

INTRODUCTION TO ALGORITHMS

LECTURE 3: UNKNOWN ARRAY SIZE PROBLEM

Yao-Chung Fan
yfan@nchu.edu.tw

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.

Problem: String Token Reserves

Given a string e.g., “Today is a good day”

Return: day good a is Today (a reverse of the given string token)

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

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

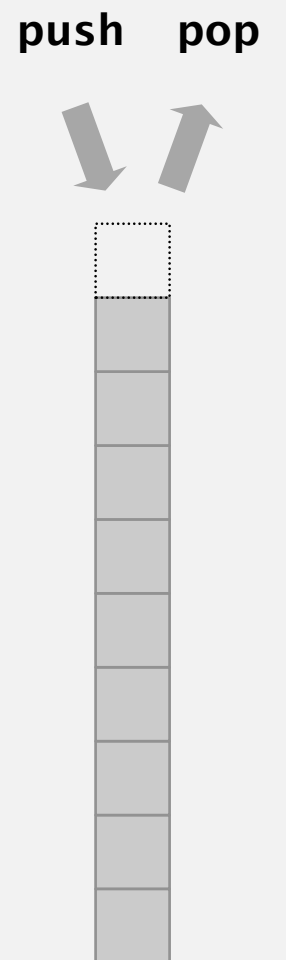
How to Implement such a requirement ?

Sample client

Warmup client. Reverse sequence of strings from standard input.

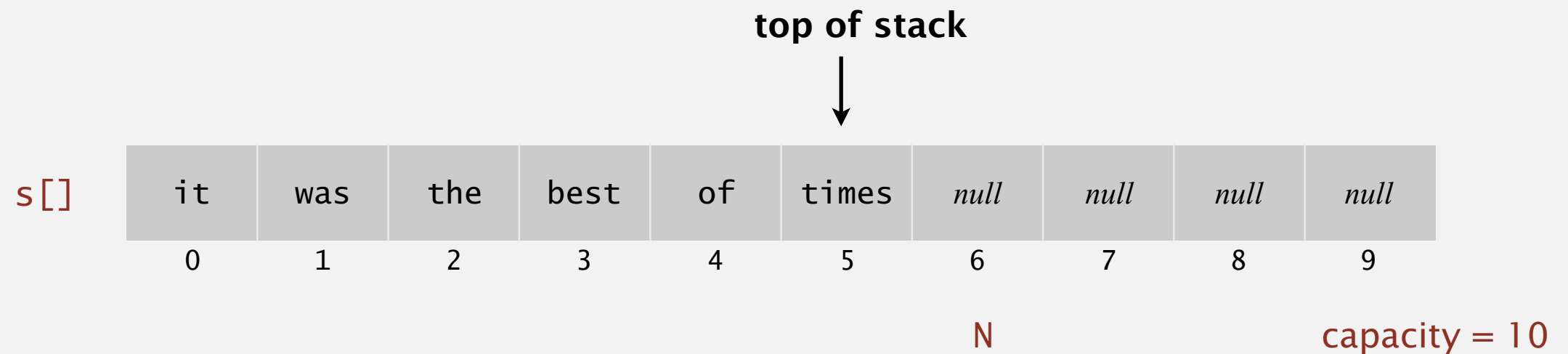
- Read string and push onto stack.
- Pop string and print.

```
public static void main(String[] args)
{
    StackOfStrings stack = new StackOfStrings();
    while (!StdIn.isEmpty())
        stack.push(StdIn.readString());
    while (!stack.isEmpty())
        StdOut.println(stack.pop());
}
```



Array implementation for String Token Reserves

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



Array implementation for String Token Reserves

```
public class FixedCapacityStackOfStrings
{
    private String[] s;
    private int N = 0;

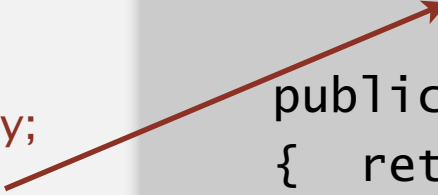
    public FixedCapacityStackOfStrings(int capacity)
    { s = new String[capacity]; }

    public boolean isEmpty()
    { return N == 0; }

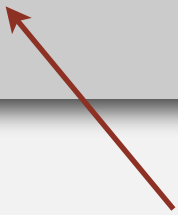
    public void push(String item)
    { s[N++] = item; }

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

use to index into array;
then increment N



decrement N;
then use to index into array



Defect. Stack overflows when N exceeds capacity.

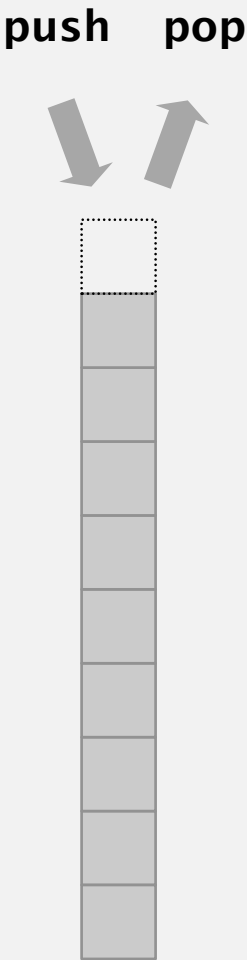
UNKNOWN ARRAY SIZE PROBLEM

- ▶ *Linked List Implementation*
- ▶ *Resizing Arrays Implementation*

Stack API

Warmup API. Stack of strings data type.

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



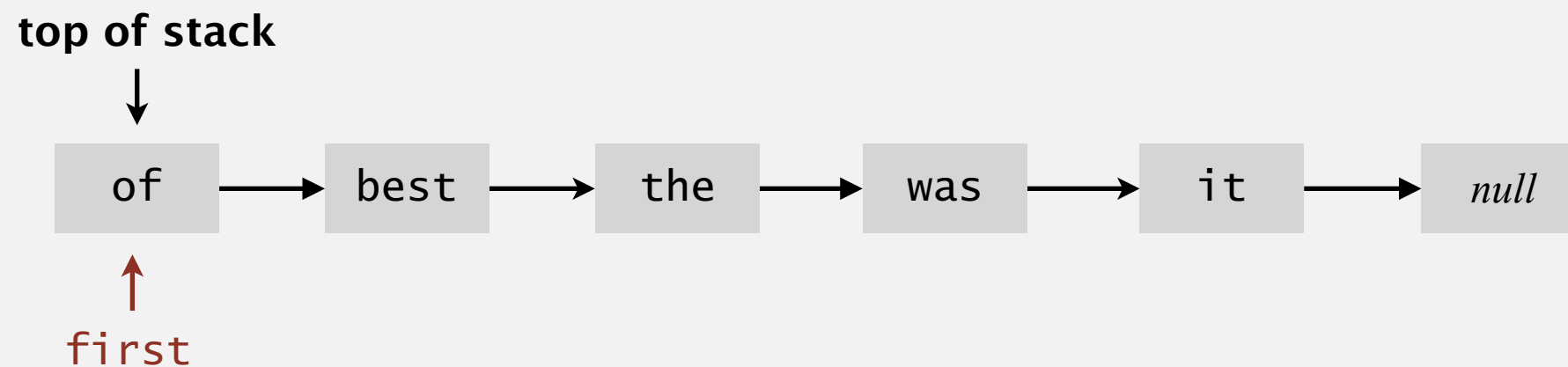
Warmup client. Reverse sequence of strings from standard input.

Stack: linked-list implementation

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

inner class

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



Stack: linked-list implementation in Java

```
public class LinkedStackOfStrings
{
    private Node first = null;

    private class Node
    {
        String item;
        Node next;
    }

    public boolean isEmpty()
    { return first == null; }

    public void push(String item)
    {
        Node oldfirst = first;
        first = new Node();
        first.item = item;
        first.next = oldfirst;
    }

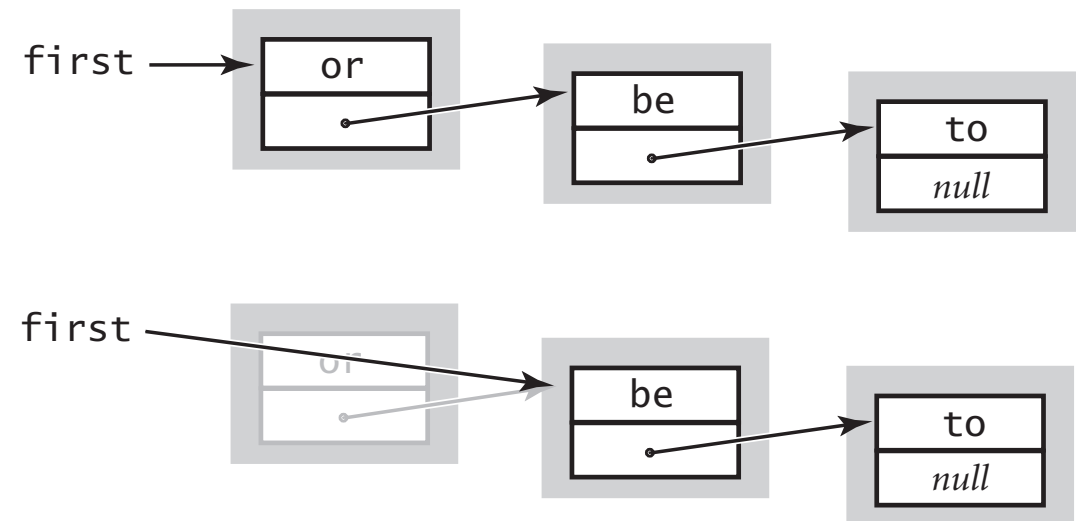
    public String pop()
    {
        String item = first.item;
        first = first.next;
        return item;
    }
}
```

save item to return

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

delete first node

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

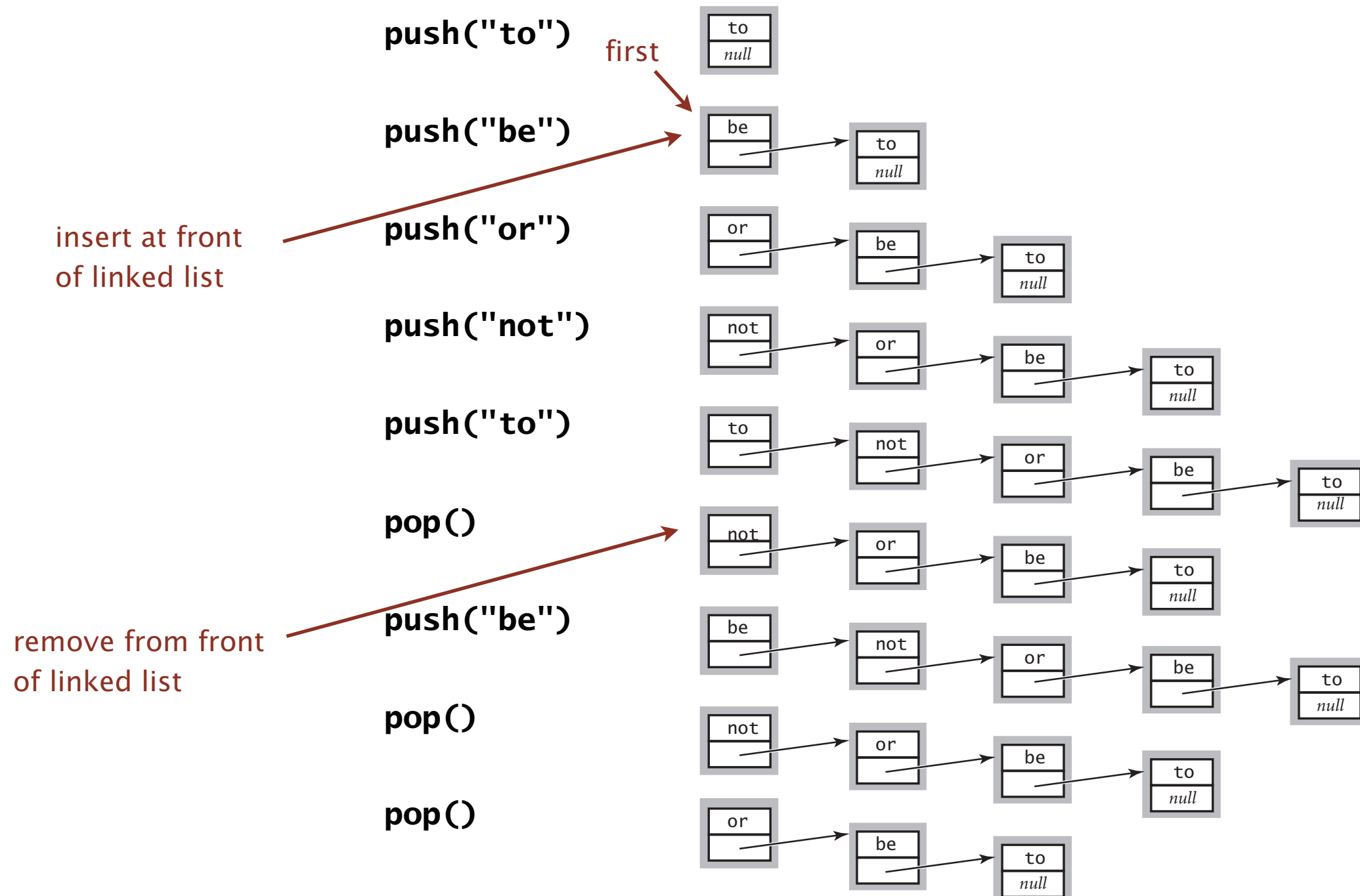


return saved item

```
return item;
```

Stack: linked-list representation

Maintain pointer to first node in a linked list; insert/remove from **front**.

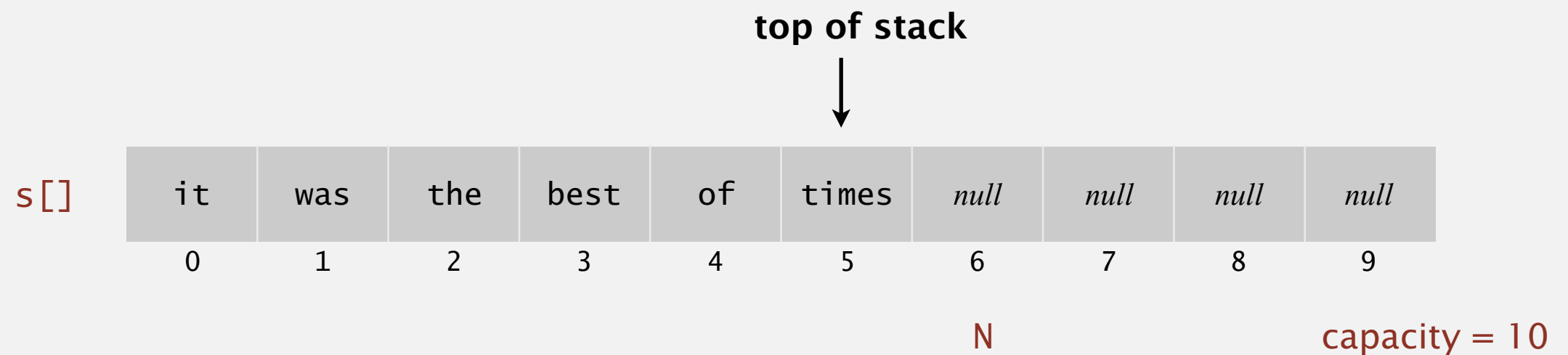


UNKNOWN ARRAY SIZE PROBLEM

- ▶ *Linked List Implementation*
- ▶ *resizing arrays Implementation*

Fixed-capacity stack: array implementation

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



Fixed-capacity stack: array implementation

```
public class FixedCapacityStackOfStrings
{
    private String[] s;
    private int N = 0;

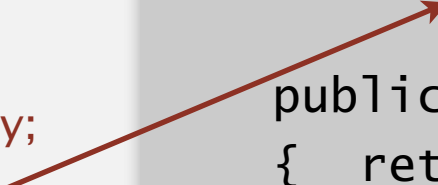
    public FixedCapacityStackOfStrings(int capacity)
    { s = new String[capacity]; }

    public boolean isEmpty()
    { return N == 0; }

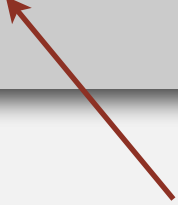
    public void push(String item)
    { s[N++] = item; }

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

use to index into array;
then increment N



decrement N;
then use to index into array



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

Q. How to grow and shrink array?

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.

```
private void resize(int capacity)
{
    String[] copy = new String[capacity];
    for (int i = 0; i < N; i++)
        copy[i] = s[i];
    s = copy;
}
```

Too expensive.

- Need to copy all items to a new array, for each operation.
- Array accesses to insert first N items = $N + (2 + 4 + 6 + 8 \dots + 2(N - 1)) \sim N^2$.

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)

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"

```
public ResizingArrayStackOfStrings()
{
    s = new String[1];
}

public void push(String item)
{
    if (N == s.length) resize(2 * s.length);
    s[N++] = item;
}

private void resize(int capacity)
{
    String[] copy = new String[capacity];
    for (int i = 0; i < N; i++)
        copy[i] = s[i];
    s = copy;
}
```

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

↑
1 array access
per push

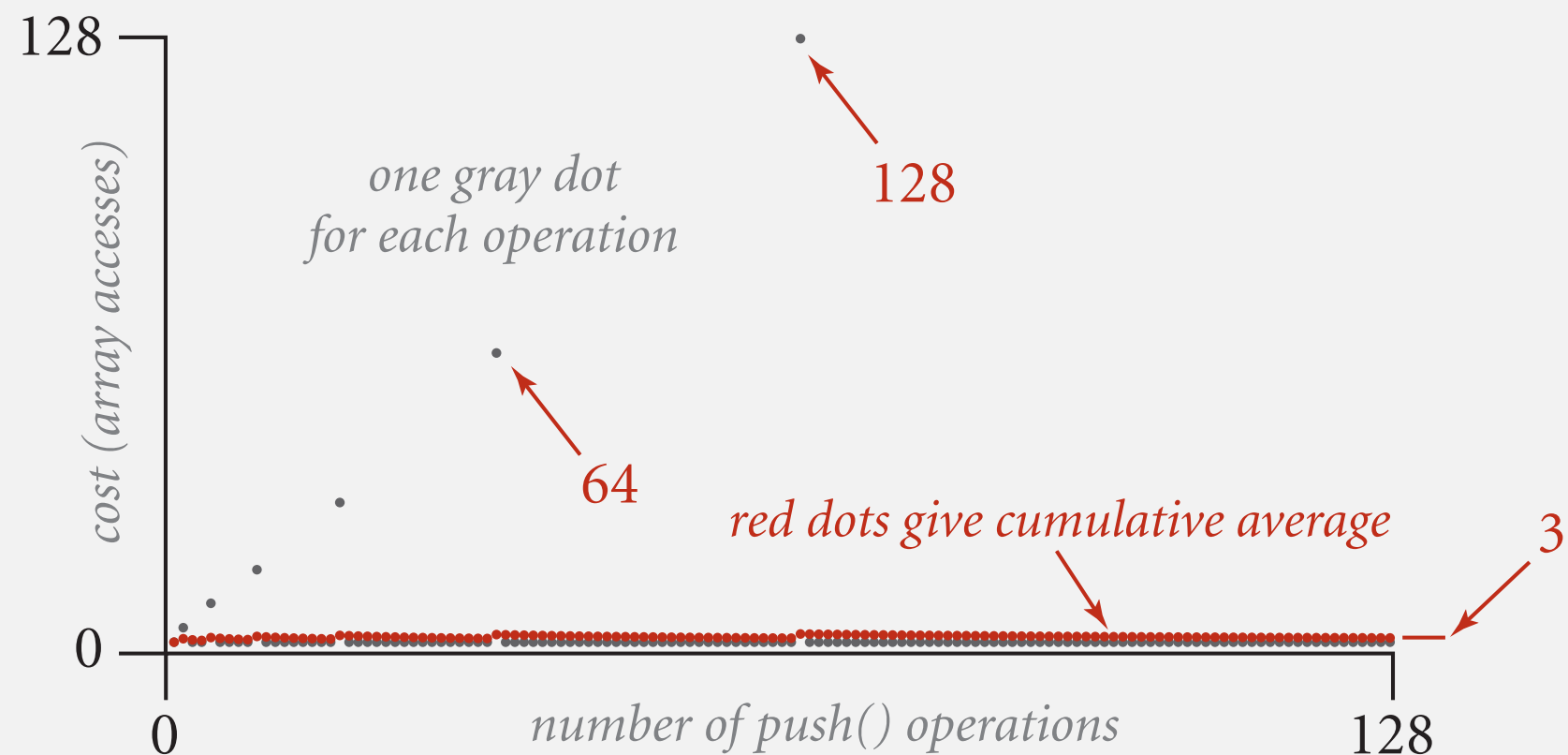
↑
k array accesses to double to size k
(ignoring cost to create new array)

Stack: amortized cost of adding to a stack

Cost of inserting first N items. $N + (2 + 4 + 8 + \dots + N) \sim 3N$.

↑
1 array access
per push

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



Stack: resizing-array implementation

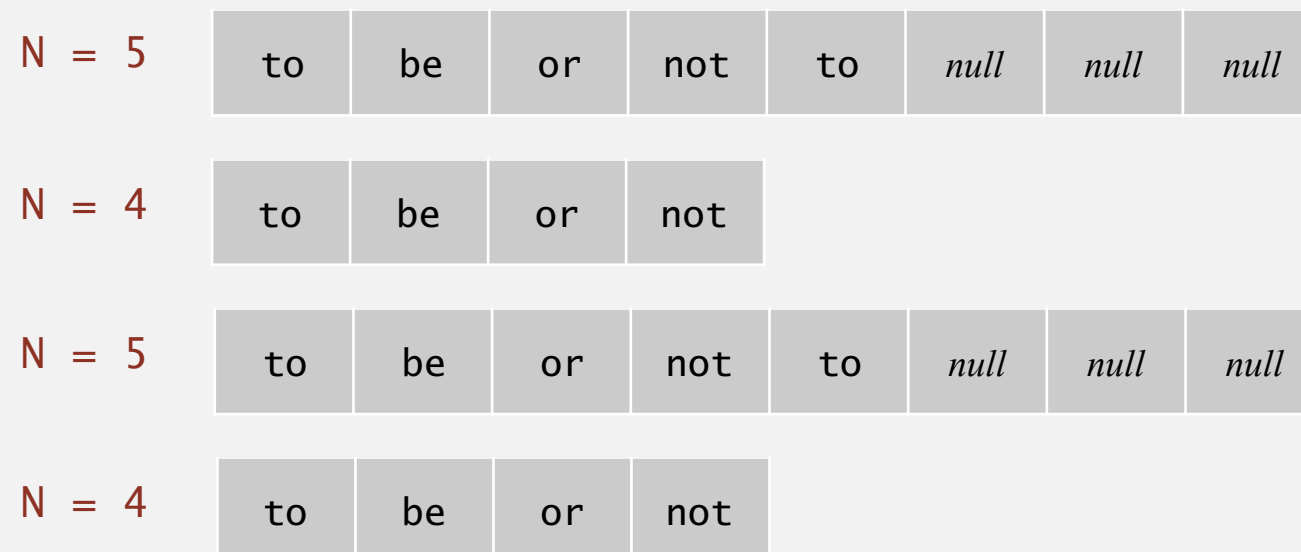
Q. How to shrink array?

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.

Stack: resizing-array implementation trace

push()	pop()	N	a.length	a[]								
				0	1	2	3	4	5	6	7	
		0	1	<i>null</i>								
to		1	1	to								
be		2	2	to	be							
or		3	4	to	be	or	<i>null</i>					
not		4	4	to	be	or	not					
to		5	8	to	be	or	not	to	<i>null</i>	<i>null</i>	<i>null</i>	
-	to	4	8	to	be	or	not	<i>null</i>	<i>null</i>	<i>null</i>	<i>null</i>	
be		5	8	to	be	or	not	be	<i>null</i>	<i>null</i>	<i>null</i>	
-	be	4	8	to	be	or	not	<i>null</i>	<i>null</i>	<i>null</i>	<i>null</i>	
-	not	3	8	to	be	or	<i>null</i>	<i>null</i>	<i>null</i>	<i>null</i>	<i>null</i>	
that		4	8	to	be	or	that	<i>null</i>	<i>null</i>	<i>null</i>	<i>null</i>	
-	that	3	8	to	be	or	<i>null</i>	<i>null</i>	<i>null</i>	<i>null</i>	<i>null</i>	
-	or	2	4	to	be	<i>null</i>	<i>null</i>					
-	be	1	2	to	<i>null</i>							
is		2	2	to	is							

Trace of array resizing during a sequence of push() and pop() operations

Stack resizing-array implementation: performance

Amortized analysis. Starting from an empty data structure, average running time per operation over a worst-case sequence of operations.

Proposition. Starting from an empty stack, any sequence of N push and pop operations takes time proportional to N .

Amortized analysis:
平均的cost

	best	worst	amortized
construct	1	1	1
push	1	N	1
pop	1	N	1
size	1	1	1

doubling and
halving operations

order of growth of running time
for resizing stack with N items

Stack implementations: resizing array vs. linked list

Tradeoffs.

Can implement a stack with either resizing array or linked list; client can use interchangeably.

Which one is better?



Stack resizing-array implementation: memory usage

Q. How much memory does a `ResizingArrayStackOfStrings` use to store N strings in the best case? Worst case?

Count only the memory owned by the stack (not the strings themselves).

Stack resizing-array implementation: memory usage

Proposition. Uses between $\sim 8 N$ and $\sim 32 N$ bytes to represent a stack with N items.

- $\sim 8 N$ when full.
- $\sim 32 N$ when one-quarter full.

```
public class ResizingArrayStackOfStrings
{
    private String[] s; ← 8 bytes × array size
    private int N = 0;
    ...
}
```

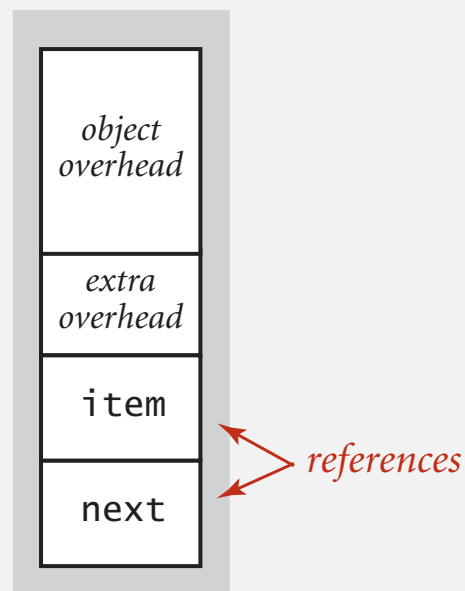
Stack: linked-list implementation performance

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

Something You Need to Know **Object Loitering**

```
public class LinkedStackOfStrings
{
    private Node first = null;

    private class Node
    {
        String item;
        Node next;
    }

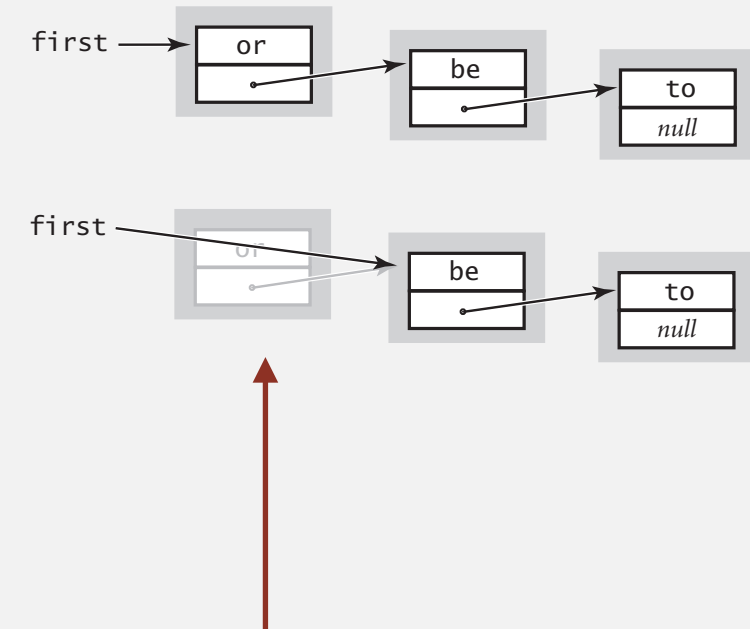
    public boolean isEmpty()
    { return first == null; }

    public void push(String item)
    {
        Node oldfirst = first;
        first = new Node();
        first.item = item;
        first.next = oldfirst;
    }

    public String pop()
    {
        String item = first.item;
        first = first.next;
        return item;
    }
}
```

delete first node

first = first.next;



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

Stack implementations: resizing array vs. linked list

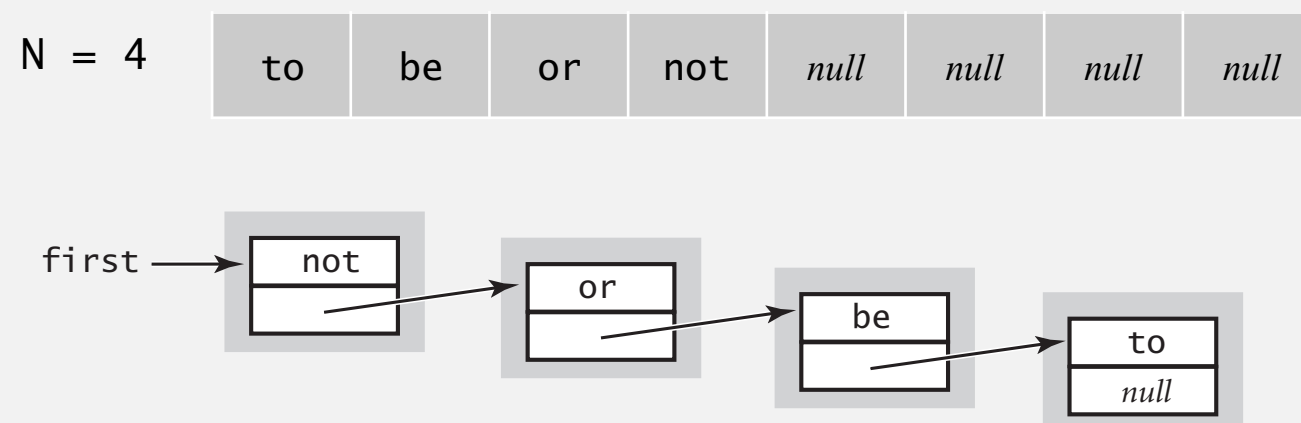
Tradeoffs. Can implement a stack with either resizing array or linked list; client can use interchangeably. Which one is better?

Linked-list implementation.

- Every operation takes constant time in the **worst case**.
- Uses extra time and space to deal with the links.

Resizing-array implementation.

- Every operation takes constant **amortized** time.
- Less wasted space.



Java collections library

List interface. `java.util.List` is API for an sequence of items.

<https://docs.oracle.com/javase/7/docs/api/index.html?java/util/package-tree.html>

public interface List<Item> implements Iterable<Item>		
	List()	<i>create an empty list</i>
boolean	isEmpty()	<i>is the list empty?</i>
int	size()	<i>number of items</i>
void	add(Item item)	<i>append item to the end</i>
Item	get(int index)	<i>return item at given index</i>
Item	remove(int index)	<i>return and delete item at given index</i>
boolean	contains(Item item)	<i>does the list contain the given item?</i>
Iterator<Item>	iterator()	<i>iterator over all items in the list</i>
...		

?

`java.util.ArrayList`
implementation

`java.util.LinkedList`
implementation

Which one is better ? What is the difference ?

java.util.ArrayList v.s. **java.util.LinkedList**



Lesson. Don't use a library until you understand its API!

```
public interface List<Item> implements Iterable<Item>
```

<code>List()</code>	<i>create an empty list</i>
<code>boolean isEmpty()</code>	<i>is the list empty?</i>
<code>int size()</code>	<i>number of items</i>
<code>void add(Item item)</code>	<i>append item to the end</i>
<code>Item get(int index)</code>	<i>return item at given index</i>
<code>Item remove(int index)</code>	<i>return and delete item at given index</i>
<code>boolean contains(Item item)</code>	<i>does the list contain the given item?</i>
<code>Iterator<Item> iterator()</code>	<i>iterator over all items in the list</i>
<code>...</code>	

Implementations. `java.util.ArrayList` uses resizing array;
`java.util.LinkedList` uses linked list.