Stacks

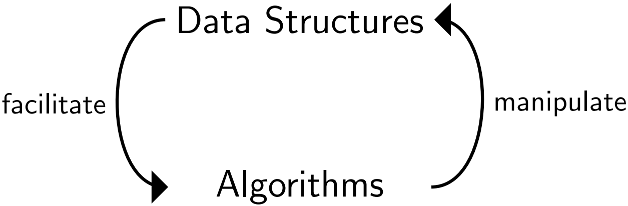
CS 240

Alex Vondrak

ajvondrak@csupomona.edu

Winter 2012

Data Structures


Data Structures

facilitate

manipulate

Algorithms

- In this class, we mostly study linear data structures
- Collections of items tend to have common operations
  - Adding elements
  - Removing elements
  - Querying for particular properties (membership, size, etc.)

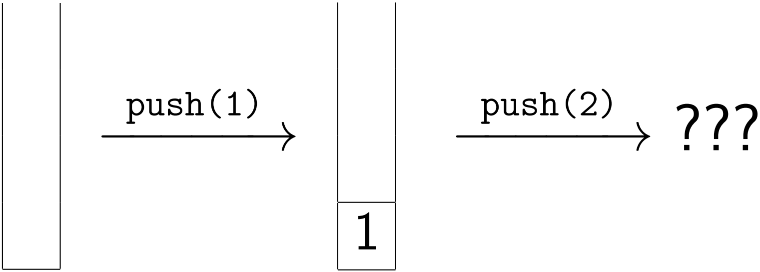
Stacks

Definition (Stack)

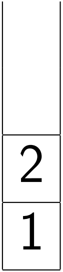
A stack is a linear data structure of items arranged from bottom to top. It's defined by three operations:

push:	To insert an item, you place it on top of the other items
pop:	To remove an item, you remove the top element
peek:	You may look at the top item of the stack without removing it; to look at anything underneath, you must pop the top

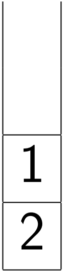
Multiple Choice Question



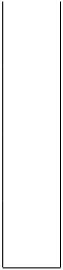
(A)



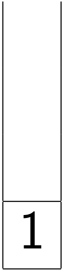
(B)



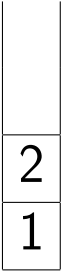
(C)



(D)



Multiple Choice Question



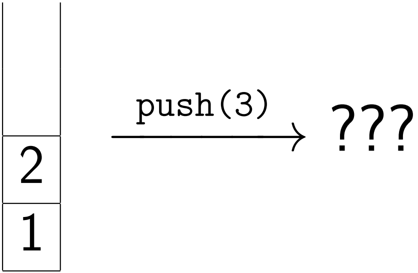
What would be the result of `peek()`?

(A) 1

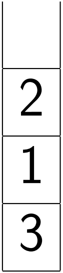
(B) 2

(C) Nothing

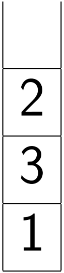
Multiple Choice Question



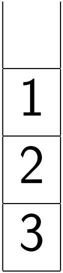
(A)



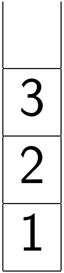
(B)



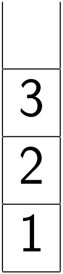
(C)



(D)



Multiple Choice Question



What would be the result of `peek()`?

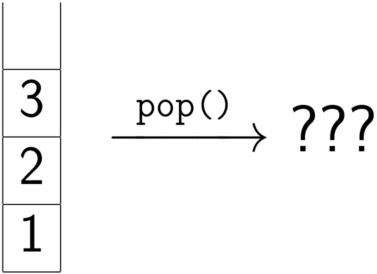
(A) 1

(B) 2

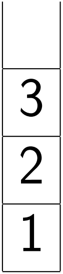
(C) 3

(D) Nothing

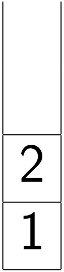
Multiple Choice Question



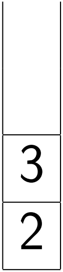
(A)



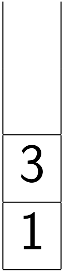
(B)



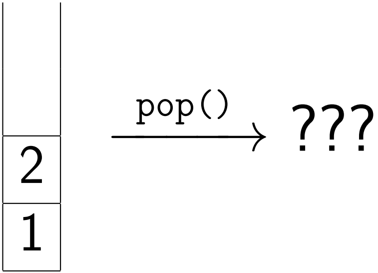
(C)



(D)

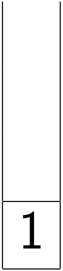


Multiple Choice Question

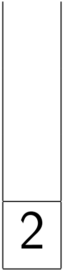


`pop()` → ???

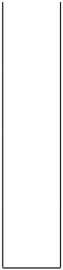
(A)



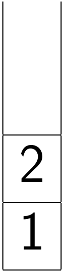
(B)



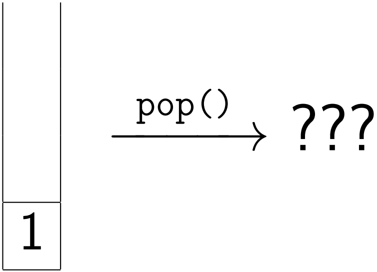
(C)



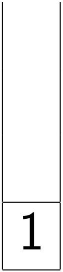
(D)



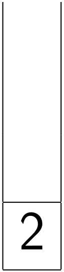
Multiple Choice Question



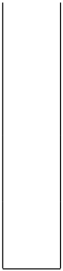
(A)



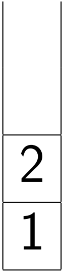
(B)



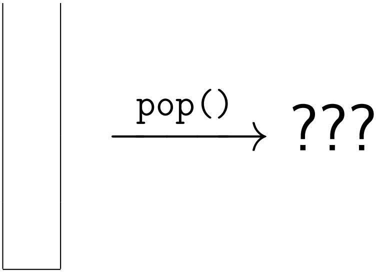
(C)



(D)



Multiple Choice Question



(A) An empty stack

(B) An error

(C) No error; the next `push` just won't change the stack

(D) None of the above

Multiple Choice Question

Is it possible to push “too many” items onto a stack?

(A) Yes: the computer may run out of memory

(B) No: conceptually, stacks don’t have a fixed size

(C) Both of the above

(D) None of the above

Error States

Definition (Underflow)

When a `pop` (or `peek`) is performed on an empty stack, the stack is said to be in an underflow state

Definition (Overflow)

When a `push` is performed on a full stack, the stack is said to be in an overflow state

Note

Conceptually, stack overflow needn't happen; in practice, it might

Multiple Choice Question

If you were to design a `Stack` class in Java that held **`int`**s, what might it look like?

What would the type of the `push` method be (ignoring errors)?

(A) `public int push()`

(B) `public int push(int item)`

(C) `public void push()`

(D) `public void push(int item)`

Multiple Choice Question

If you were to design a `Stack` class in Java that held **`int`**s, what might it look like?

What would the type of the pop method be (ignoring errors)?

(A) **`public int`** `pop()`

(B) **`public int`** `pop(`**`int`** `item)`

(C) **`public void`** `pop()`

(D) **`public void`** `pop(`**`int`** `item)`

Multiple Choice Question

If you were to design a `Stack` class in Java that held **`int`**`s`, what might it look like?

What `Exceptions` might **`public void`** `push(`**`int`** `item)` throw?

(A) **`throws`** `StackUnderflowException`

(B) **`throws`** `StackOverflowException`

(C) **`throws`** `StackUnderflowException, StackOverflowException`

(D) None

Multiple Choice Question

If you were to design a `Stack` class in Java that held **`int`**s, what might it look like?

What `Exceptions` might **`public int`** `pop()` throw?

(A) **`throws`** `StackUnderflowException`

(B) **`throws`** `StackOverflowException`

(C) **`throws`** `StackUnderflowException, StackOverflowException`

(D) None

Multiple Choice Question

If you were to design a `Stack` class in Java that held **`int`**`s`, what might it look like?

What would the type of the `peek` method be?

(A) `public int peek(int item)`

(B) `public int peek(int item) throws StackUnderflowException`

(C) `public int peek()`

(D) `public int peek() throws StackUnderflowException`

Interfaces

```
interface Stack {
    public void push(int item);

    public int pop() throws StackUnderflowException;

    public int peek() throws StackUnderflowException;
}

class SomeStackImplementation implements Stack {
    /* must implement all the methods */
}
```

Reverse Polish Notation (RPN)

Definition

Normally, we write math operators in infix notation:

`A + B`

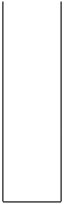
In postfix (or Reverse Polish) notation, we write:

`A B +`

Stack-Based Evaluation

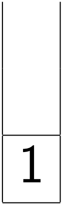
- If we see a number, push it to the data stack
- If we see an operator, pop the operands and push the result

`1 2 + 3 * 4 -`

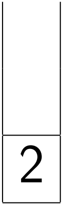


$\xrightarrow{1}$???

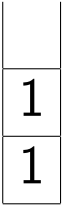
(A)



(B)



(C)

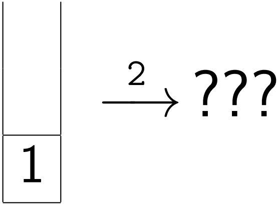


(D)

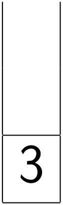
Stack-Based Evaluation

- If we see a number, push it to the data stack
- If we see an operator, pop the operands and push the result

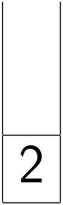
`1 2 + 3 * 4 -`



(A)

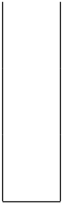


(B)



(C)

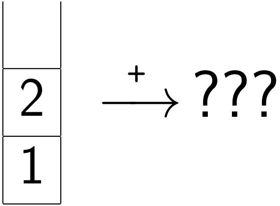
(D)



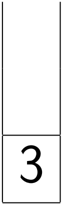
Stack-Based Evaluation

- If we see a number, push it to the data stack
- If we see an operator, pop the operands and push the result

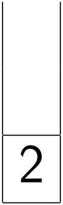
`1 2 + 3 * 4 -`



(A)

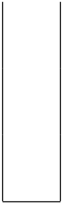


(B)



(C)

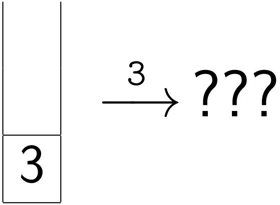
(D)



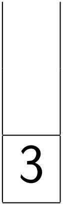
Stack-Based Evaluation

- If we see a number, push it to the data stack
- If we see an operator, pop the operands and push the result

`1 2 + 3 * 4 -`

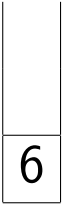


(A)

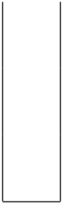


(B)

(C)



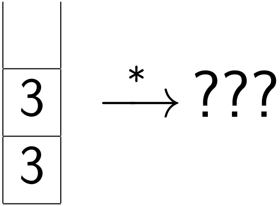
(D)



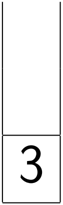
Stack-Based Evaluation

- If we see a number, push it to the data stack
- If we see an operator, pop the operands and push the result

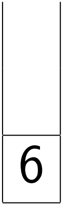
`1 2 + 3 * 4 -`



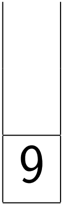
(A)



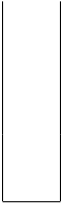
(B)



(C)



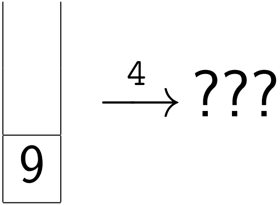
(D)



Stack-Based Evaluation

- If we see a number, push it to the data stack
- If we see an operator, pop the operands and push the result

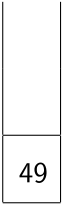
`1 2 + 3 * 4 -`



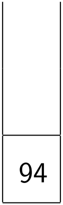
(A)

(B)

(C)



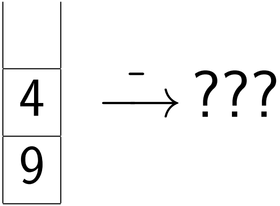
(D)



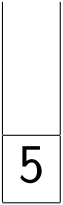
Stack-Based Evaluation

- If we see a number, push it to the data stack
- If we see an operator, pop the operands and push the result

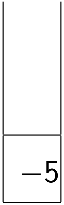
`1 2 + 3 * 4 -`



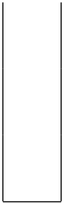
(A)



(B)



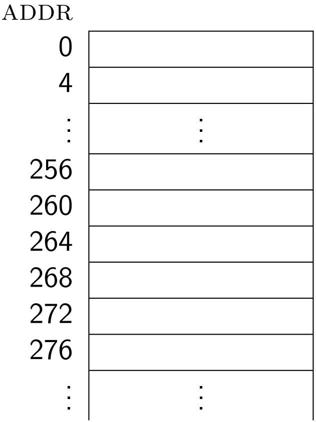
(C)



(D) Depends

Arrays

```
int[] array = new int[3];
for (int i = 0; i < array.length; i++) {
   array[i] = 100;
}
```



How many bits are in a byte?

(A) 2

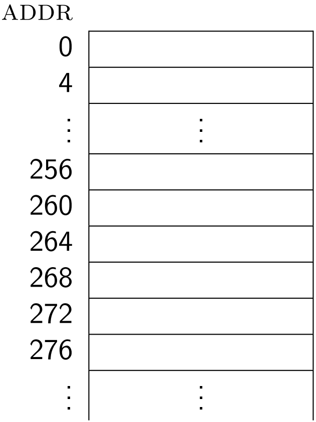
(B) 4

(C) 8

(D) 16

Arrays

```
int[] array = new int[3];
for (int i = 0; i < array.length; i++) {
   array[i] = 100;
}
```



How many bytes are in a 32-bit word?

(A) 2

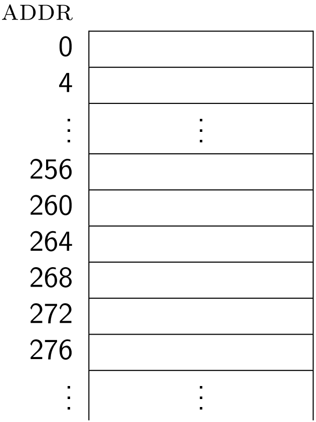
(B) 4

(C) 8

(D) 16

Arrays

```
int[] array = new int[3];
for (int i = 0; i < array.length; i++) {
   array[i] = 100;
}
```



Can array fit in a word?

(A) Yes

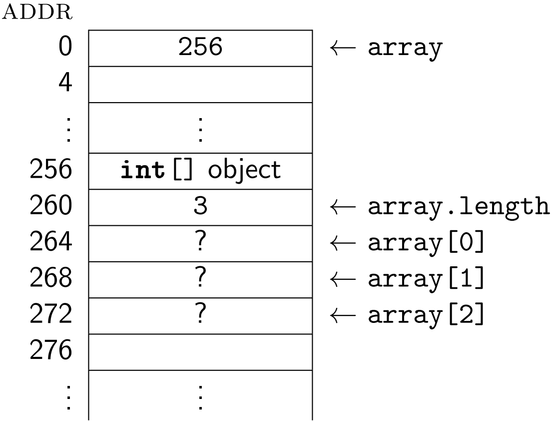
(B) No

(C) Depends on its length

(D) None of the above

Arrays

```
int[] array = new int[3];
for (int i = 0; i < array.length; i++) {
   array[i] = 100;
}
```



ADDR		
0	256	← `array`
4		
⋮	⋮	
256	`int[]` object	
260	3	← `array.length`
264	?	← `array[0]`
268	?	← `array[1]`
272	?	← `array[2]`
276		
⋮	⋮	

What are the initial values stored in `array`?

(A) **`null`**

(B) `0`

(C) Nothing is stored

(D) `NaN`

Arrays

```
int[] array = new int[3];
for (int i = 0; i < array.length; i++) {
    array[i] = 100;
}
```

ADDR		
0	256	← `array`
4		
⋮	⋮	
256	**`int[]`** object	
260	3	← `array.length`
264	0	← `array[0]`
268	0	← `array[1]`
272	0	← `array[2]`
276		
⋮	⋮	

Can `i` fit in a word?

(A) Yes

(B) No

(C) Depends on what number it is

(D) None of the above

Arrays

```
int[] array = new int[3];
for (int i = 0; i < array.length; i++) {
    array[i] = 100;
}
```

ADDR		
0	256	← `array`
4	0	← `i`
⋮	⋮	
256	`int[]` object	
260	3	← `array.length`
264	100	← `array[0]`
268	0	← `array[1]`
272	0	← `array[2]`
276		
⋮	⋮	

How many bytes away from 260 is `array[0]`?

(A) 1

(B) 2

(C) 4

(D) 8

Arrays

```
int[] array = new int[3];
for (int i = 0; i < array.length; i++) {
   array[i] = 100;
}
```

ADDR		
0	256	← `array`
4	1	← `i`
⋮	⋮	
256	**int**[] object	
260	3	← `array.length`
264	100	← `array[0]`
268	100	← `array[1]`
272	0	← `array[2]`
276		
⋮	⋮	

How many bytes away from 260 is `array[1]`?

(A) 2

(B) 4

(C) 8

(D) 12

Arrays

```
int[] array = new int[3];
for (int i = 0; i < array.length; i++) {
   array[i] = 100;
}
```

ADDR		
0	256	← `array`
4	2	← `i`
⋮	⋮	
256	**`int`**`[]` object	
260	3	← `array.length`
264	100	← `array[0]`
268	100	← `array[1]`
272	100	← `array[2]`
276		
⋮	⋮	

How many bytes away from 260 is `array[2]`?

(A) 2

(B) 4

(C) 8

(D) 12

Arrays

```
int[] array = new int[3];
for (int i = 0; i < array.length; i++) {
    array[i] = 100;
}
```

ADDR		
0	256	← `array`
4	3	← `i`
⋮	⋮	
256	`int[]` object	
260	3	← `array.length`
264	100	← `array[0]`
268	100	← `array[1]`
272	100	← `array[2]`
276		
⋮	⋮	

What is the relationship between an array index and its address?

(A) `addr=base+idx`

(B) `addr=base*idx`

(C) `addr=base+4*idx`

(D) `addr=base+base*idx`

Arrays

```
int[] array = new int[3];
for (int i = 0; i < array.length; i++) {
   array[i] = 100;
}
```

ADDR		
0	256	← `array`
4	3	← `i`
⋮	⋮	
256	**int**[] object	
260	3	← `array.length`
264	100	← `array[0]`
268	100	← `array[1]`
272	100	← `array[2]`
276		
⋮	⋮	

What is O the running time of an array access in terms of its size?

(A) $O(1)$

(B) $O(n)$

(C) $O(\log n)$

(D) None of the above

ArrayStack

```
class ArrayStack implements Stack {
   // Idea: use an array to implement a stack
}
```

What fields should we have?

ArrayStack

```
class ArrayStack implements Stack {
   private static final int INITIAL_CAPACITY = 10;
   private int[] data;
   private int top;
}
```

What methods should we have?

Constructor

```
class ArrayStack implements Stack {
    private static final int INITIAL_CAPACITY = 10;
    private int[] data;
    private int top;

    public ArrayStack() {
        // ...
    }
}
```

What should this method do?

(A) Set top to `INITIAL_CAPACITY`

(B) Set top to `data.length`

(C) Set data to an array of top elements

(D) Set data to an array of `INITIAL_CAPACITY` elements

Constructor

```
class ArrayStack implements Stack {
    private static final int INITIAL_CAPACITY = 10;
    private int[] data;
    private int top;

    public ArrayStack() {
        this.data = new int[this.INITIAL_CAPACITY];
        // ...
    }
}
```

Where should `this.top` start?

(A) `-1`

(B) `0`

(C) `data.length`

(D) `INITIAL_CAPACITY-1`

Constructor

```
class ArrayStack implements Stack {
   private static final int INITIAL_CAPACITY = 10;
   private int[] data;
   private int top;

   public ArrayStack() {
      this.data = new int[this.INITIAL_CAPACITY];
      this.top = -1;
   }

   // ...
}
```

Helper Methods

```
class ArrayStack implements Stack {
    // ...

    public int size() {
        // ?
    }

    // ...
}
```

How should we calculate the size of the stack?

(A) `this.INITIAL_CAPACITY`

(B) `this.top`

(C) `this.top + 1`

(D) `this.data.length`

Helper Methods

```
class ArrayStack implements Stack {
    // ...

    public int size() {
        return this.top + 1;
    }

    public boolean isEmpty() {
        // ?
    }
}
```

Which of the following is the best way to check if the stack is empty?

(A) `this.top == -1`

(B) `this.top < 0`

(C) `this.size() == 0`

(D) `this.size() <= 0`

Helper Methods

```
class ArrayStack implements Stack {
    // ...

    public int size() {
        return this.top + 1;
    }

    public boolean isEmpty() {
        return this.size() == 0;
    }

    // ...
}
```

peek

```
public int peek() throws StackUnderflowException {
    // ?
}
```

Where does `StackUnderflowException` come from?

(A) It's a class in a Java library

(B) It doesn't matter; the code will still compile

(C) Nowhere; we need to define it ourselves

(D) It's automatically defined when we declare it in the **`throws`** clause

Detour: `StackUnderflowException`

```
class StackUnderflowException extends /* ? */ {
    // ...
}
```

What should be the parent class of `StackUnderflowException`?

(A) `RuntimeException`

(B) `Exception`

(C) `Error`

(D) `Throwable`

Detour: `StackUnderflowException`

```
class StackUnderflowException extends RuntimeException {
    public StackUnderflowException() {
        super("Stack underflow.");
    }
}
```

What does **`super`** refer to?

- (A) The constructor of the parent class
- (B) The constructor of the `Object` class
- (C) The constructor of the `StackUnderflowException` class
- (D) The constructor of the `Throwable` class

peek

```
public int peek() throws StackUnderflowException {
   return /* ? */;
}
```

What value should peek return?

(A) `this.data[0]`

(B) `this.data[this.data.length - 1]`

(C) `this.data[this.size()]`

(D) `this.data[this.top]`

peek

```
public int peek() throws StackUnderflowException {
   return this.data[this.top];
}
```

When might this definition cause problems?

(A) It won't

(B) When the stack is empty

(C) When the stack is full

(D) When the user is stupid

peek

```
public int peek() throws StackUnderflowException {
    if (this.isEmpty()) {
        // ...
    }
    return this.data[this.top];
}
```

What should we do when the stack is empty?

(A) Throw an `ArrayIndexOutOfBoundsException`

(B) Throw a new `IndexOutOfBoundsException`

(C) Throw a `StackUnderflowException`

(D) Print out an error message

peek

```
public int peek() throws StackUnderflowException {
    if (this.isEmpty()) {
        throw new StackUnderflowException();
    }
    return this.data[this.top];
}
```

pop

```
public int pop() throws StackUnderflowException {
    // ...
}
```

What makes pop different from `peek`?

(A) We must modify the contents of the stack

(B) We must decrement `this.top`

(C) We can't throw a `StackUnderflowException`

(D) Nothing; just return `this.peek()`

pop

```
public int pop() throws StackUnderflowException {
    int result = this.peek();
    // ...
}
```

Do we need to handle the case where the stack's empty?

(A) Yes: `peek` might throw an Excpetion

(B) Yes: if the stack's empty, we shouldn't call `peek`

(C) No: `peek` throws the Exception for us

(D) No: `pop` won't be called on empty stacks

pop

```
public int pop() throws StackUnderflowException {
    int result = this.peek();
    // ...
}
```

What do we do with the top element of the stack (i.e., `this.data[this.top]`)?

(A) Overwrite it with **`null`**

(B) Overwrite it with 0

(C) Overwrite it with −1

(D) Nothing

pop

```
public int pop() throws StackUnderflowException {
    int result = this.peek();
    // ...
}
```

What do we do with **this**.top now?

(A) **this**.top--;

(B) **this**.top++;

(C) **this**.top = 0;

(D) **this**.top = **this**.size() - 1;

pop

```
public int pop() throws StackUnderflowException {
    int result = this.peek();
    this.top--;
    // ...
}
```

What do we have left to do?

(A) Nothing

(B) Return `result`

(C) Check if the stack is empty

(D) Print out `result`

pop

```
public int pop() throws StackUnderflowException {
    int result = this.peek();
    this.top--;
    return result;
}
```

push

```
public void push(int item) {
    // ...
}
```

What happens to `this.top` as we push?

(A) It increments by 1

(B) It decrements by 1

(C) Nothing

push

```
public void push(int item) {
    this.top++;
    // ...
}
```

What happens to `this.data`?

(A) `this.data[this.top] = item`

(B) `this.data[this.top + 1] = item`

(C) `this.data[0] = item`

(D) None of the above

push

```
public void push(int item) {
    this.top++;
    this.data[this.top] = item;
}
```

What if `this.top > this.data.length`?

(A) Can't happen

(B) Let the Java Runtime worry about that

(C) Throw a `StackOverflowException`

(D) We should allocate a new, bigger array

push

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

`grow`

What should be the type of the `grow` method?

(A) `public int grow()`

(B) `public void grow()`

(C) `private void grow()`

(D) `private void grow(int byHowMuch)`

grow

```
private void grow() {
   int[] biggerArray = new int[2 * this.data.length + 1];
   for (int i = 0; i < this.data.length; i++) {
      biggerArray[i] = this.data[i];
   }
   this.data = biggerArray;
}
```

Efficiency

```
public int size() {
    return this.top + 1;
}
```

What is the worst-case running time of `size` in terms of O of a function of the size of the stack, n?

(A) $O(1)$

(B) $O(\log n)$

(C) $O(n)$

(D) None of the above

Efficiency

```
public boolean isEmpty() {
   return this.size() == 0;
}
```

What is the worst-case running time of `isEmpty` in terms of O of a function of the size of the stack, n?

(A) $O(1)$

(B) $O(\log n)$

(C) $O(n)$

(D) None of the above

Efficiency

```
public int peek() throws StackUnderflowException {
    if (this.isEmpty()) {
        throw new StackUnderflowException();
    }
    return this.data[this.top];
}
```

What is the worst-case running time of `peek` in terms of O of a function of the size of the stack, n?

(A) $O(1)$

(B) $O(\log n)$

(C) $O(n)$

(D) None of the above

Efficiency

```
public int pop() throws StackUnderflowException {
    int result = this.peek();
    this.top--;
    return result;
}
```

What is the worst-case running time of pop in terms of O of a function of the size of the stack, n?

(A) $O(1)$

(B) $O(\log n)$

(C) $O(n)$

(D) None of the above

Efficiency

```
private void grow() {
   int[] biggerArray = new int[2 * this.data.length + 1];
   for (int i = 0; i < this.data.length; i++) {
      biggerArray[i] = this.data[i];
   }
   this.data = biggerArray;
}
```

What is the worst-case running time of `grow` in terms of O of a function of the size of the stack, n?

(A) $O(1)$

(B) $O(\log n)$

(C) $O(n)$

(D) None of the above

Efficiency

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

What is the worst-case running time of `push` in terms of O of a function of the size of the stack, n?

(A) $O(1)$

(B) $O(\log n)$

(C) $O(n)$

(D) None of the above

Amortized Analysis

Definition

A method of algorithm analysis that considers the entire sequence of operations in a program

Pros

- If a costly operation occurs infrequently, we'd like to count the expected "worst case" running time, not the *absolute* worst
- Thus, gives us a "fairer" impression of expected running times

Cons

- More difficult to analyze
- Often confused with average case analysis—the distinction is important!

What If. . . ?

Let's look at a sequence of `push` operations onto the same `ArrayStack`. However, instead of our "double the size" `grow`, what if we had

```
private void grow() {
   int[] biggerArray = new int[this.data.length + 1];
   for (int i = 0; i < this.data.length; i++) {
      biggerArray[i] = this.data[i];
   }
   this.data = biggerArray;
}
```

Multiple Choice Question

Suppose we have a sequence of n `push` operations to our `ArrayStack`. For simplicity, assume:

- `INITIAL_CAPACITY` is 1
- The cost of writing/copying an array element is 1 operation
- Other operations cost 0

How many operations are performed in a sequence of $n = 1$ push(es)?

(A) 1

(B) 2

(C) 3

(D) 4

Multiple Choice Question

Suppose we have a sequence of n `push` operations to our `ArrayStack`. For simplicity, assume:

- `INITIAL_CAPACITY` is 1
- The cost of writing/copying an array element is 1 operation
- Other operations cost 0

How many operations are performed in a sequence of $n = 2$ push(es)?

(A) 1+1

(B) 1+2

(C) 1+3

(D) 1+4

Multiple Choice Question

Suppose we have a sequence of n `push` operations to our `ArrayStack`. For simplicity, assume:

- `INITIAL_CAPACITY` is 1
- The cost of writing/copying an array element is 1 operation
- Other operations cost 0

How many operations are performed in a sequence of $n = 3$ push(es)?

(A) 1+2+1

(B) 1+2+2

(C) 1+2+3

(D) 1+2+4

Multiple Choice Question

Suppose we have a sequence of n `push` operations to our `ArrayStack`. For simplicity, assume:

- `INITIAL_CAPACITY` is 1
- The cost of writing/copying an array element is 1 operation
- Other operations cost 0

Consider an arbitrary number of pushes, n. What is the cost of the first push in this sequence?

(A) 1
(B) 2
(C) 3
(D) 4

Multiple Choice Question

Suppose we have a sequence of n `push` operations to our `ArrayStack`. For simplicity, assume:

- `INITIAL_CAPACITY` is 1
- The cost of writing/copying an array element is 1 operation
- Other operations cost 0

Consider an arbitrary number of pushes, n. What is the cost of the second push in this sequence?

(A) 1

(B) 2

(C) 1+2

(D) None of the above

Multiple Choice Question

Suppose we have a sequence of n `push` operations to our `ArrayStack`. For simplicity, assume:

- `INITIAL_CAPACITY` is 1
- The cost of writing/copying an array element is 1 operation
- Other operations cost 0

Consider an arbitrary number of pushes, n. What is the cost of the third push in this sequence?

(A) 1

(B) 2

(C) 3

(D) 1+2+3

Multiple Choice Question

Suppose we have a sequence of n `push` operations to our `ArrayStack`. For simplicity, assume:

- `INITIAL_CAPACITY` is 1
- The cost of writing/copying an array element is 1 operation
- Other operations cost 0

Consider an arbitrary number of pushes, n. In general, what's the cost of the i^{th} push in this sequence?

(A) i

(B) $1 + 2 + \cdots + i$

(C) n

(D) 1

Multiple Choice Question

Suppose we have a sequence of n `push` operations to our `ArrayStack`. For simplicity, assume:

- `INITIAL_CAPACITY` is 1
- The cost of writing/copying an array element is 1 operation
- Other operations cost 0

Consider an arbitrary number of pushes, n. In general, what's the total cost of this sequence?

(A) $1 + 2 + \cdots + i$

(B) $1 + 2 + \cdots + n$

(C) n

(D) i

Multiple Choice Question

Suppose we have a sequence of n `push` operations to our `ArrayStack`. For simplicity, assume:

- `INITIAL_CAPACITY` is 1
- The cost of writing/copying an array element is 1 operation
- Other operations cost 0

Consider an arbitrary number of pushes, n. In general, what's the average cost of each `push` in this sequence?

(A) $(1 + 2 + \cdots + n)/i$

(B) $(1 + 2 + \cdots + n)/n$

(C) n^2

(D) 1

Multiple Choice Question

To get an idea of what

$$\frac{1 + 2 + \cdots + n}{n}$$

is, we'll show that

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

How should we prove this?

(A) Axiomatically

(B) Recursively

(C) Inductively

(D) Productively

Multiple Choice Question

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

Proof (by induction on n).

Base Case ($n =?$): …

Inductive Step: …

☐

What is the base base?

(A) $n = 1$

(B) $n = 0$

(C) $n = i$

(D) $n = k + 1$

Multiple Choice Question

Proof (by induction on n).

Base Case ($n = 1$): $\sum_{i=1}^{1} i =? \quad = \quad 1 = \frac{n(n+1)}{2}$

Inductive Step: …

☐

What is $\sum_{i=1}^{1} i$?

(A) 1

(B) $1 + 1$

(C) 0

(D) None of the above

Multiple Choice Question

Proof (by induction on n).

Base Case ($n = 1$): $\sum_{i=1}^{1} i = 1 \quad = \quad 1 = \frac{1(1+1)}{2}$

Inductive Step: Assume equality holds for $n = k$. Show that equality holds at $n = ?$

☐

What do we show in the inductive step?

(A) The equality holds at $n = n + 1$

(B) The equality holds at $n = k + 1$

(C) The equality holds at $n = n - 1$

(D) The equality holds at $n = k - 1$

Multiple Choice Question

Proof (By induction on n).

Base Case ($n = 1$): $\sum_{i=1}^{1} i = 1 \quad = \quad 1 = \frac{1(1+1)}{2}$

Inductive Step: Assume equality holds for $n = k$. Show that equality holds at $n = k + 1$.

$$\sum_{i=1}^{k} i = \frac{k(k+1)}{2} \qquad \text{(Inductive Hypothesis)}$$

☐

What should we do now?

(A) Add $k + 1$ to both sides

(B) Break the $\sum$ into $k + \sum_{i=1}^{k-1} i$

(C) Multiply both sides by 2

(D) Plug in $k + 1$ for n

Multiple Choice Question

Proof (By induction on n).

Base Case ($n = 1$): $\sum_{i=1}^{1} i = 1 \quad = \quad 1 = \frac{1(1+1)}{2}$

Inductive Step: Assume equality holds for $n = k$. Show that equality holds at $n = k + 1$.

$$\sum_{i=1}^{k} i = \frac{k(k+1)}{2} \qquad \text{(Inductive Hypothesis)}$$

$$(k+1) + \sum_{i=1}^{k} i = (k+1) + \frac{k(k+1)}{2}$$

☐

What is the left side equal to?

(A) $\sum_{i=1}^{k} i$

(B) $\sum_{i=0}^{k} i$

(C) $\sum_{i=1}^{k+1} i$

(D) $\sum_{i=1}^{k} (i+1)$

QED

Proof (By induction on n).

Base Case ($n = 1$): $\sum_{i=1}^{1} i = 1 \quad = \quad 1 = \frac{1(1+1)}{2}$

Inductive Step: Assume equality holds for $n = k$. Show that equality holds at $n = k + 1$.

$$\sum_{i=1}^{k} i = \frac{k(k+1)}{2} \qquad \text{(Inductive Hypothesis)}$$

$$(k+1) + \sum_{i=1}^{k} i = (k+1) + \frac{k(k+1)}{2}$$

$$\sum_{i=1}^{k+1} i = \frac{(k+1)((k+1)+1)}{2}$$

□

Multiple Choice Question

So, now we know

$$\frac{1 + 2 + \cdots + n}{n}$$

is the same as

$$\frac{n(n+1)/2}{n}$$

which is the same as

$$\frac{n+1}{2}$$

The average running time of a `push` (with our grow-by-1 strategy) is thus

(A) $O(1)$

(B) $O(n)$

(C) $O(n^2)$

(D) $O(\log n)$

Multiple Choice Question

Now suppose we had our "double the size" `grow`. For simplicity, assume:

- `INITIAL_CAPACITY` is 1
- The cost of writing/copying an array element is 1 operation
- Other operations cost 0
- `grow` exactly doubles the size of the array (rather than having that +1 at the end)

In a sequence of n `push`es, how many operations are used to write the new data into the array (not for growing)?

(A) 1

(B) n

(C) $1 + 2 + \cdots + n$

(D) Can't be determined

Multiple Choice Question

Now suppose we had our "double the size" `grow`. For simplicity, assume:

- `INITIAL_CAPACITY` is 1
- The cost of writing/copying an array element is 1 operation
- Other operations cost 0
- `grow` exactly doubles the size of the array (rather than having that +1 at the end)

How often do we call `grow`?

(A) Every time we call `push`

(B) Every second call to `push`

(C) Every n calls to `push`

(D) Only on certain calls to `push` (varying)

Multiple Choice Question

Now suppose we had our "double the size" `grow`. For simplicity, assume:

- `INITIAL_CAPACITY` is 1
- The cost of writing/copying an array element is 1 operation
- Other operations cost 0
- `grow` exactly doubles the size of the array (rather than having that +1 at the end)

How many operations are used for the first `grow`?

(A) 1
(B) 2
(C) 3
(D) 4

Multiple Choice Question

Now suppose we had our "double the size" `grow`. For simplicity, assume:

- `INITIAL_CAPACITY` is 1
- The cost of writing/copying an array element is 1 operation
- Other operations cost 0
- `grow` exactly doubles the size of the array (rather than having that +1 at the end)

How many operations are used for the second `grow`?

(A) 1

(B) 2

(C) 3

(D) 4

Multiple Choice Question

Now suppose we had our "double the size" `grow`. For simplicity, assume:

- `INITIAL_CAPACITY` is 1
- The cost of writing/copying an array element is 1 operation
- Other operations cost 0
- `grow` exactly doubles the size of the array (rather than having that +1 at the end)

How many operations are used for the third `grow`?

(A) 1

(B) 2

(C) 3

(D) 4

Multiple Choice Question

Now suppose we had our "double the size" `grow`. For simplicity, assume:

- `INITIAL_CAPACITY` is 1
- The cost of writing/copying an array element is 1 operation
- Other operations cost 0
- `grow` exactly doubles the size of the array (rather than having that +1 at the end)

How many operations are used for the i^{th} `grow`?

(A) $2 \times i$

(B) 2^i

(C) $\log_2 i$

(D) None of the above

Multiple Choice Question

So, we have roughly

$$\sum_{i=0}^{\log_2 n} 2^i = 2n - 1$$

operations total for the `grows` in a sequence of n `pushes`.

How many operations are there total for the `n` pushes?

(A) n

(B) $2n - 1$

(C) $2n$

(D) $3n - 1$

Multiple Choice Question

Averaged out over n operations, the cost of `push` is about

$$\frac{3n}{n} = 3$$

Thus, our “double the size” implementation of `push` is

(A) $O(1)$

(B) $O(n)$

(C) $O(n^2)$

(D) All of the above

Multiple Choice Question

Is there any reason that a stack should only have integers?

(A) Yes

(B) No

Converting Infix To Postfix

- If you see a left parenthesis, push it onto the stack
- If you see a number, write it to the output
- If you see an operator, push it onto the stack
- Otherwise, next symbol should be a right parenthesis, and the top of the stack should be an operator
  - Pop the operator and write it to the output
  - Top of the stack should be a left parenthesis, so pop and discard
- At the end of the input, stack should be empty

Examples

- `((1+2)*3)`
- `((1+2)*(3+4))`