} + \underbrace{2}_{\text{second growth}} + \underbrace{4}_{\text{third growth}} + \underbrace{8}_{\text{fourth growth}} + \ldots +$$

Doubling The Size Each Time

Suppose, for simplicity:

- The internal array starts with a capacity of 1
- The cost of writing/copying an array element is 1 operation

Q: What is the average cost of a sequence of n `pushes`?

A: The growths will happen less frequently, but will cost

$$\underbrace{1}_{\text{first growth}} + \underbrace{2}_{\text{second growth}} + \underbrace{4}_{\text{third growth}} + \underbrace{8}_{\text{fourth growth}} + \ldots + \underbrace{2^i}_{\text{for } 2^i < n}$$

Doubling The Size Each Time

Suppose, for simplicity:

- The internal array starts with a capacity of 1
- The cost of writing/copying an array element is 1 operation

Q: What is the average cost of a sequence of n `pushes`?

A: The growths will happen less frequently, but will cost

$$\underbrace{1}_{\text{first growth}} + \underbrace{2}_{\text{second growth}} + \underbrace{4}_{\text{third growth}} + \underbrace{8}_{\text{fourth growth}} + \ldots + \underbrace{2^i}_{\text{for } 2^i < n} \in O(n)$$

Adding that to the cost of storing (1 for each `push`), that's still $O(n)$. Averaged out over n operations, that's $O(1)$.