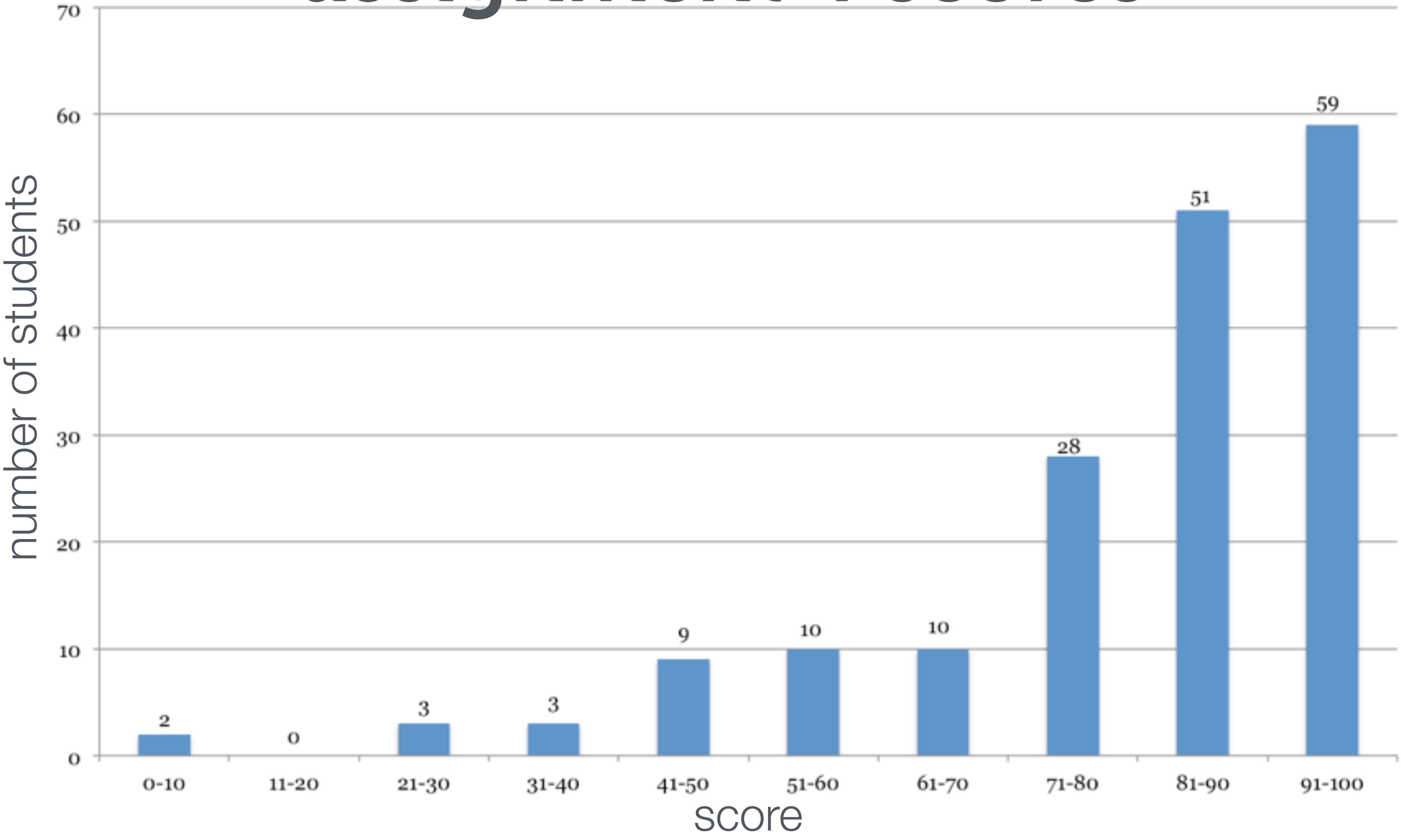
STACKS

cs2420 | Introduction to Algorithms and Data Structures | Spring 2015

administrivia…

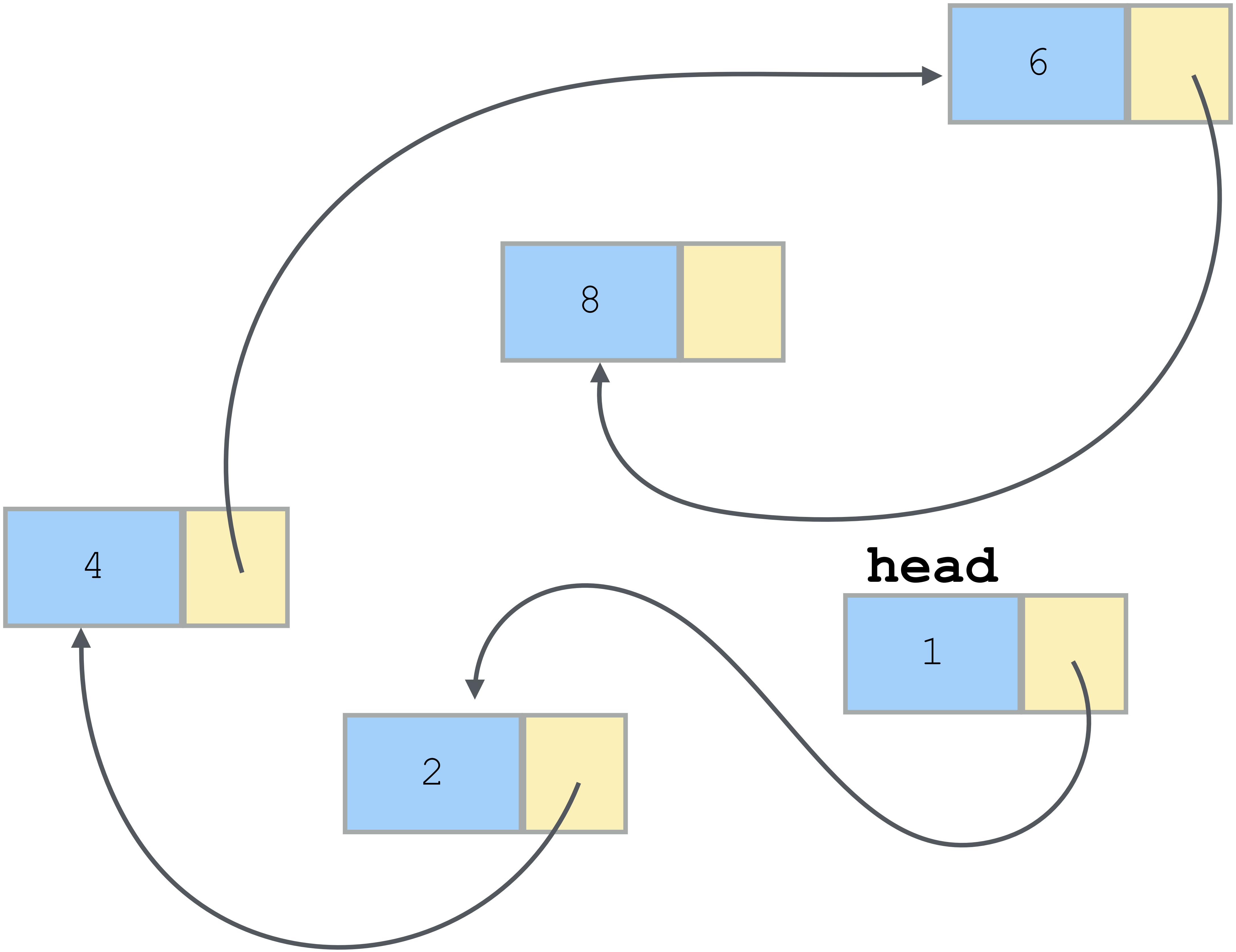
-assignment 6 due on Thursday at midnight

assignment 4 scores



last time…

linked lists


6
8
4
head
1
2

linked list vs **array**

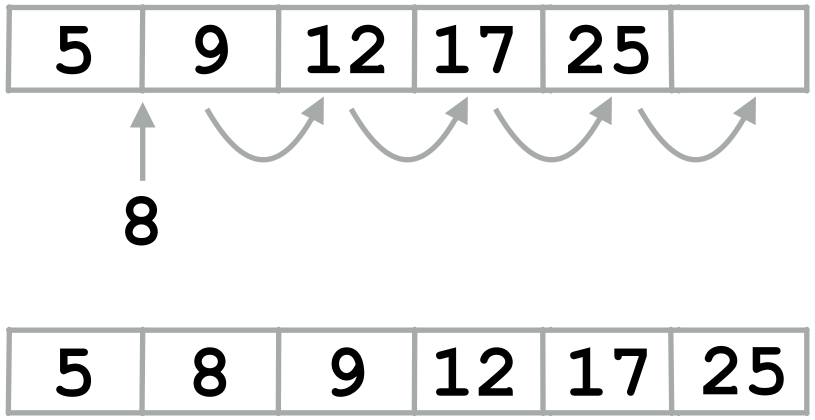
-cost of accessing a random item at location `i`?

-cost of `removeFirst()`?

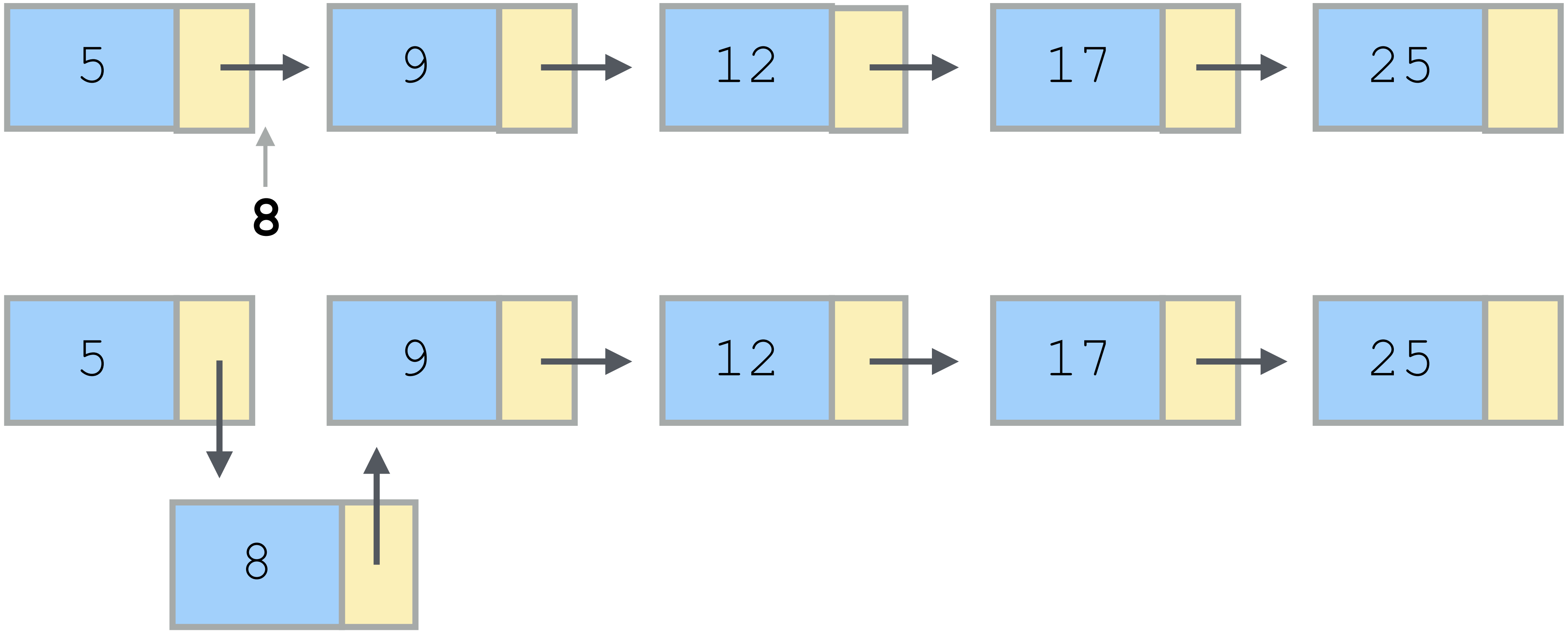
-cost of `addFirst()`?

A) **c**
B) **log N**
C) **N**
D) **N log N**
E) **N^2**
F) **N^3**

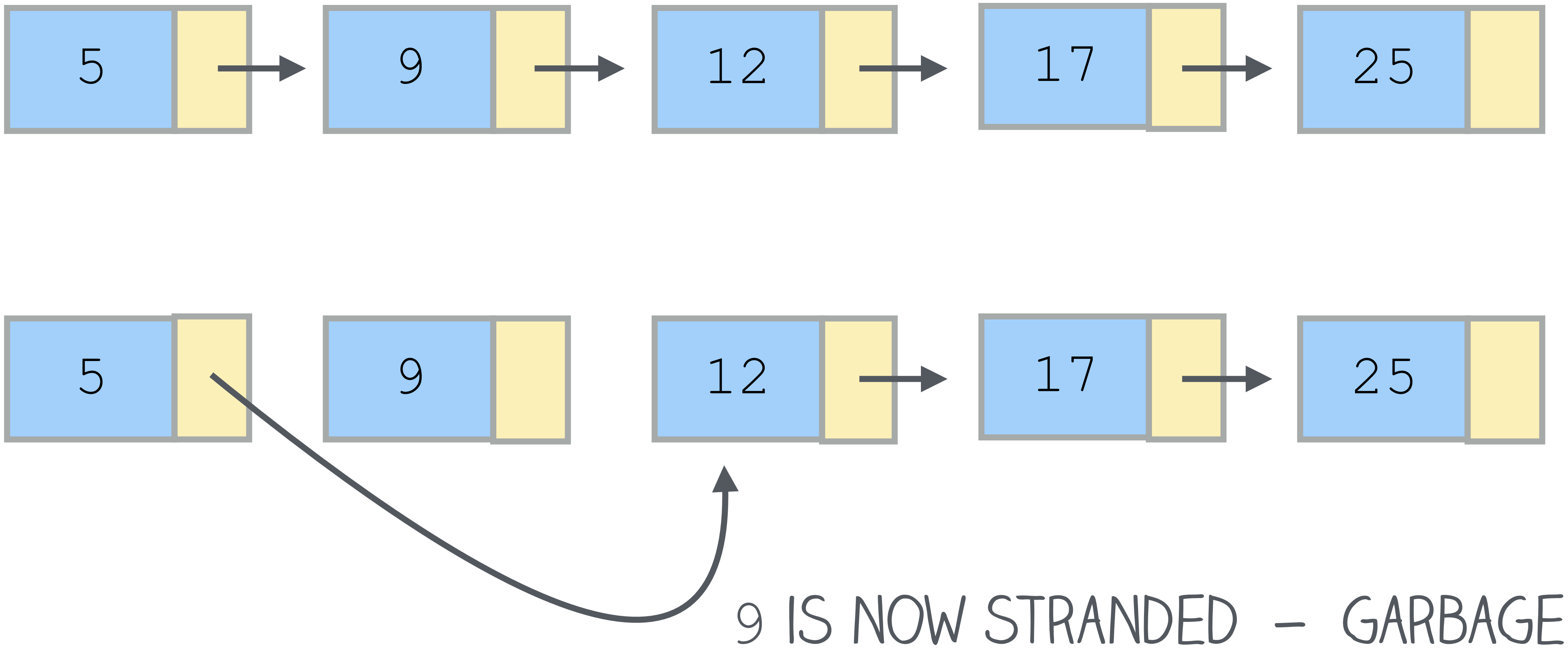
inserting into an array:



inserting into a linked list:



deletion from a linked list:



9 IS NOW STRANDED – GARBAGE COLLECTOR WILL CLEAN IT UP

doubly-linked lists

-nodes have a link to `next` *and* `previous` node

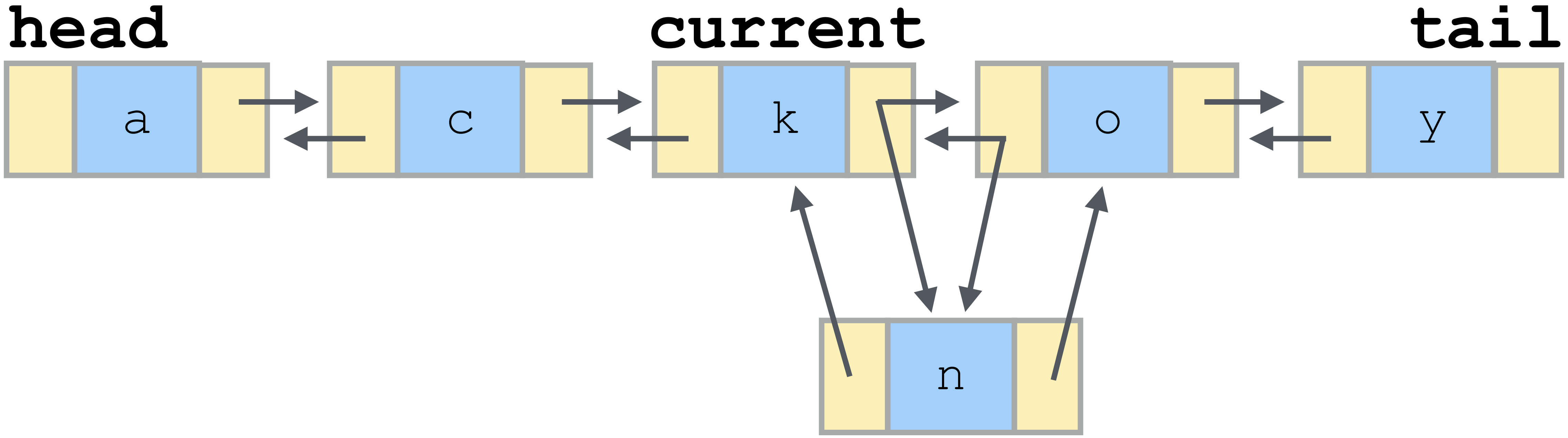
-allows for traversal in either forward or reverse order

-maintains a `tail` node as well as a `head` node

doubly-linked list insertion:

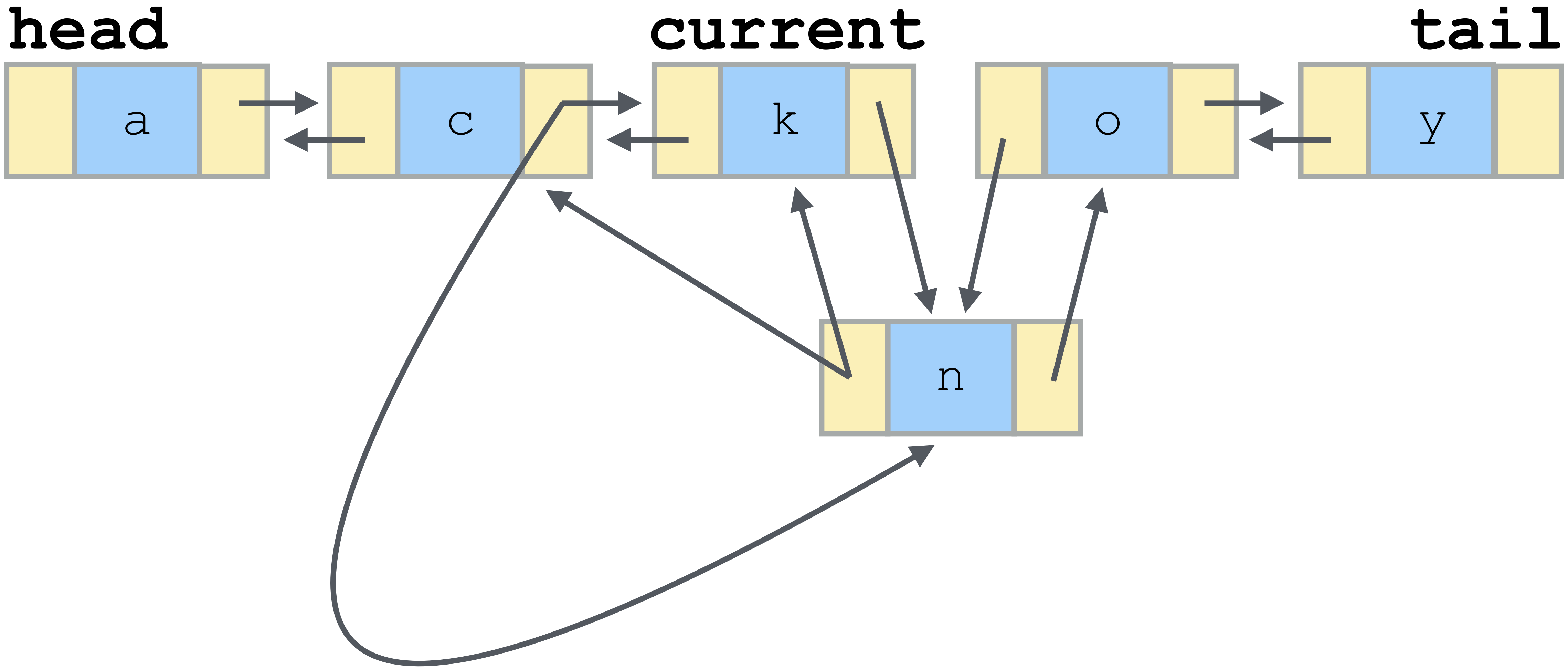
```
newNode = new Node<Character>();
newNode.data = ’n’;

newNode.prev = current;
newNode.next = current.next;
newNode.prev.next = newNode;
newNode.next.prev = newNode;
```



doubly-linked list deletion:

```
current.prev.next = current.next;
current.next.prev = current.prev;
```



LinkedList VS ArrayList

insertion & deletion: (assuming position is known)	**O(c)**	**O(N)**
accessing a random item:	**O(N)**	**O(c)**

- choose the structure based on the expected use
    - what is the common case?

today…

stacks

-a **stack** is a data structure in which insertion and removal is restricted to the **top** (or end) of the list

-also called FIRST-IN, LAST-OUT (FILO)

- -insertion always adds an item to the end
- -deletion always removes an item from the end

important methods

-`push`

-inserts an item on to the top of the stack

-`pop`

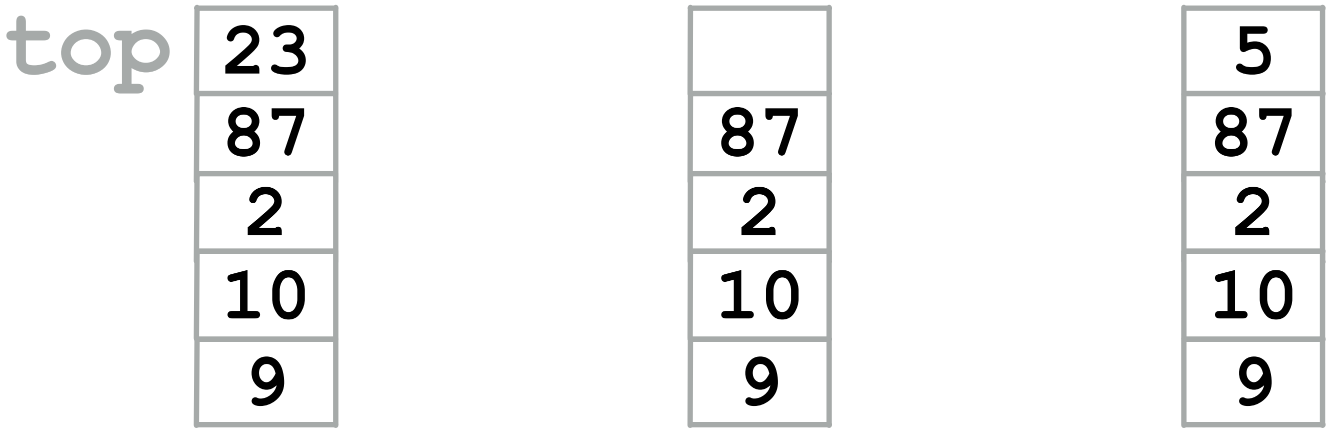
-removes and returns the item on the top of the stack

-`peek`

-returns but does not remove the top of the stack

-consecutive calls to `pop` wil return items in the reverse order that they were `pushed`

```
pop();
push(5);
```



IT IS USEFUL TO THINK OF STACKS AS STANDING UPRIGHT!
(LIKE A STACK OF DISHES)

performance

-`push`, `pop`, and `peek` must all be **O(1)**

-we need a very efficient data structure if we expect to only access the last element

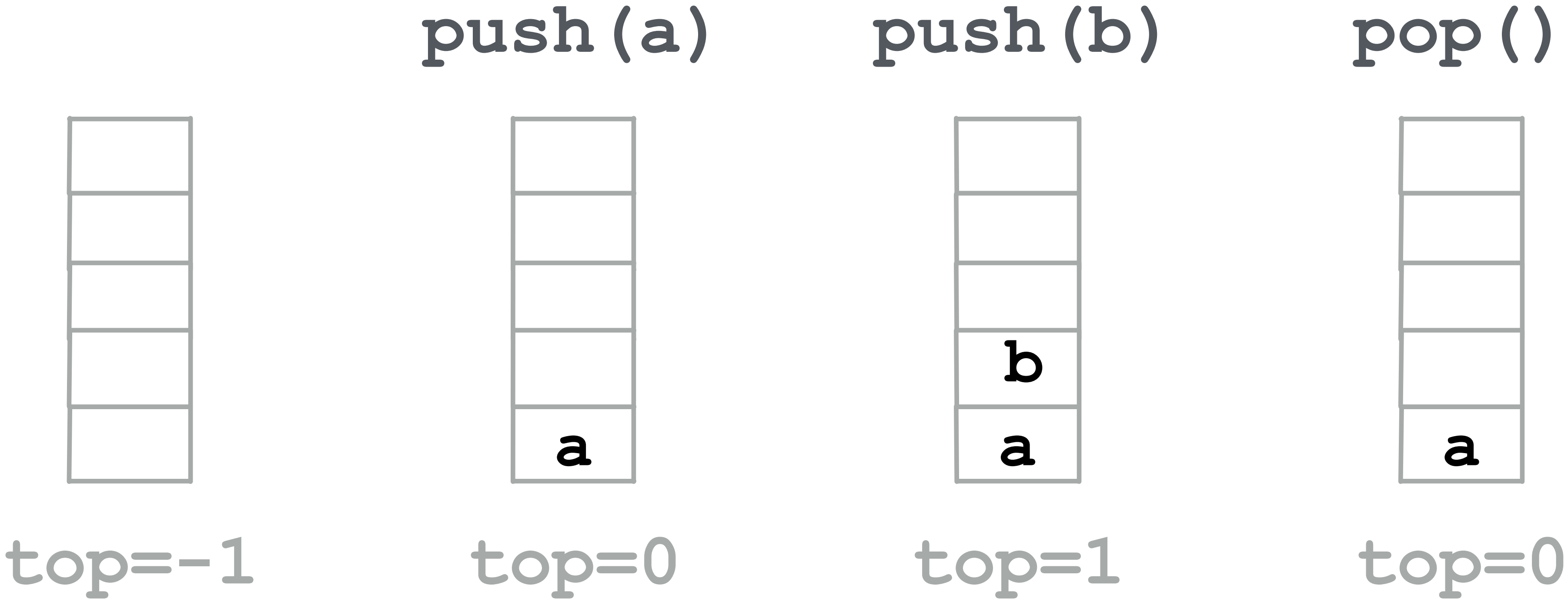
HOW CAN WE IMPLEMENT A STACK SO THAT ALL 3 OPERATIONS ARE GUARANTEED TO BE **O(1)**?

as an array...

-NOTE: keep track of a `top` index

-to `push`, increment `top`, then add the item at that index

-to `pop`, return the item at index `top`, and decrement `top`



performance

-if we try to `push` when the underlying array is full, the array must be grown

-any `push` that requires resizing the array takes **O(N)** time

-all other operations are constant, **O(1)**

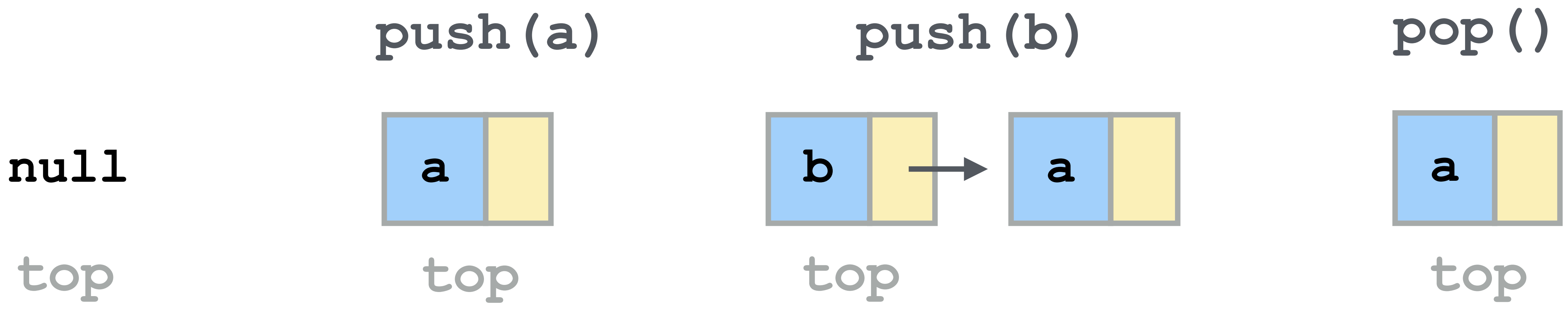
-since `pushes` that resize the array are rare, the average case for push is still **O(1)**

as a linked list…

-treat the `head` as the `top` of the stack

-to `push`, add to the beginning of the linked list

-to `pop`, return the `top` and remove the first item



performance

-linked lists never incur the penalty or resizing

- -adds to a linked list are always **O(1)**

-no wasted extra array space

-all stack operations are **O(1)**

-a stack can be easily implemented on top of an existing linked list with very little extra code!

EXAMPLE: call stack (again!)

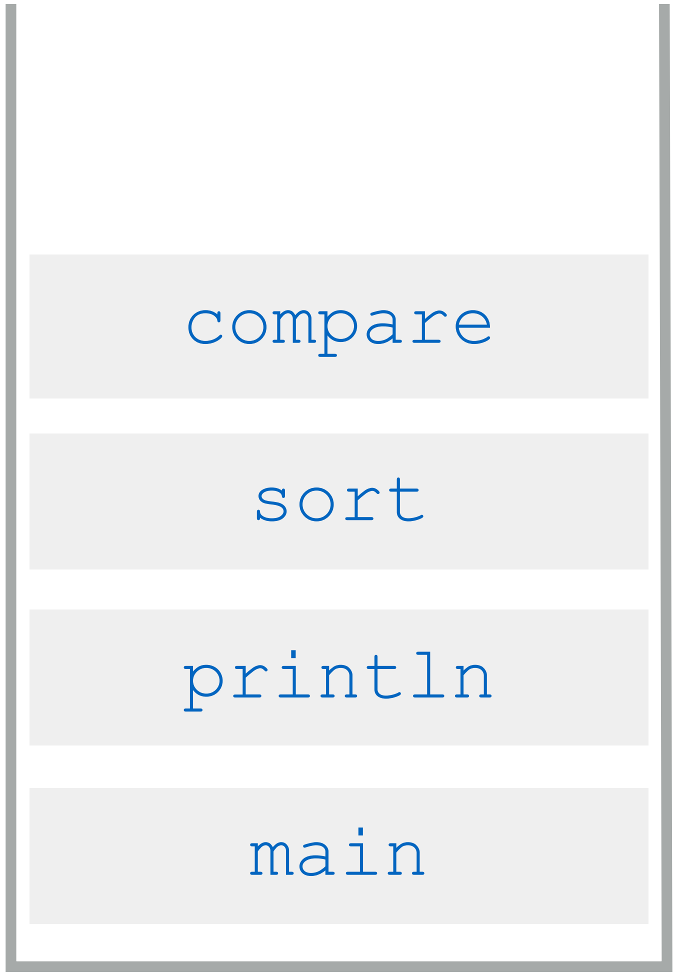
-every time a method is invoked a unique *frame* is created

-when that method returns, execution resumes in the calling frame

-methods return in reverse order in which they were called

  -FILO!

  -what method is the first in and last out?



compare

sort

println

main

call stack

EXAMPLE: symbol matcher

-part of the compilation process for Java’s compiler (and many others) is **symbol matching**

-every `{` must be matched with a corresponding `}`

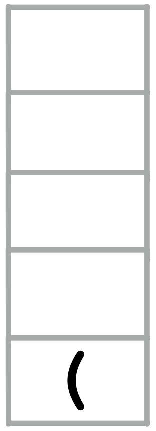
- -same for `()` and `[]`

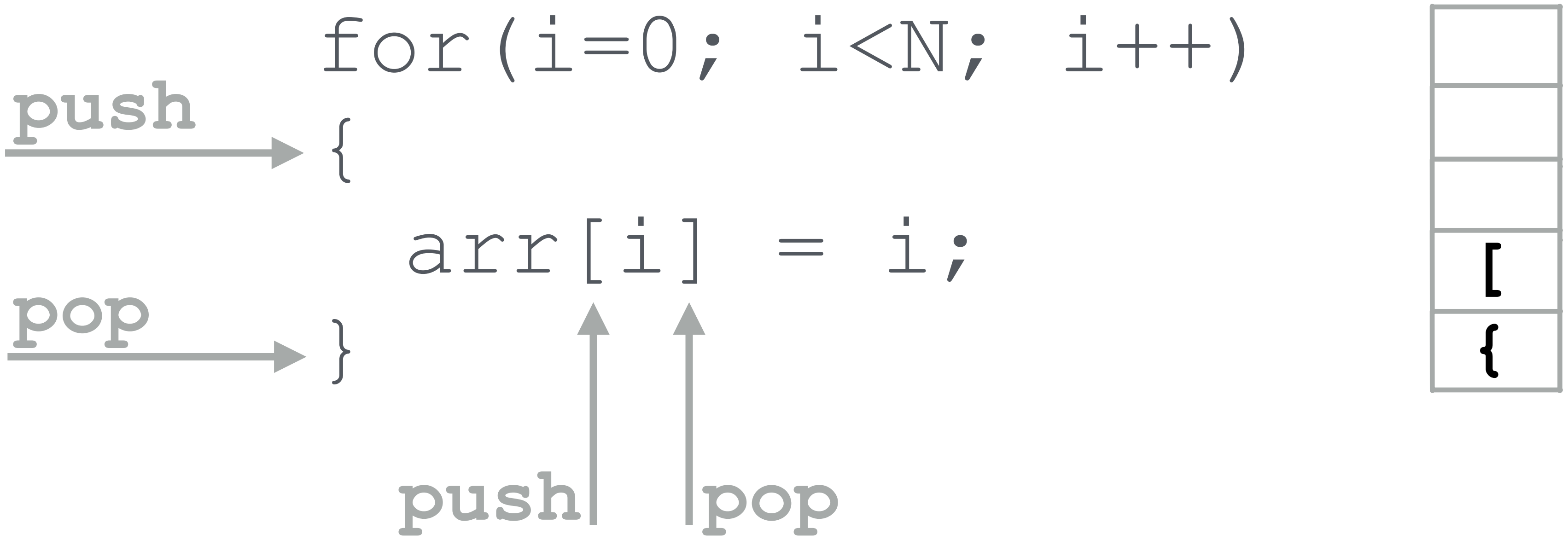
-how can we use a stack to determine if all brace symbols are matched?

```
for(i=0; i<N; i++)
{
  arr[i] = i;
}
```

push pop

```
for(i=0; i<N; i++)
{
  arr[i] = i;
}
```





```
for(i=0; i<N; i++)
{
  arr[i] = i;
}
```

IF END OF INPUT IS REACHED AND THE STACK IS EMPTY...
ALL THE SYMBOLS ARE BALANCED!

next time…

- reading
  - chapter 16
  - chapter 2
    - *http://opendatastructures.org/ods-java/*

- homework
  - assignment 6 due Thursday