CS 106 INTRODUCTION TO DATA STRUCTURES

SPRING 2020

PROF. SARA MATHIESON

HAVERFORD COLLEGE

ADMIN

- Video on (if possible)
- Microphone off (except questions, discussion)
    - Feel free to just ask questions, don’t need to raise your hand

- Many people are now 10-13 hours ahead, some are 3 hours behind

- Tomorrow: google sign in sheet for lab, then use the google form to join the queue. We will invite you to the zoom meeting when you’re first on the queue.

- Tomorrow: we will have labs at 8:30am, 9:30am, 10:30am, and 11:30am

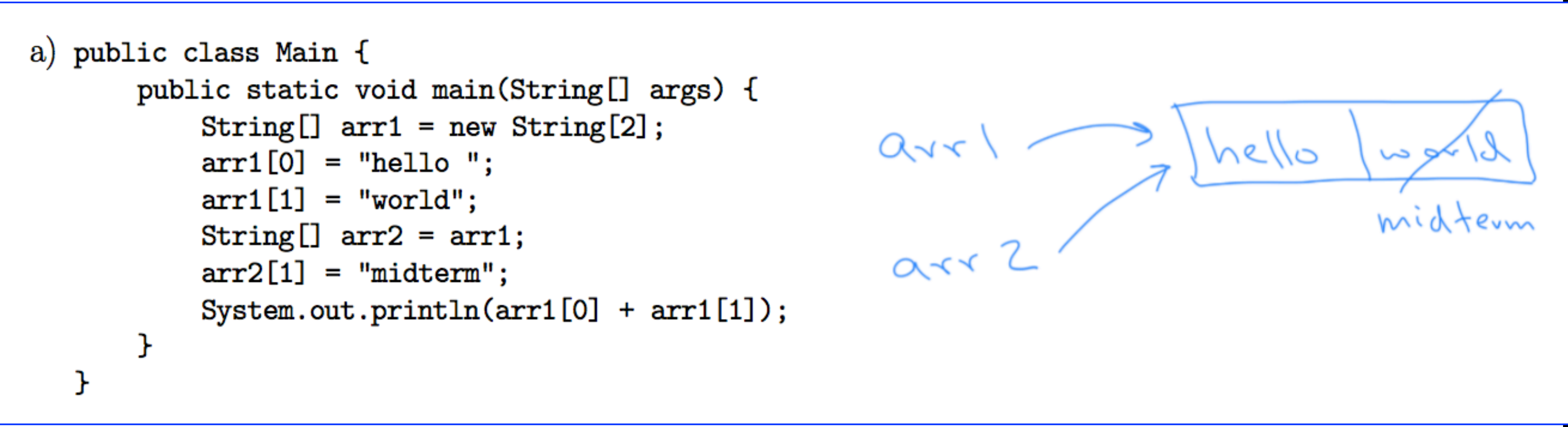
MAR 19 OUTLINE

- Common issues on Midterm 1
- Continue graphs
- Graph implementations

MAR 19 OUTLINE

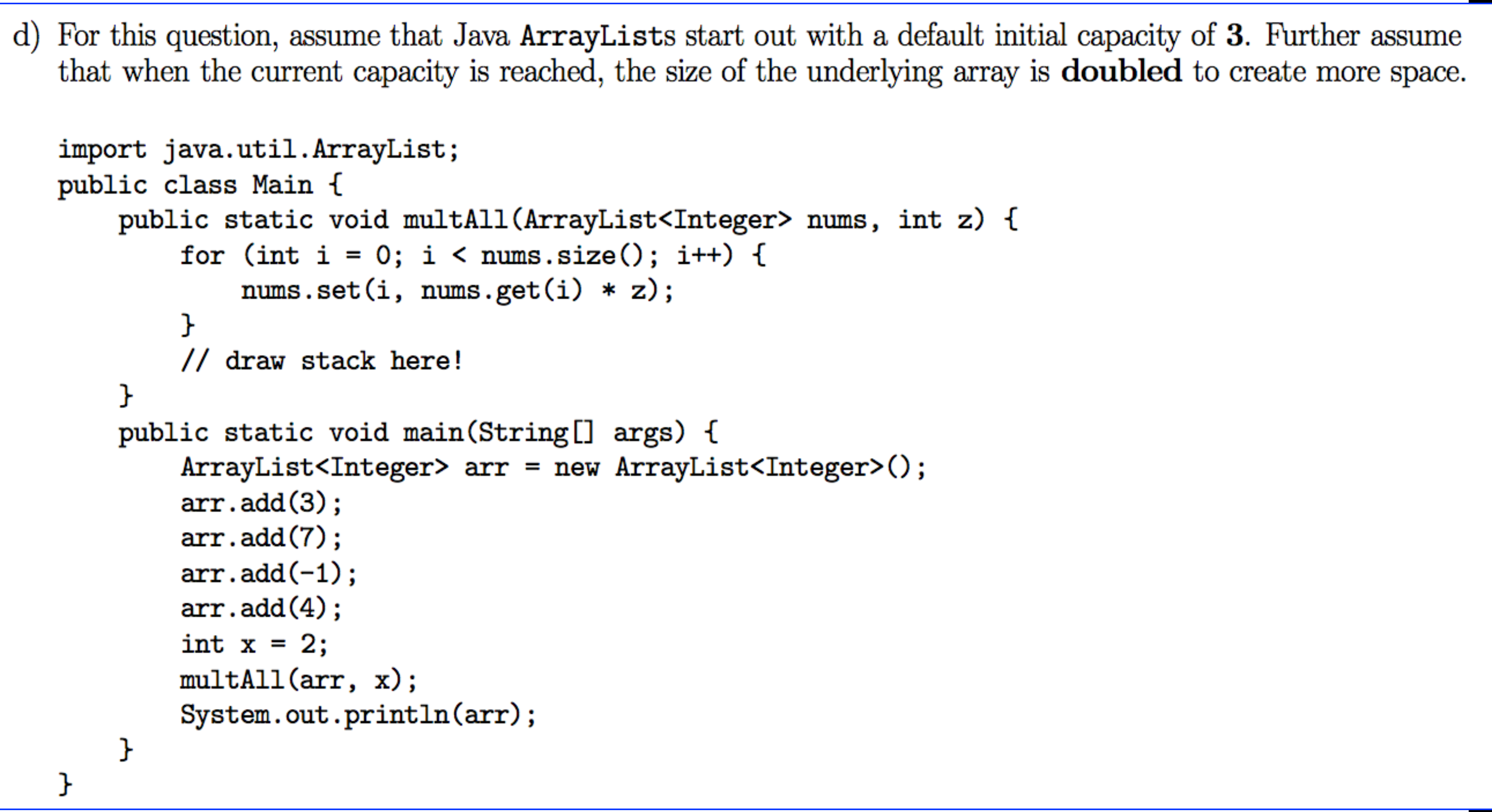
- Common issues on Midterm 1
- Continue graphs
- Graph implementations

MIDTERM: PART 1



```
a) public class Main {
       public static void main(String[] args) {
           String[] arr1 = new String[2];
           arr1[0] = "hello ";
           arr1[1] = "world";
           String[] arr2 = arr1;
           arr2[1] = "midterm";
           System.out.println(arr1[0] + arr1[1]);
       }
   }
```

MIDTERM: PART 1

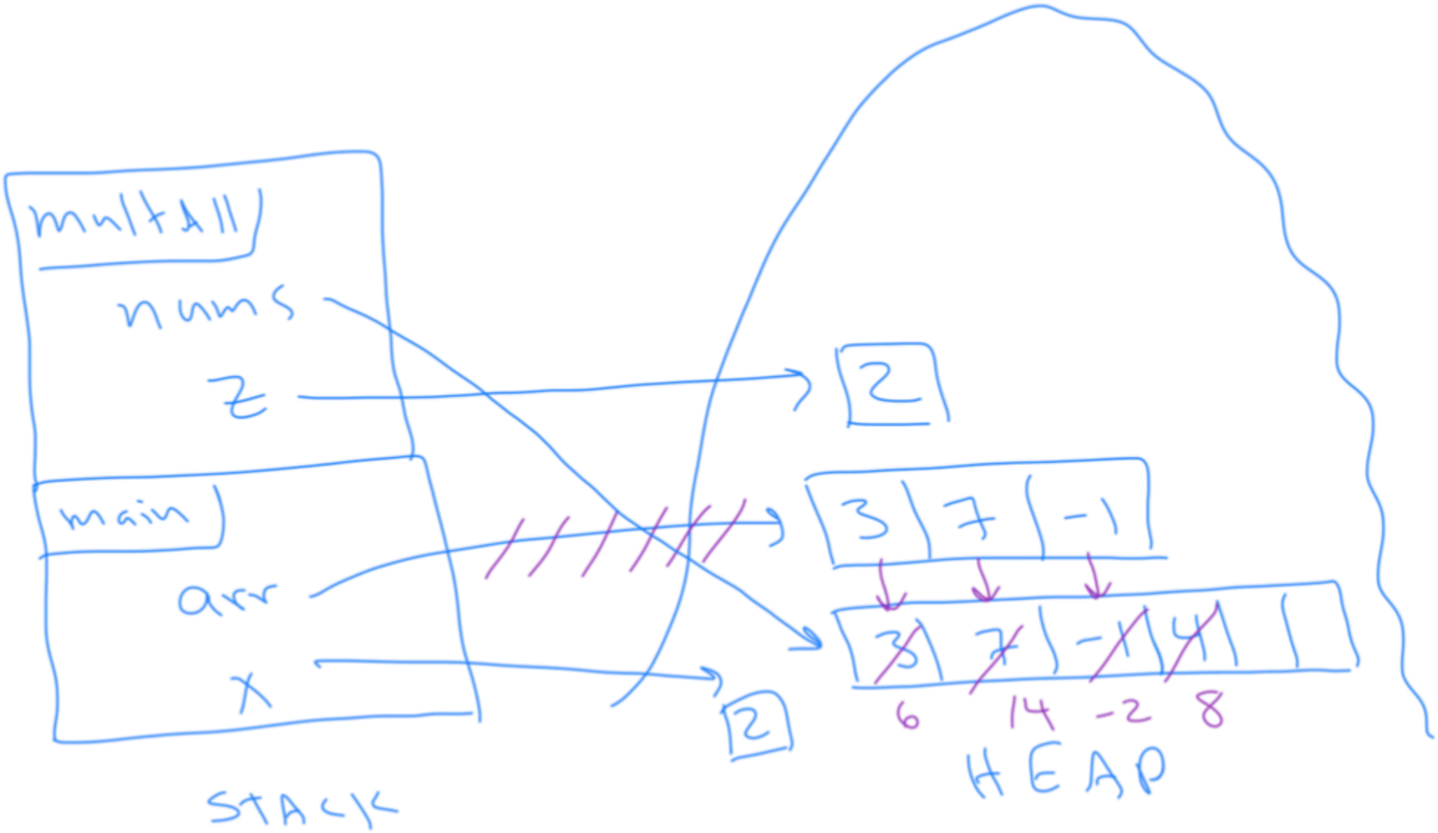


d) For this question, assume that Java **ArrayLists** start out with a default initial capacity of **3**. Further assume that when the current capacity is reached, the size of the underlying array is **doubled** to create more space.

```
import java.util.ArrayList;
public class Main {
    public static void multAll(ArrayList<Integer> nums, int z) {
        for (int i = 0; i < nums.size(); i++) {
            nums.set(i, nums.get(i) * z);
        }
        // draw stack here!
    }
    public static void main(String[] args) {
        ArrayList<Integer> arr = new ArrayList<Integer>();
        arr.add(3);
        arr.add(7);
        arr.add(-1);
        arr.add(4);
        int x = 2;
        multAll(arr, x);
        System.out.println(arr);
    }
}
```

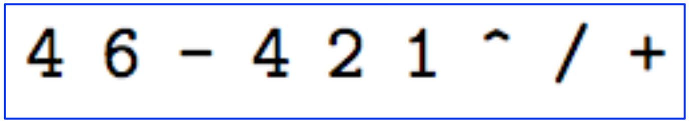
MIDTERM: PART 1

i) Draw the function call stack and heap as it would look at the line "`// draw stack here!`". You may ignore the loop variable `i`, and assume the user clicked "Run" to start the program. Make sure to show how `arr` changes as elements are added.

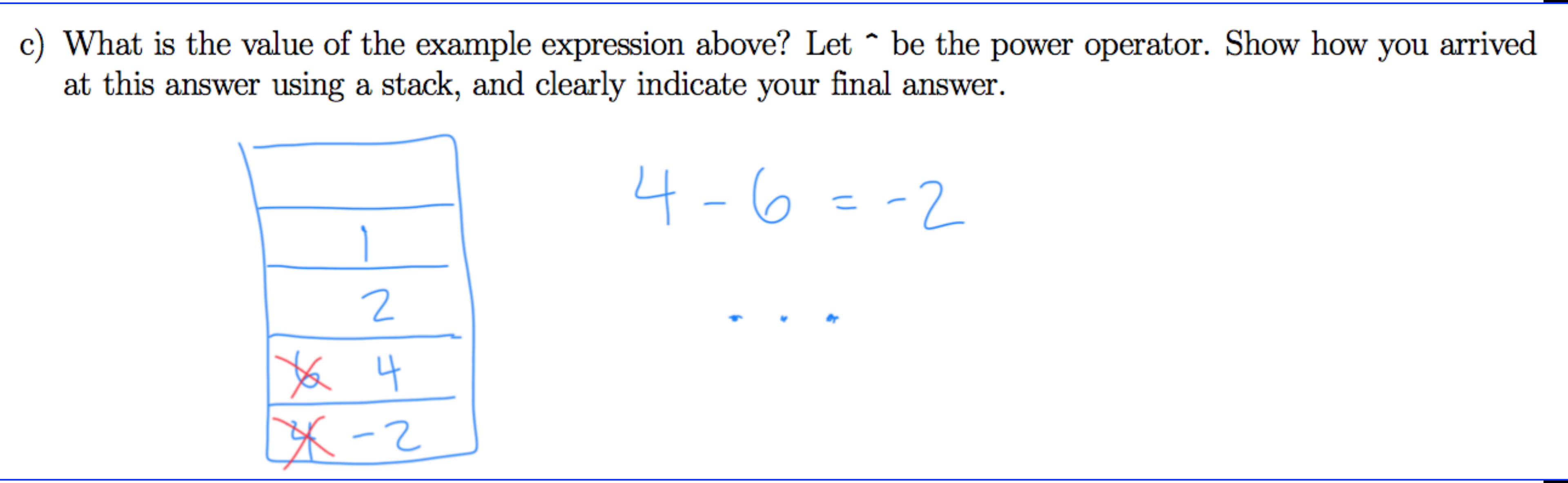


ii) What variables are in scope at the line "`// draw stack here!`"?

MIDTERM: PART 2



```
4 6 - 4 2 1 ^ / +
```



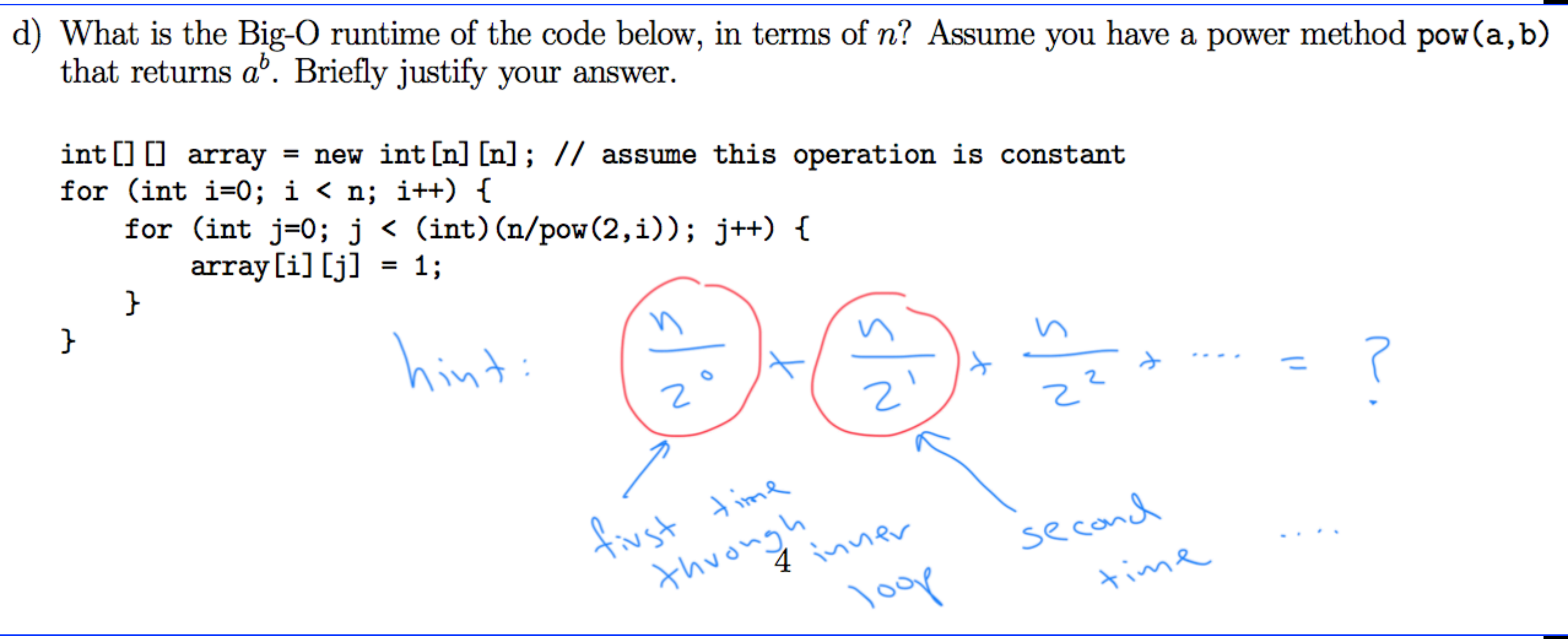
c) What is the value of the example expression above? Let ^ be the power operator. Show how you arrived at this answer using a stack, and clearly indicate your final answer.

1
2
~~6~~ 4
~~4~~ -2

4 - 6 = -2

. . .

MIDTERM: PART 2



d) What is the Big-O runtime of the code below, in terms of n? Assume you have a power method `pow(a,b)` that returns a^b. Briefly justify your answer.

```
int[][] array = new int[n][n]; // assume this operation is constant
for (int i=0; i < n; i++) {
    for (int j=0; j < (int)(n/pow(2,i)); j++) {
        array[i][j] = 1;
    }
}
```

hint: $\frac{n}{2^0} + \frac{n}{2^1} + \frac{n}{2^2} + \cdots = ?$

first time through inner loop

second time

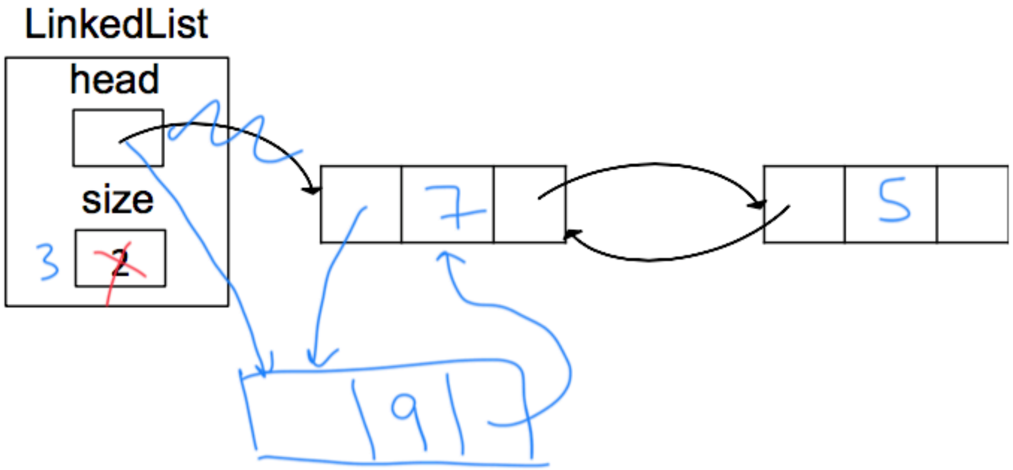
....

MIDTERM: PART 3

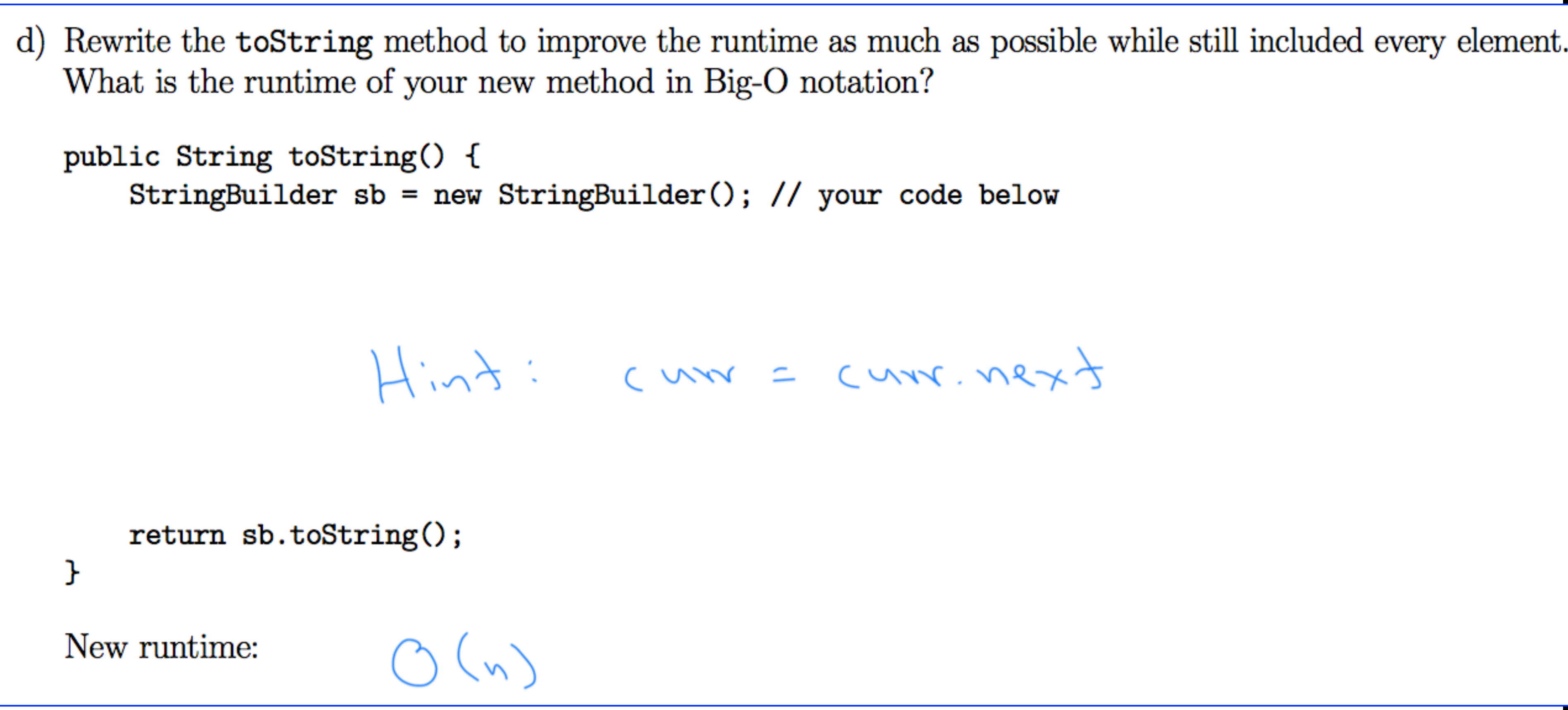
a) Say I have the following code in a **main** method:

```
LinkedList<Integer> nums = new LinkedList<Integer>();
nums.add(5);
nums.add(7);
```

This will add two numbers to a list of Integers. In the digram below, label which node contains the data 5 and which contains the data 7 (i.e. write the numbers in the center of each node).



MIDTERM: PART 3



d) Rewrite the **toString** method to improve the runtime as much as possible while still included every element. What is the runtime of your new method in Big-O notation?

```
public String toString() {
    StringBuilder sb = new StringBuilder(); // your code below
```

Hint: curr = curr.next

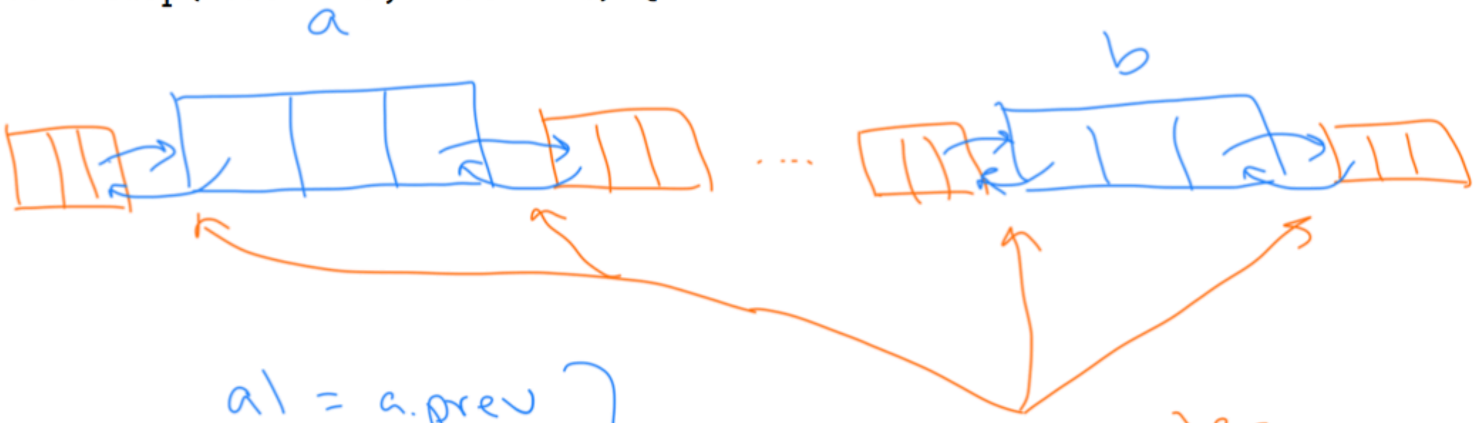
```
    return sb.toString();
}
```

New runtime: O(n)

MIDTERM: PART 3

f) Add a **swap** method to **LinkedList** that takes two nodes and swaps them in the list. For example, on a list containing `[c, a, b]` in that order, calling **swap** with the nodes containing elements **a** and **b** should change the list to now have `[c, b, a]`. Note that **swap** must swap the *nodes*, not just their contents. You can assume that the two parameters are always valid references to nodes in the list. Drawing a diagram may be helpful.

```
// This swap method is within the LinkedList class above
// precondition: a and b are valid Node references (not the data inside the Nodes)
public void swap(Node<E> a, Node<E> b) {
```



a1 = a.prev

a2 = a.next

b1 = b.prev

b2 = b.next

Save!

8 pointer updates!

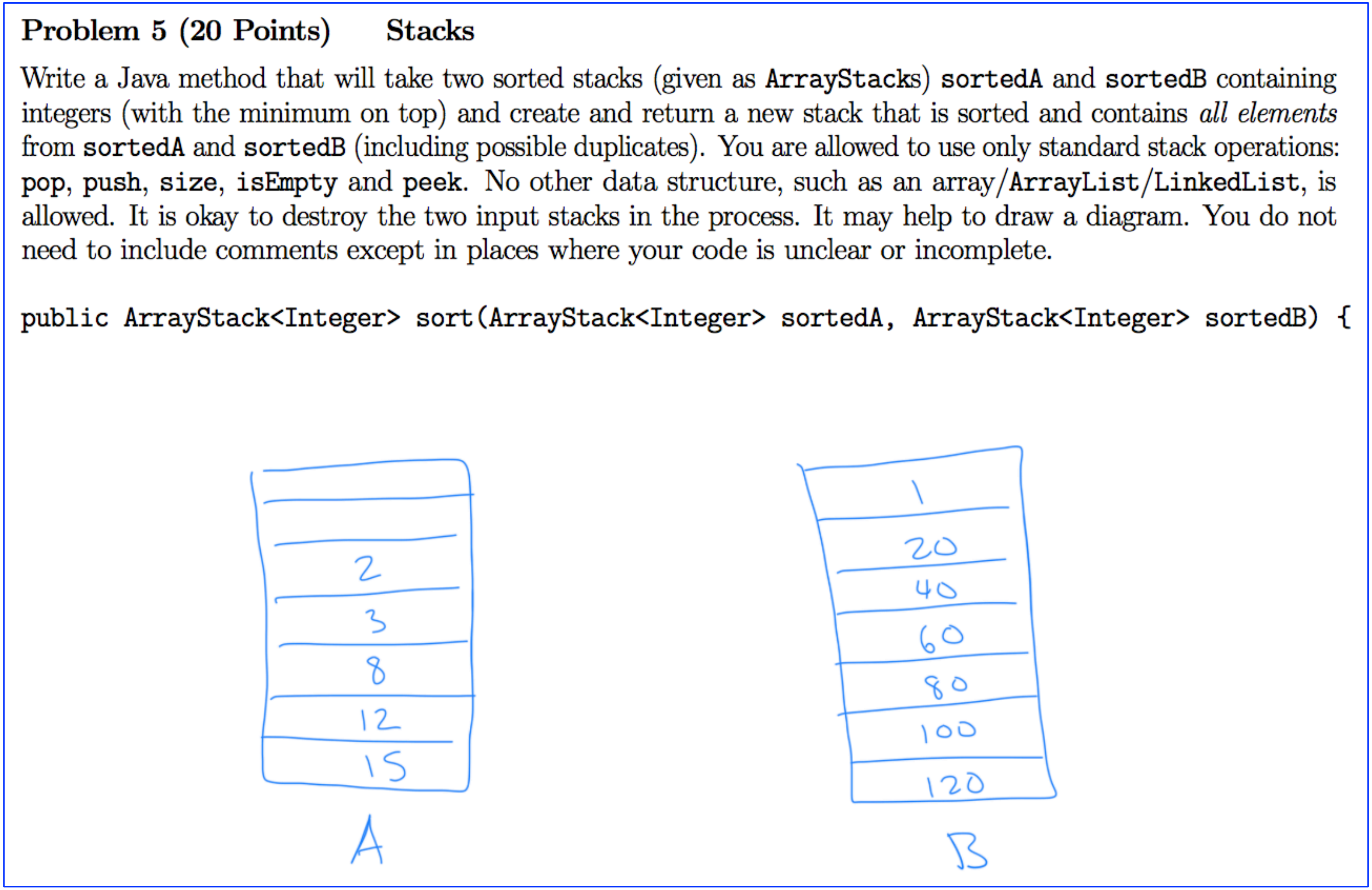
```
}
```

MIDTERM: PART 4

b) A **Major** class that extends **Student** and is designed to hold data specific to CS majors (i.e. "requirements completed" and "senior thesis"). You may leave "requirements completed" as a **String**. Assume that the input to the constructor is the same, and that the constructor is only called for students who are actually CS majors.

```
public class Major extends Student {

        ⋮

        public Major(String[] row) {
            super(row)
                ⋮
        }
}
```

MIDTERM: PART 5



Problem 5 (20 Points) **Stacks**

Write a Java method that will take two sorted stacks (given as **ArrayStacks**) **sortedA** and **sortedB** containing integers (with the minimum on top) and create and return a new stack that is sorted and contains *all elements* from **sortedA** and **sortedB** (including possible duplicates). You are allowed to use only standard stack operations: **pop**, **push**, **size**, **isEmpty** and **peek**. No other data structure, such as an array/**ArrayList**/**LinkedList**, is allowed. It is okay to destroy the two input stacks in the process. It may help to draw a diagram. You do not need to include comments except in places where your code is unclear or incomplete.

```
public ArrayStack<Integer> sort(ArrayStack<Integer> sortedA, ArrayStack<Integer> sortedB) {
```

MAR 19 OUTLINE

- Common issues on Midterm 1
- **Continue graphs**
- Graph implementations

THE GRAPH ADT

The designation of the graph as undirected or directed happens at construction time.

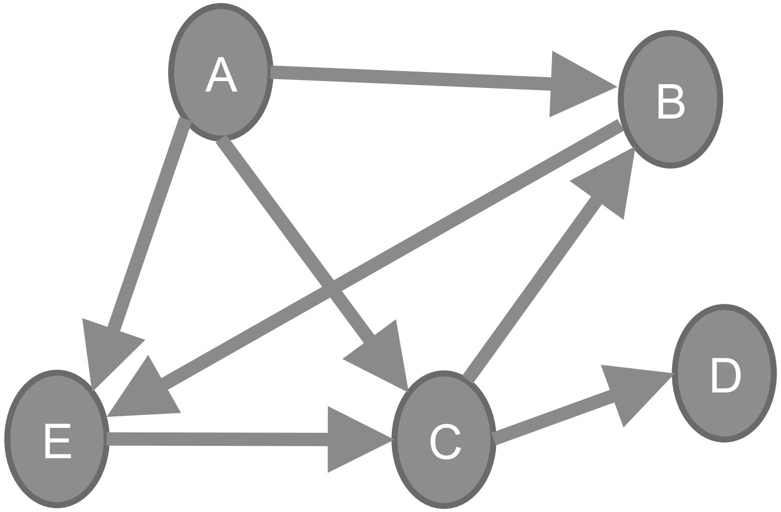
```
numVertices()
vertices()
numEdges()
edges()
getEdge(u,v)
endpoints(e)
removeVertex(v)
outDegree(v)
inDegree(v)
outgoingEdges(v)
incomingEdges(v)
insertVertex(elem)
insertEdge(u,v,elem)
removeEdge(e)
```

Note: there are many ways to implement a Graph!

PAIR EXERCISE

What should each of these methods return for this specific graph?

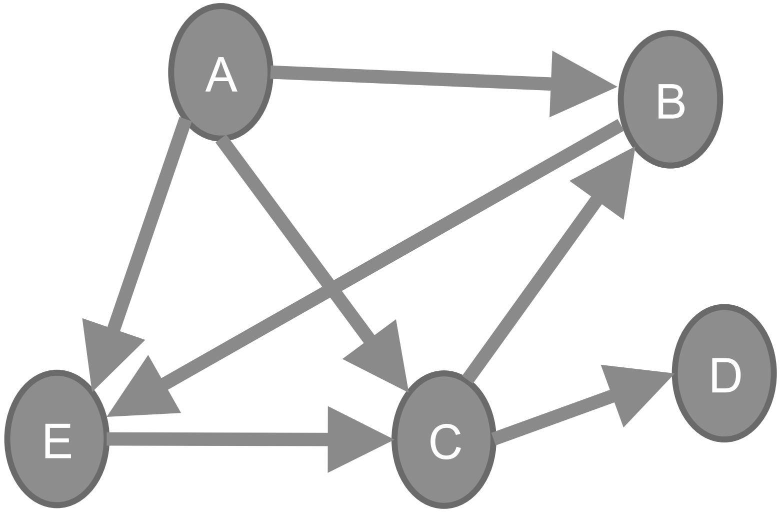
Method	Returns
`vertices()`	
`numVertices()`	
`numEdges()`	
`outDegree(C)`	
`inDegree(B)`	
`outgoingEdges(A)`	
`incomingEdges(B)`	



PAIR EXERCISE

What should each of these methods return for this specific graph?

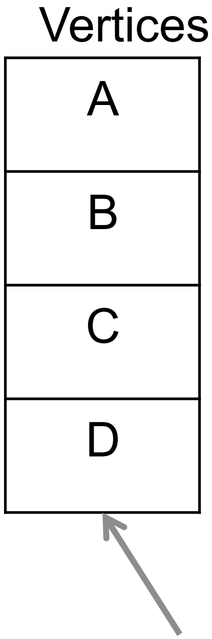
Method	Returns
`vertices()`	{A, B, C, D, E}
`numVertices()`	5
`numEdges()`	7
`outDegree(C)`	2
`inDegree(B)`	2
`outgoingEdges(A)`	{E, C, B}
`incomingEdges(B)`	{A, C}



MAR 19 OUTLINE

- Common issues on Midterm 1
- Continue graphs
- Graph implementations

GRAPH ADJACENCY LIST REPRESENTATION



Vertices

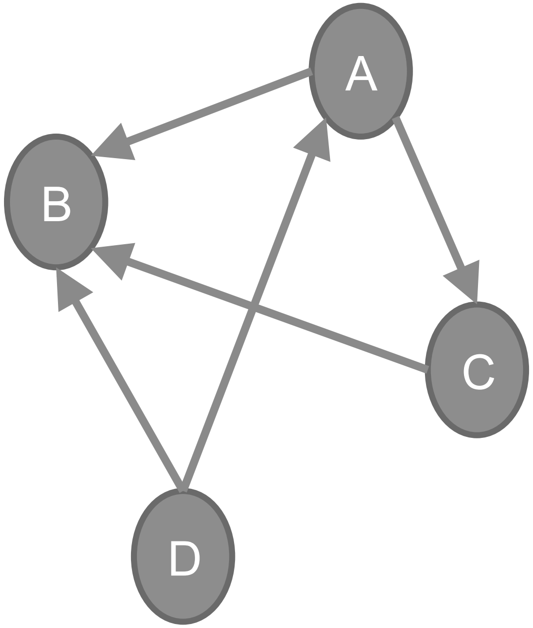
A

B

C

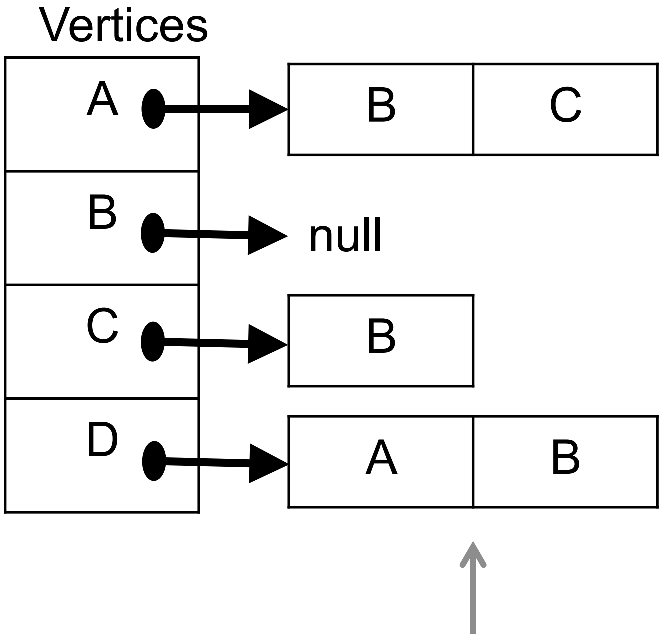
D

One instance variable:
list of Vertices



See section 14.2 of the book for more info!

GRAPH ADJACENCY LIST REPRESENTATION



Vertices

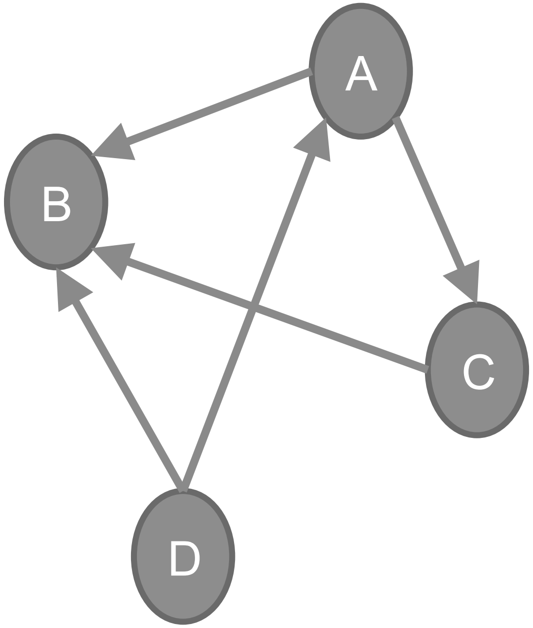
A → B C

B → null

C → B

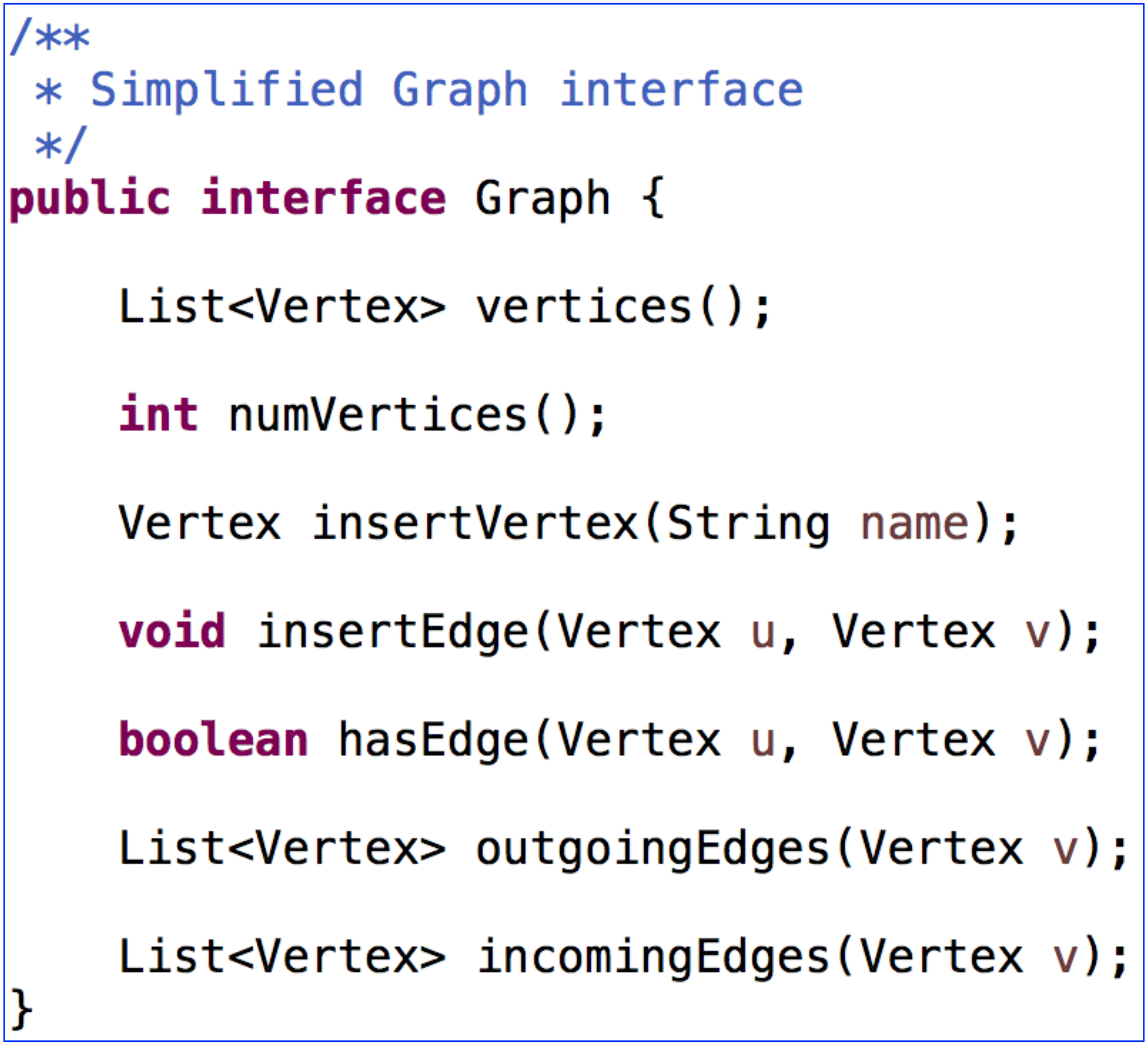
D → A B

Each Vertex contains a list of destination edges



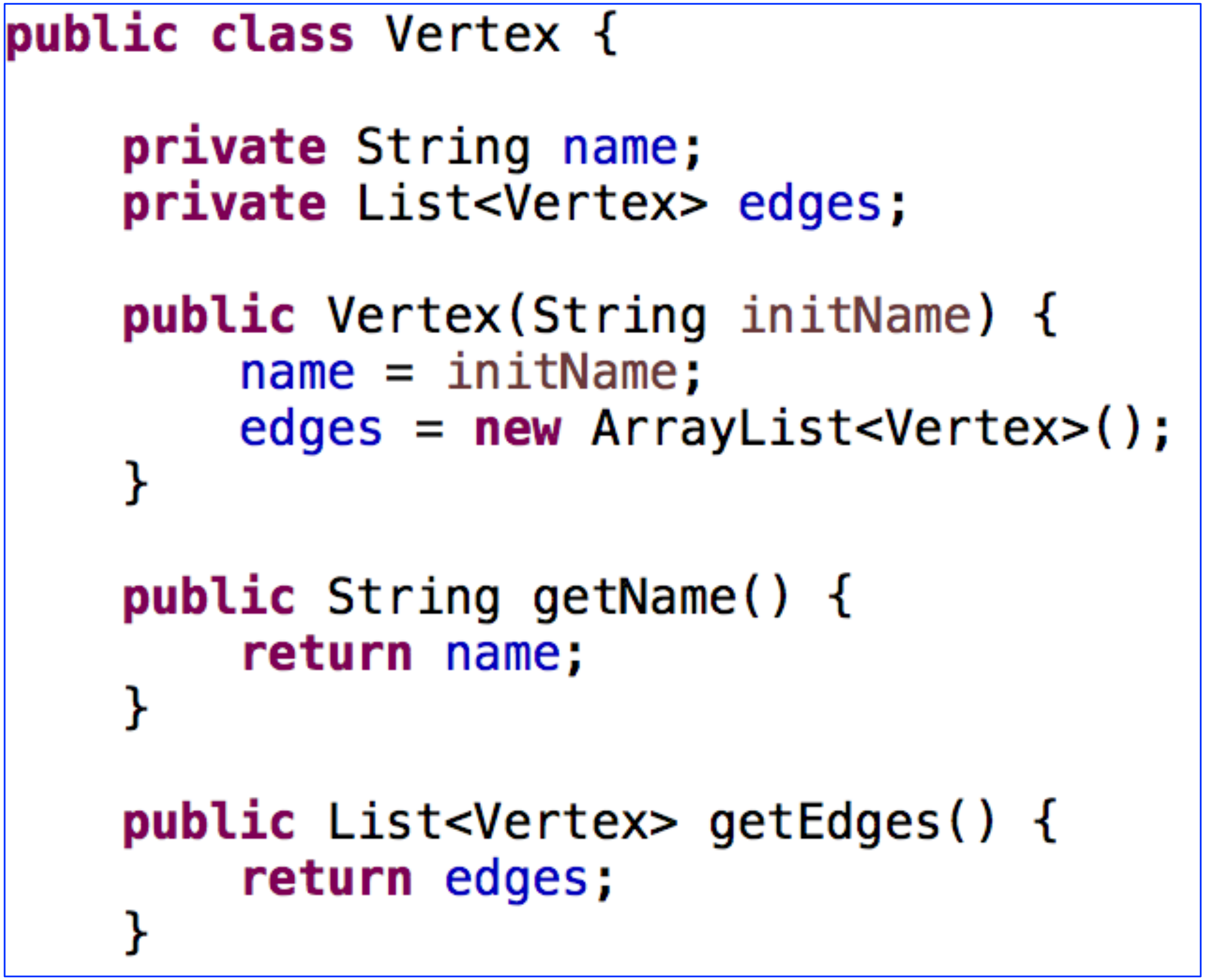
See section 14.2 of the book for more info!

SIMPLIFIED GRAPH INTERFACE



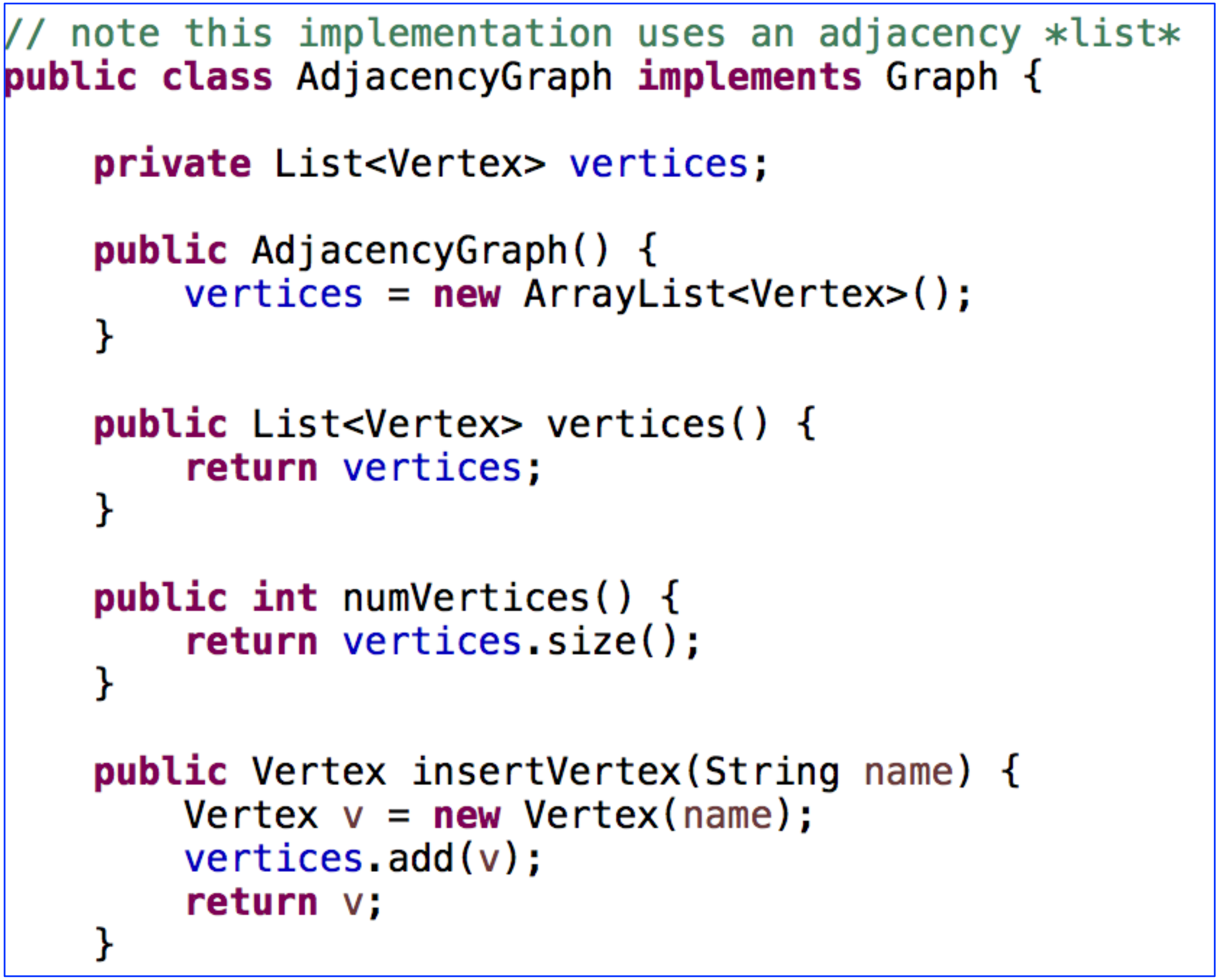
```java
/**
 * Simplified Graph interface
 */
public interface Graph {

    List<Vertex> vertices();

    int numVertices();

    Vertex insertVertex(String name);

    void insertEdge(Vertex u, Vertex v);

    boolean hasEdge(Vertex u, Vertex v);

    List<Vertex> outgoingEdges(Vertex v);

    List<Vertex> incomingEdges(Vertex v);
}
```

START OF VERTEX CLASS



```java
public class Vertex {

    private String name;
    private List<Vertex> edges;

    public Vertex(String initName) {
        name = initName;
        edges = new ArrayList<Vertex>();
    }

    public String getName() {
        return name;
    }

    public List<Vertex> getEdges() {
        return edges;
    }
```

START OF ADJACENCY GRAPH CLASS



```
// note this implementation uses an adjacency *list*
public class AdjacencyGraph implements Graph {

    private List<Vertex> vertices;

    public AdjacencyGraph() {
        vertices = new ArrayList<Vertex>();
    }

    public List<Vertex> vertices() {
        return vertices;
    }

    public int numVertices() {
        return vertices.size();
    }

    public Vertex insertVertex(String name) {
        Vertex v = new Vertex(name);
        vertices.add(v);
        return v;
    }
```

GRAPH ADJACENCY LIST RUNTIMES (PAIR EXERCISE)

Let n be the number of vertices. Fill in the runtime for each method below.

```
List<Vertex> vertices()

int numVertices()

Vertex insertVertex(elem)

void insertEdge(u,v)

boolean hasEdge(u,v)

List<Vertex> outgoingEdges(v)

List<Vertex> incomingEdges(v)
```

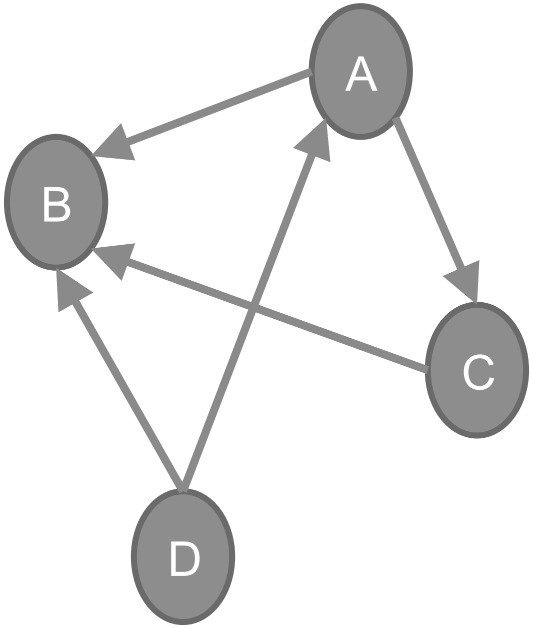
GRAPH ADJACENCY LIST RUNTIMES (PAIR EXERCISE)

Let n be the number of vertices. Fill in the runtime for each method below.

`List<Vertex> vertices()`	O(1)
`int numVertices()`	O(1)
`Vertex insertVertex(elem)`	O(1)
`void insertEdge(u,v)`	O(1)
`boolean hasEdge(u,v)`	O(n)
`List<Vertex> outgoingEdges(v)`	O(1)
`List<Vertex> incomingEdges(v)`	$O(n^2)$

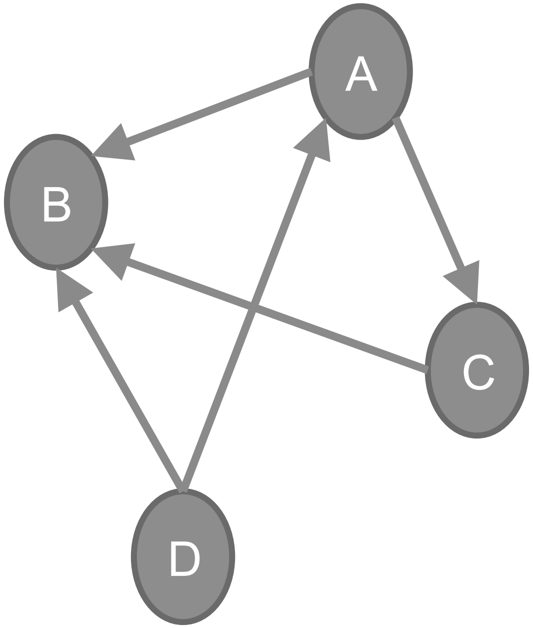
ADJACENCY MATRIX

	A	B	C	D
A				
B				
C				
D				



ADJACENCY MATRIX

	A	B	C	D
A	0	1	1	0
B	0	0	0	0
C	0	1	0	0
D	1	1	0	0



GRAPH ADJACENCY MATRIX RUNTIMES

Let n be the number of vertices. Fill in the runtime for each method below.

```
List<Vertex> vertices()

int numVertices()

Vertex insertVertex(elem)

void insertEdge(u,v)

boolean hasEdge(u,v)

List<Vertex> outgoingEdges(v)

List<Vertex> incomingEdges(v)
```

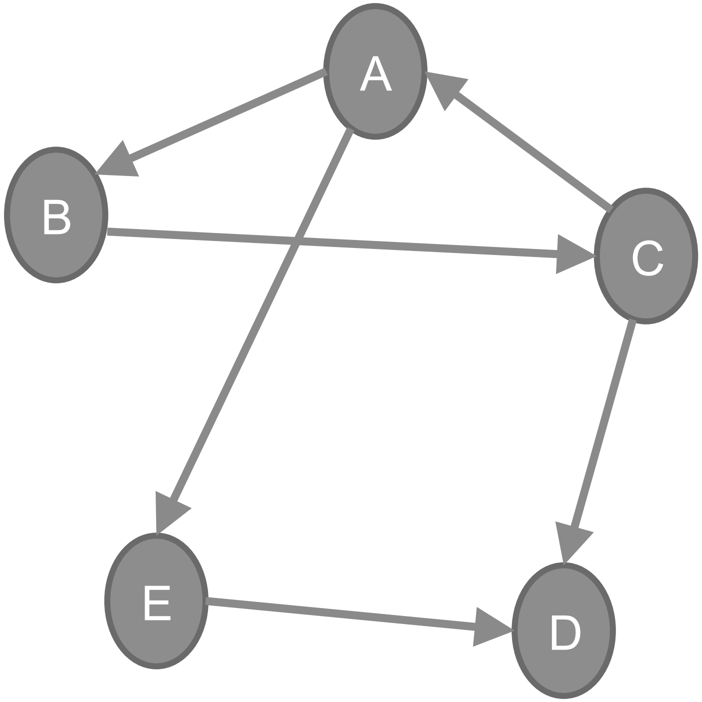
GRAPH ADJACENCY MATRIX RUNTIMES

Let n be the number of vertices. Fill in the runtime for each method below.

List<Vertex> vertices()	O(1)
int numVertices()	O(1)
Vertex insertVertex(elem)	O(1)
void insertEdge(u,v)	O(1)
boolean hasEdge(u,v)	O(1)
List<Vertex> outgoingEdges(v)	O(n)
List<Vertex> incomingEdges(v)	O(n)

EXERCISES (AFTER CLASS)

What's the adjacency matrix for this graph?



What operations might be slow with an adjacency matrix?