http://algs4.cs.princeton.edu

1.3 BAGS, QUEUES, AND STACKS

- *stacks*
- *resizing arrays*
- *queues*
- *generics*
- *iterators*
- *applications*

Do some review before labs start
Extra credit – 2 points to lab grade

- Using the stack class from Java, write a program that will push 20 random integers (between 1 and 100) onto the stack
- Print the stack
- Pop the top integer and print that integer
- Print the stack again
- Print the result of checking if 50 is in the stack
- Print the result of checking if the stack is empty
- After popping all the numbers off of the stack, print the average of the numbers
  - Will not include the top number because it was already popped
- Run it a few times to make sure you are getting a different average each time
- DUE: 9/6 at 10am
- Competition – used 14 lines of executable code

Iterable Collections

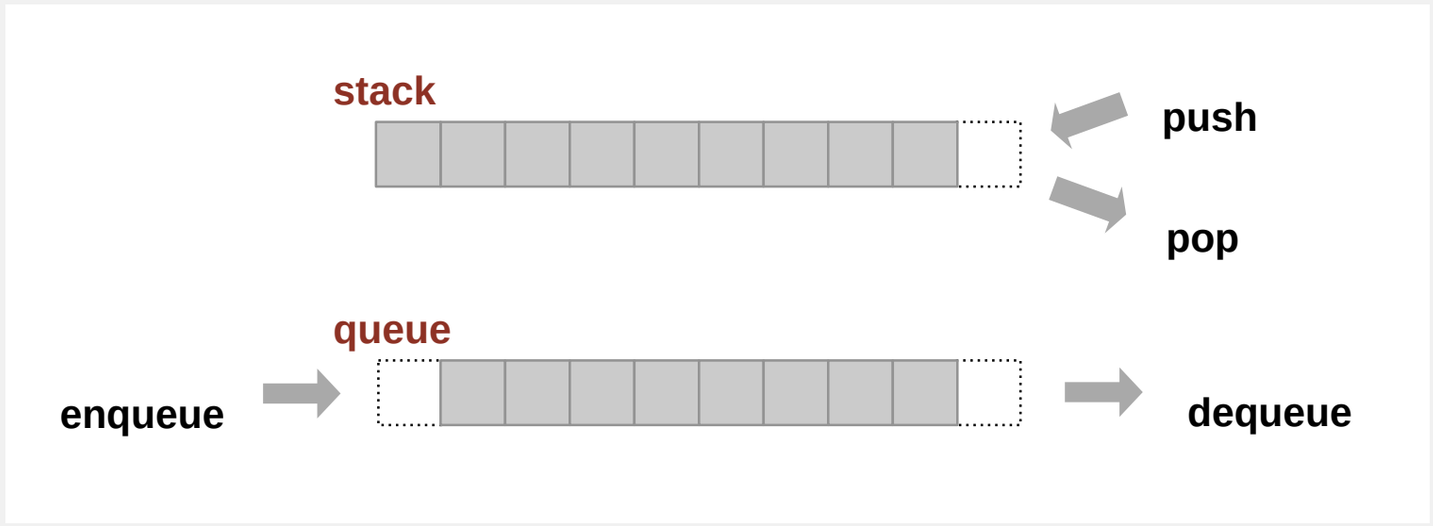
- class Bag implements iterable<Item>
  - The internal elements of the class that implements an iterable allows moving through the elements (using a for-each loop)

```
public static void main(String[] args) {
        List list = new ArrayList();
        list.add("one");
        list.add("two");
        list.add("three");

        for(Object o : list){
           System.out.println(o.toString());
        }
}
```

Stacks and queues

Fundamental data types.

- Value: collection of objects.
- Operations: insert, remove, iterate, test if empty.
- Intent is clear when we insert.
- Which item do we remove?



Stack.

Queue.

How does each data structure store data? What is that called?

Client, implementation, interface

Separate interface and implementation.

Ex: stack, queue, bag, priority queue, symbol table, union-find, ...

Benefits.

- Client can't know details of implementation $\Rightarrow$
  client has many implementation from which to choose.
- Implementation can't know details of client needs $\Rightarrow$
  many clients can re-use the same implementation.
- Design: creates modular, reusable libraries.
- Performance: use optimized implementation where it matters.

Client: program using operations defined in interface.

Implementation: actual code implementing operations.

Interface: description of data type, basic operations.

http://algs4.cs.princeton.edu

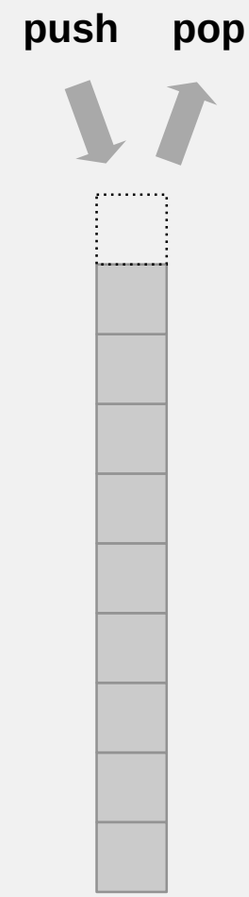
1.3 Bags, Queues, and Stacks

- *stacks*
- *resizing arrays*
- *queues*
- *generics*
- *iterators*
- *applications*

Stack API

Warmup API. Stack of strings data type.

public class StackOfStrings		
	StackOfStrings()	*create an empty stack*
void	push(String item)	*insert a new string onto stack*
String	pop()	*remove and return the string most recently added*
boolean	isEmpty()	*is the stack empty?*
int	size()	*number of strings on the stack*

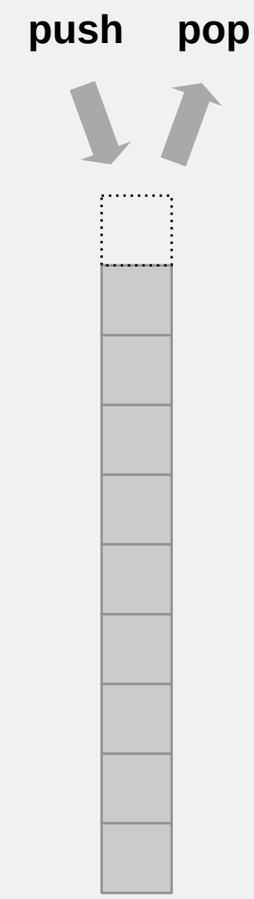


StackOfStrings.java
LOOK AT THE CODE AND TELL ME HOW IT IS IMPLEMENTED

Sample client

Warmup client. Reverse sequence of strings from standard input.

- Read string and push onto stack.
- Pop string and print.




```
% more tinyTale.txt
it was the best of times ...

% java ReverseStrings < tinyTale.txt
... times of best the was it
```

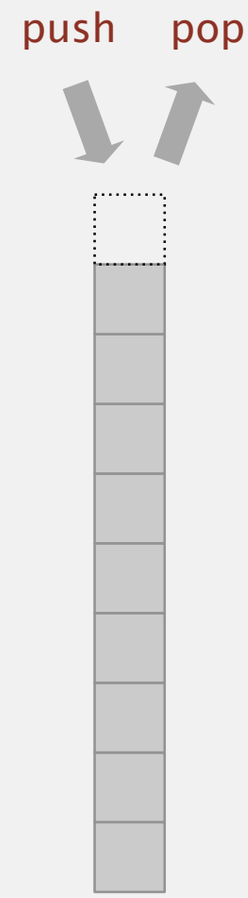
[ignoring newlines]

ReverseStrings.java
StackOfStrings.java

Stack test client

Read strings from standard input.

- If string equals "-", pop string from stack and print.
- Otherwise, push string onto stack.



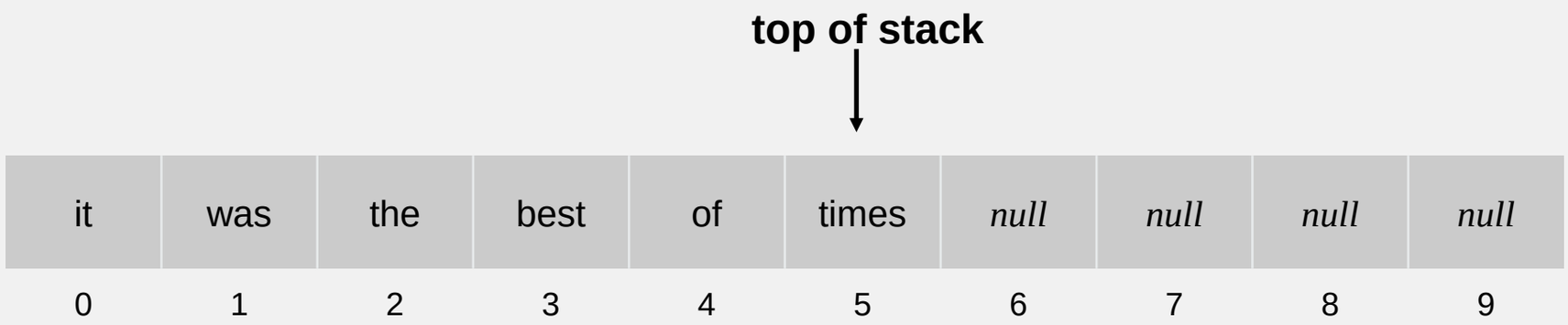

```
% more tobe.txt
to be or not to - be - -

% java StackOfStringsApps < tobe.txt
to be not
Left on stack: or be to
```

StackOfStringsApp.java
StackOfStrings.java

How to implement a fixed-capacity stack with an array?

A. Can't be done efficiently with an array.

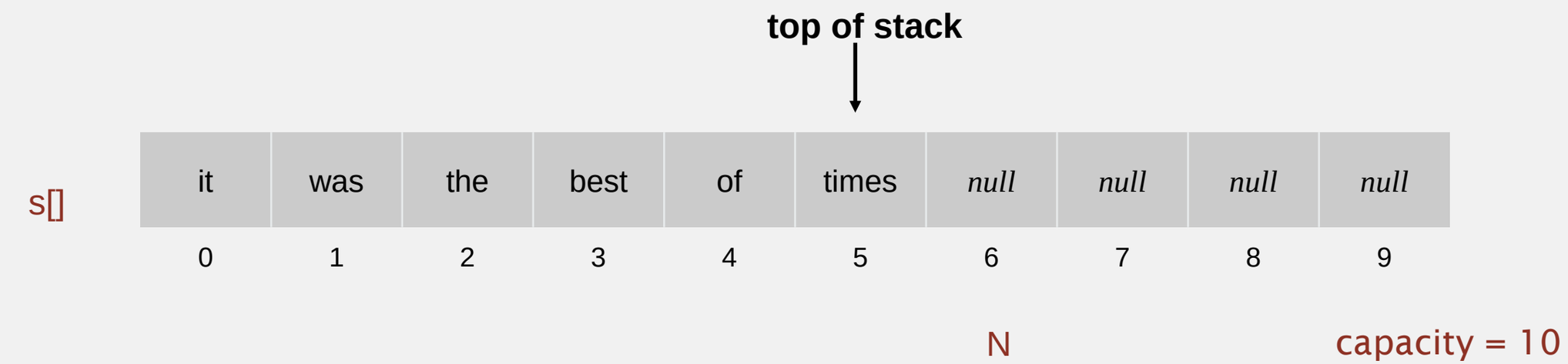
B.



C.

Fixed-capacity stack: array implementation

- Use array s[] to store N items on stack.
- push(): add new item at s[N].
- pop(): remove item from s[N-1].



Defect. Stack overflows when N exceeds capacity. [stay tuned]

Stack considerations

Overflow and underflow.

- Underflow: throw exception if pop from an empty stack.
- Overflow: use resizing array for array implementation. [stay tuned]

Null items. We allow null items to be inserted.

Loitering. Holding a reference to an object when it is no longer needed.

```
public String pop()
{  return s[--N];  }
```

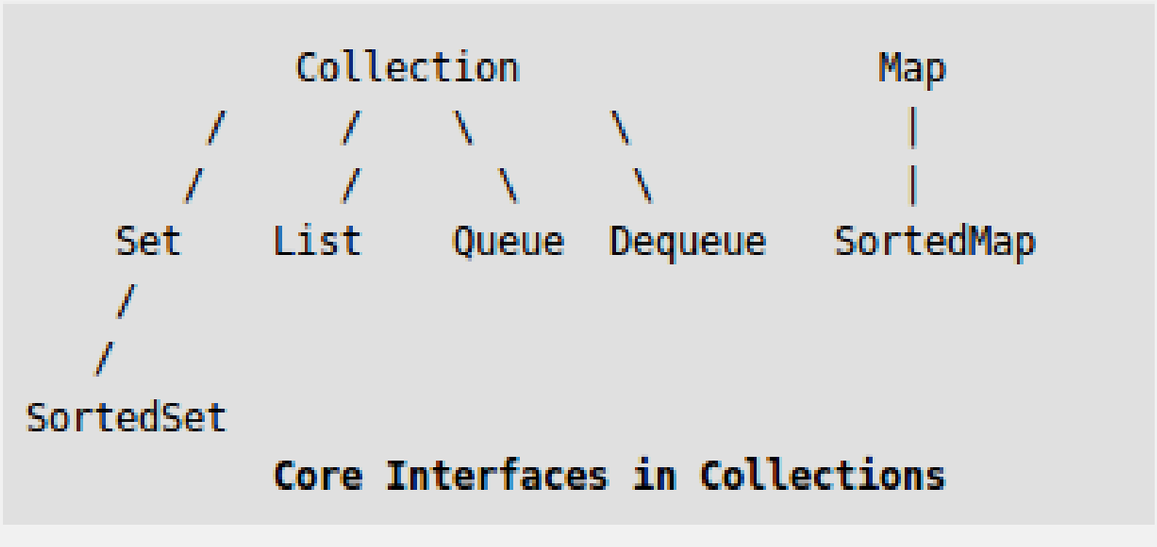
loitering

```
public String pop()
{
   String item = s[--N];
   s[N] = null;
   return item;
}
```

this version avoids "loitering":
garbage collector can reclaim memory for an object only if no outstanding references

Linked List

Linked List implements the list interface



Core Interfaces in Collections

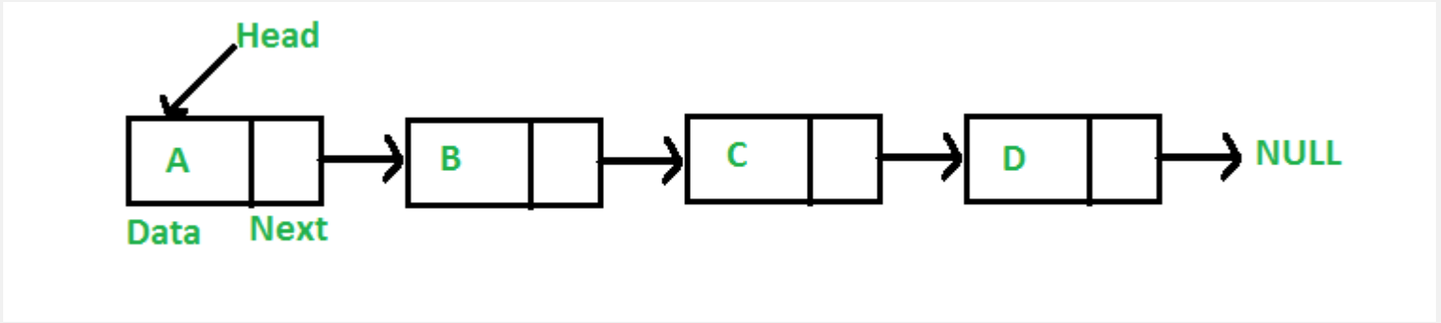
List interface calls for:

void add(int index,Object O)

boolean addAll(int index, Collection c)

Object remove(int index): Object get(int index)

Object set(int index, Object new)

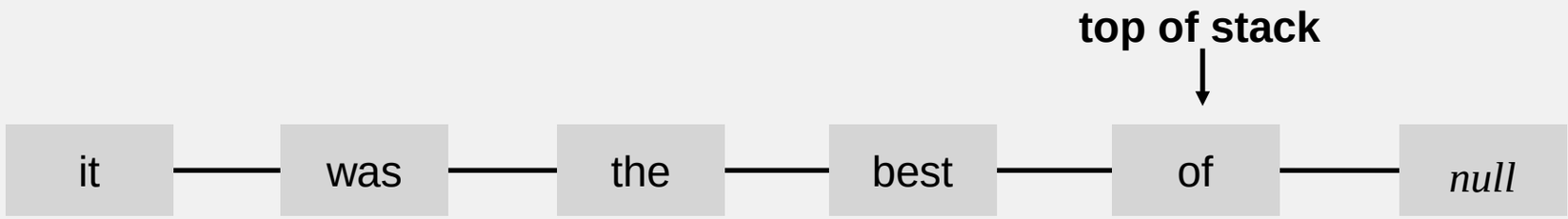


Taken from geeksforgeeks.org

How to implement a stack with a linked list?

A. Can't be done efficiently with a singly-linked list.

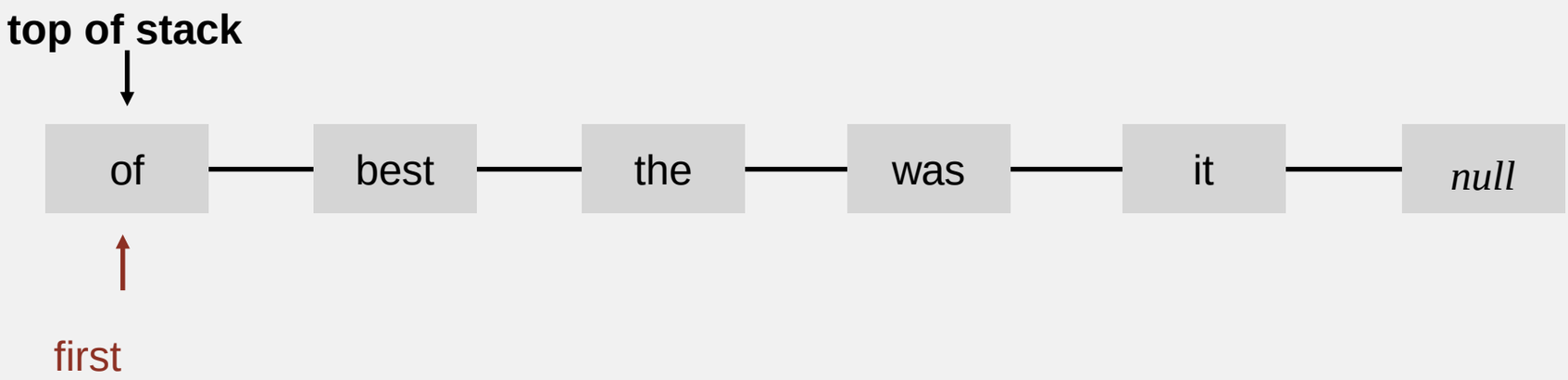
B.



C.

Stack: linked-list implementation

- Maintain pointer first to first node in a singly-linked list.
- Push new item before first.
- Pop item from first.



Stack pop: linked-list implementation

inner class

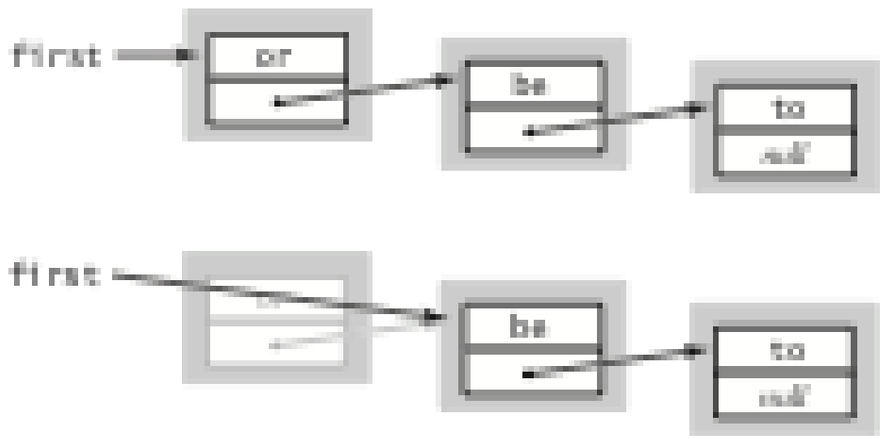
```
private class Node
{
   String item;
   Node next;
}
```

save item to return

```
String item = first.item;
```

delete first node

```
first = first.next;
```



return saved item

```
return item;
```

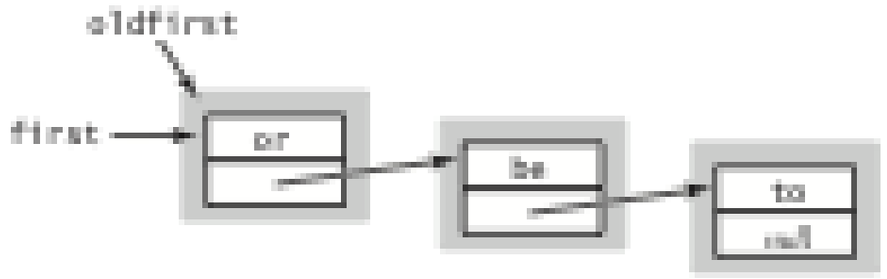
Stack push: linked-list implementation

inner class

```
private class Node
{
   String item;
   Node next;
}
```

save a link to the list

```
Node oldfirst = first;
```

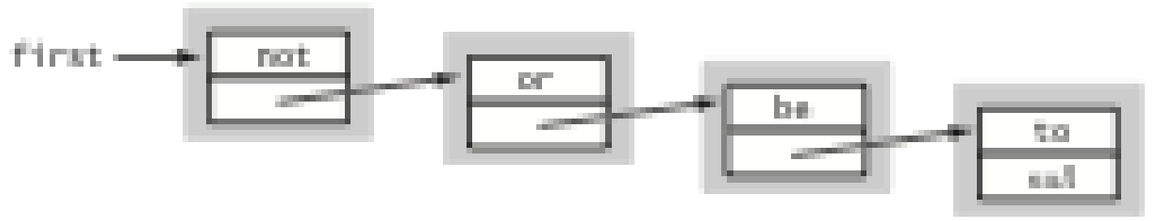


create a new node for the beginning

```
first = new Node();
```


set the instance variables in the new node

```
first.item = "not";
first.next = oldfirst;
```



Stack: linked-list implementation in Java

ListStack.java
StackOfStringsApp.java

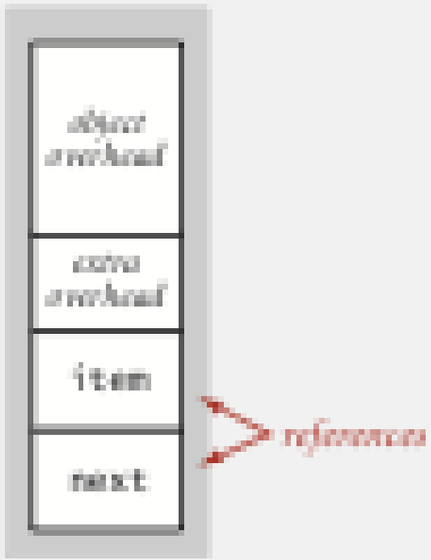
Stack: linked-list implementation performance

Proposition. Every operation takes constant time in the worst case.

Proposition. A stack with N items uses $\sim 40\,N$ bytes.

inner class

```
private class Node
{
   String item;
   Node next;
}
```



16 bytes (object overhead)

8 bytes (inner class extra overhead)

8 bytes (reference to String)

8 bytes (reference to Node)

40 bytes per stack node

Remark. This accounts for the memory for the stack
(but not the memory for strings themselves, which the client owns).

http://algs4.cs.princeton.edu

1.3 Bags, Queues, and Stacks

- *stacks*
- *resizing arrays*
- *queues*
- *generics*
- *iterators*
- *applications*

Stack: resizing-array implementation

Problem. Requiring client to provide capacity does not implement API!

Q. How to grow and shrink array?

First try.

- push(): increase size of array s[] by 1.
- pop(): decrease size of array s[] by 1.

Too expensive.

- Need to copy all items to a new array, for each operation.
- Array accesses to insert first N items = $N + (2 + 4 + \ldots + 2(N-1)) \sim N^2$.

infeasible for large N

1 array access per push

2(k–1) array accesses to expand to size k (ignoring cost to create new array)

Challenge. Ensure that array resizing happens infrequently.

Stack: resizing-array implementation

Q. How to grow array?

A. If array is full, create a new array of twice the size, and copy items.

"repeated doubling"

Array accesses to insert first N = 2^i items. $N + (2 + 4 + 8 + \ldots + N) \sim 3N.$

1 array access per push

k array accesses to double to size k (ignoring cost to create new array)

LAB QUESTION

Stack: resizing-array implementation

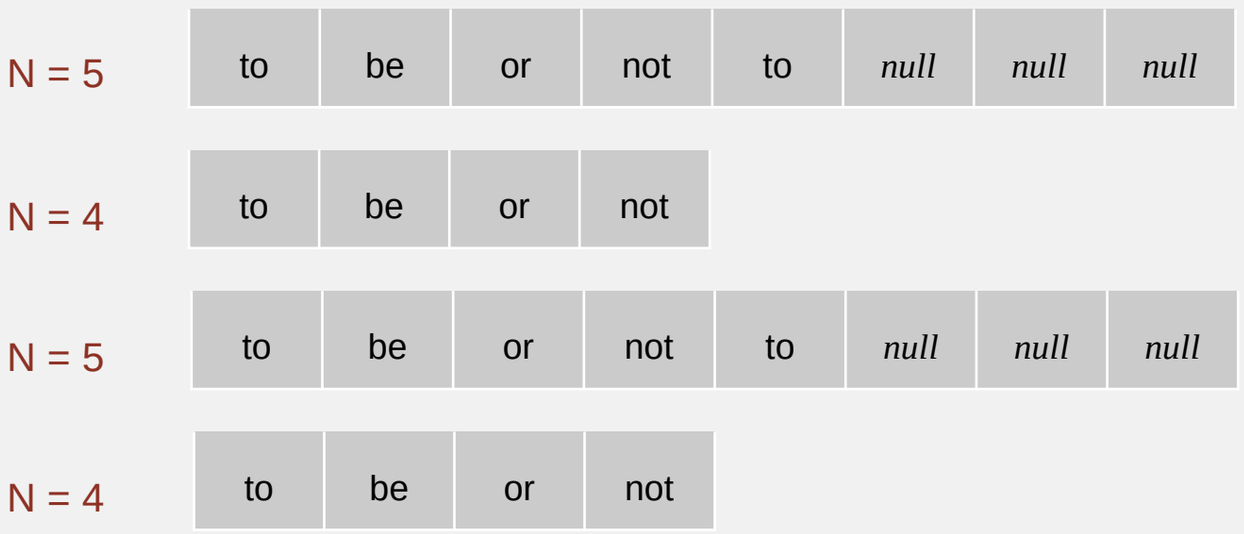
Q. How to shrink array?

First try.

- push(): double size of array s[] when array is full.
- pop(): halve size of array s[] when array is one-half full.

Too expensive in worst case.

- Consider push-pop-push-pop-... sequence when array is full.
- Each operation takes time proportional to N.



Stack: resizing-array implementation

Q. How to shrink array?

Efficient solution.

- push(): double size of array s[] when array is full.
- pop(): halve size of array s[] when array is one-quarter full.

```
public String pop() {
   String item = s[--N];
   s[N] = null;
   if (N > 0 && N == s.length/4) resize(s.length/2);
   return item;
}
```

Invariant. Array is between 25% and 100% full.

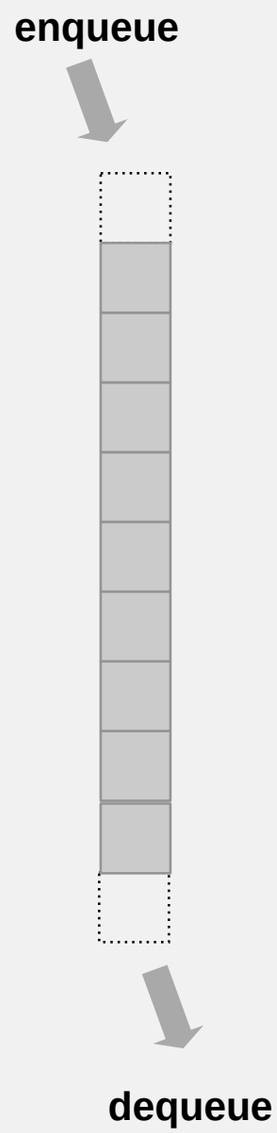
http://algs4.cs.princeton.edu

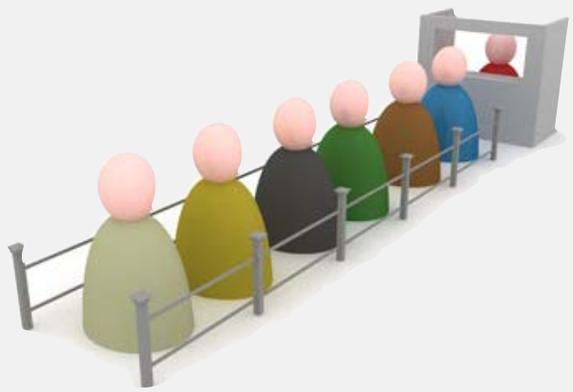
1.3 Bags, Queues, and Stacks

- *stacks*
- *resizing arrays*
- *queues*
- *generics*
- *iterators*
- *applications*

Queue API

public class Queue	
Queue()	*create an empty queue*
void enqueue(Item item)	*insert a new item onto queue*
Item dequeue()	*remove and return the string least recently added*
boolean isEmpty()	*is the queue empty?*
int size()	*number of items on the queue*

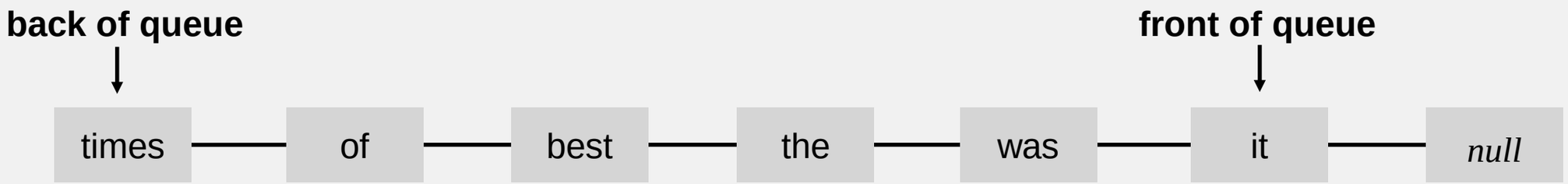




How to implement a queue with a linked list?

A. Can't be done efficiently with a singly-linked list.

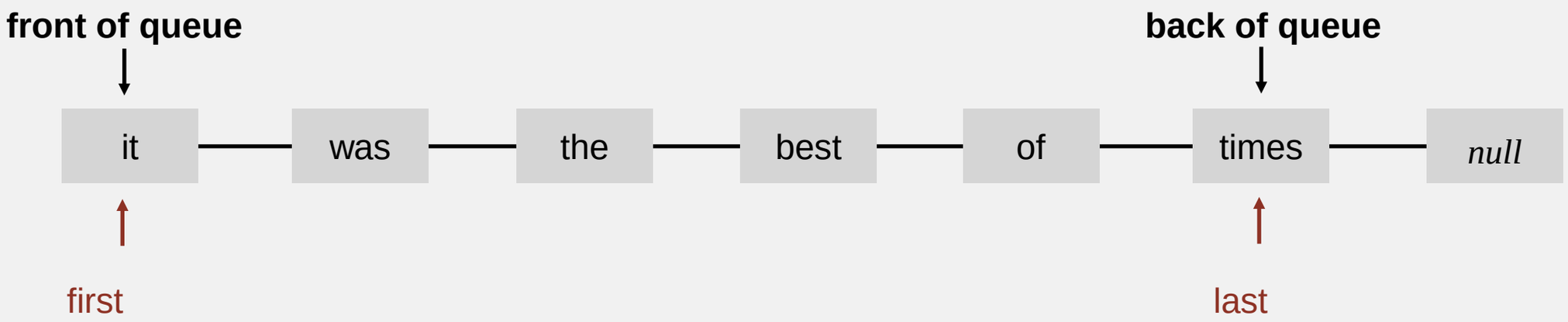
B.



C.

Queue: linked-list implementation

- Maintain one pointer first to first node in a singly-linked list.
- Maintain another pointer last to last node.
- Dequeue from first.
- Enqueue after last.



Queue dequeue: linked-list implementation

inner class

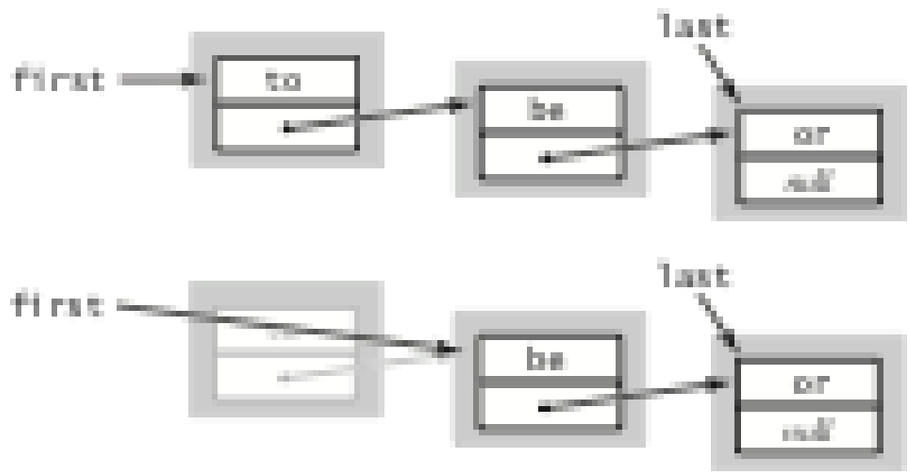
```
private class Node
{
   String item;
   Node next;
}
```

save item to return

```
String item = first.item;
```

delete first node

```
first = first.next;
```



return saved item

```
return item;
```

Remark. Identical code to linked-list stack pop().

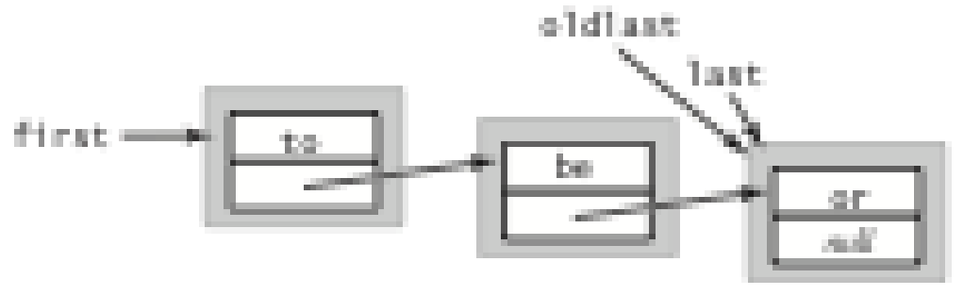
Queue enqueue: linked-list implementation

inner class

```
private class Node
{
   String item;
   Node next;
}
```

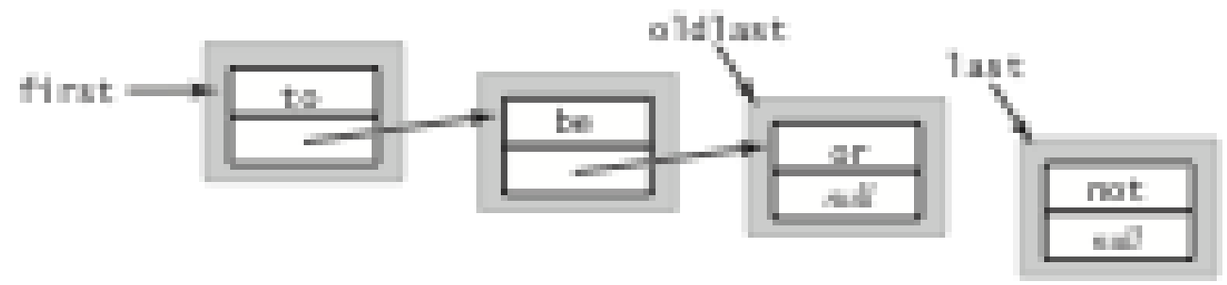
save a link to the last node

```
Node oldlast = last;
```



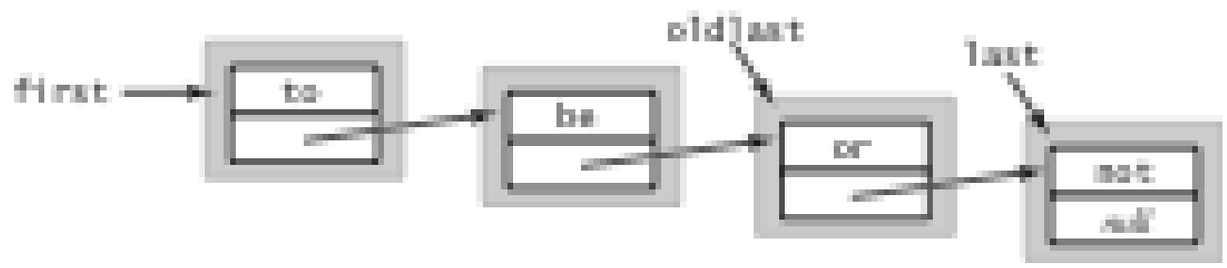
create a new node for the end

```
last = new Node();
last.item = "not";
```



link the new node to the end of the list

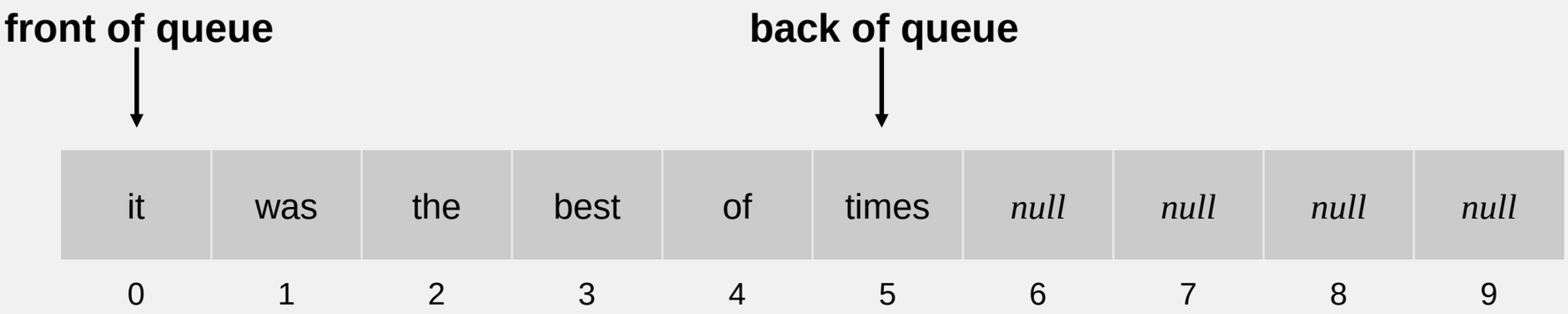
```
oldlast.next = last;
```



How to implement a fixed-capacity queue with an array?

A. Can't be done efficiently with an array.

B.



C.