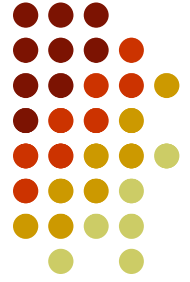
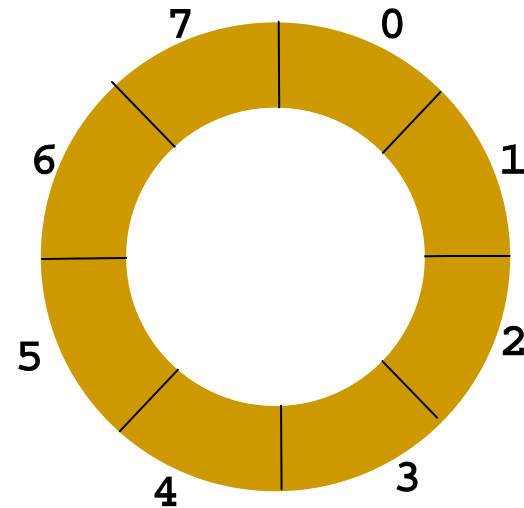
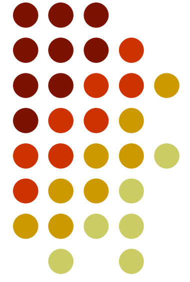


Circular Arrays

- **Neat trick**: use a ***circular array*** to insert and remove items from a queue in constant time
- The idea of a circular array is that the end of the array "wraps around" to the start of the array

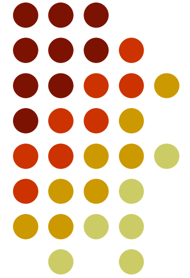


The mod Operator



- The `mod` operator (%) is used to calculate remainders:
  - `1%5 = 1, 2%5 = 2, 5%5 = 0, 8%5 = 3`
- `mod` can be used to calculate the front and back positions in a circular array, therefore avoiding comparisons to the array size
  - The back of the queue is:
    - `(front + count - 1) % items.length`
    - where `count` is the number of items currently in the queue
  - After removing an item the front of the queue is:
    - `(front + 1) % items.length;`

Array Queue Example



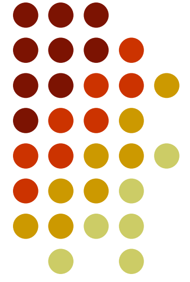
front =	0
count =	1

6					
0	1	2	3	4	5

```
//Java Code
Queue q = new Queue();
q.enqueue(6);
```

insert item at (front + count) % items.length

Array Queue Example

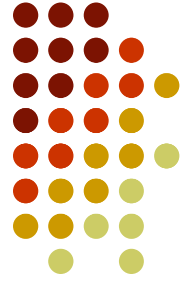


front =	0
count =	5

6	4	7	3	8	
0	1	2	3	4	5

```
//Java Code
Queue q = new Queue();
q.enqueue(6);
q.enqueue(4);
q.enqueue(7);
q.enqueue(3);
q.enqueue(8);
```

Array Queue Example



front	=	2
count	=	4

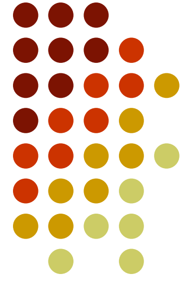
6	4	7	3	8	9
0	1	2	3	4	5

make front = (0 + 1) % 6 = 1

make front = (1 + 1) % 6 = 2

```java
//Java Code
Queue q = new Queue();
q.enqueue(6);
q.enqueue(4);
q.enqueue(7);
q.enqueue(3);
q.enqueue(8);
q.dequeue();//front = 1
q.dequeue();//front = 2
q.enqueue(9);
```

Array Queue Example



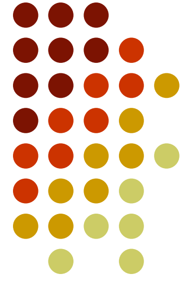
front =	2
count =	5

5		7	3	8	9
0	1	2	3	4	5

```
//Java Code
Queue q = new Queue();
q.enqueue(6);
q.enqueue(4);
q.enqueue(7);
q.enqueue(3);
q.enqueue(8);
q.dequeue();//front = 1
q.dequeue();//front = 2
q.enqueue(9);
q.enqueue(5);
```

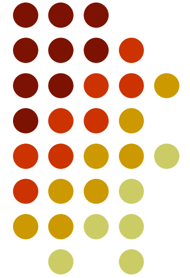
insert at (front + count) % 6
= (2 + 4) % 6 = 0

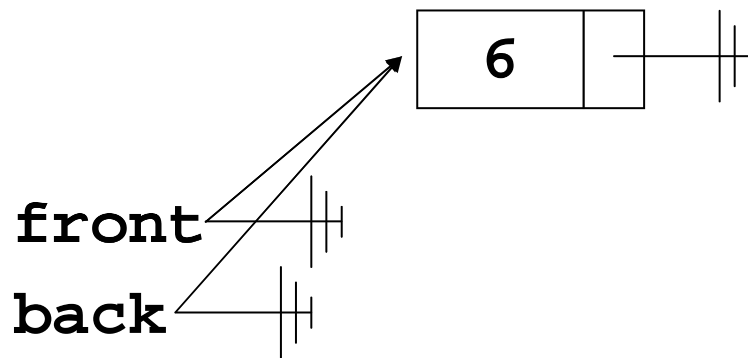
Queue: Linked List Implementation



- Removing items from the front of the queue is straightforward
- But we need to insert items at the back of the queue in constant time
  - So cannot traverse through the list
  - By using an additional reference (and a little administration) we can keep track of the node at the back of the queue

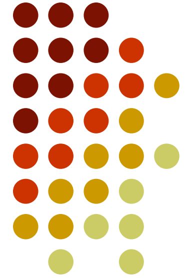
List Queue Example

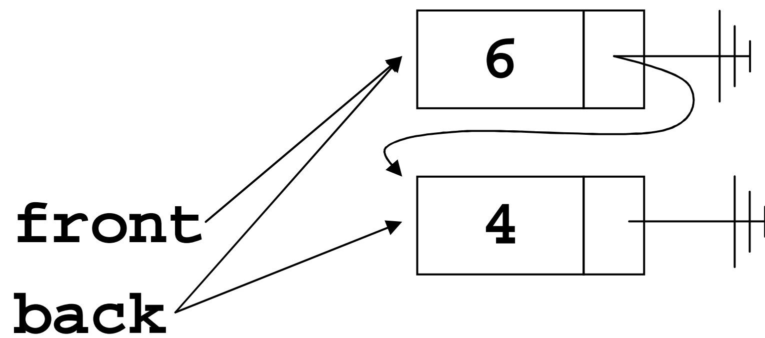





```
//Java Code
Queue q = new Queue();
q.enqueue(6);
```

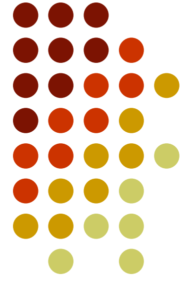
List Queue Example

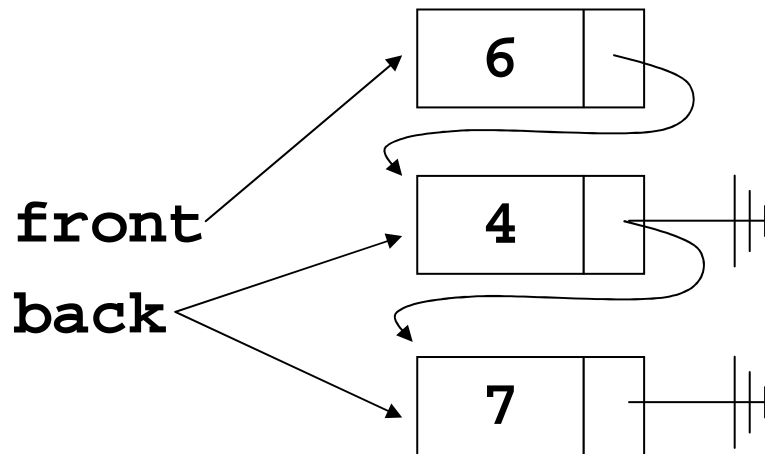





```
//Java Code
Queue q = new Queue();
q.enqueue(6);
q.enqueue(4);
```

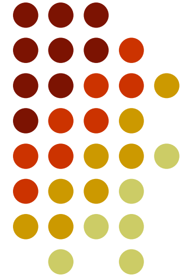
List Queue Example

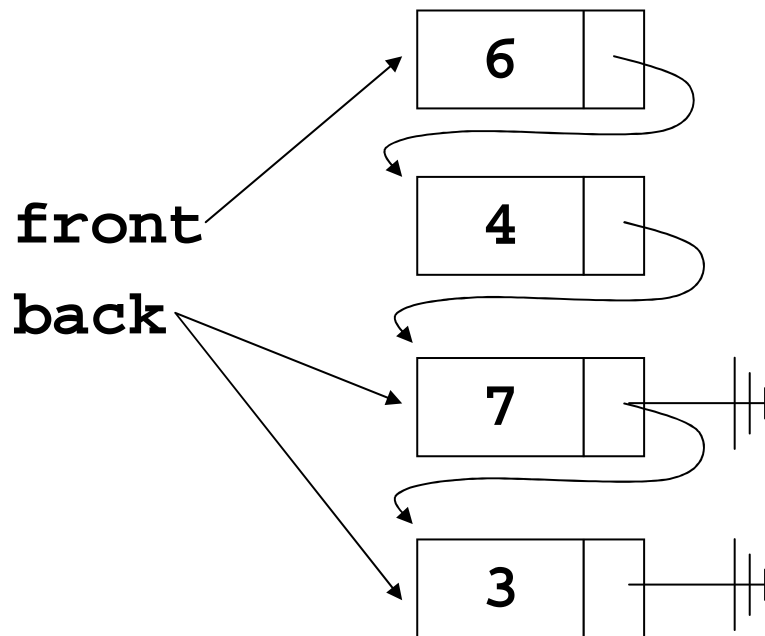





```java
//Java Code
Queue q = new Queue();
q.enqueue(6);
q.enqueue(4);
q.enqueue(7);
```

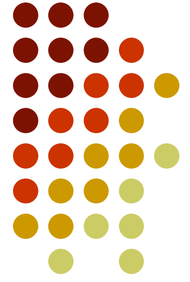
List Queue Example

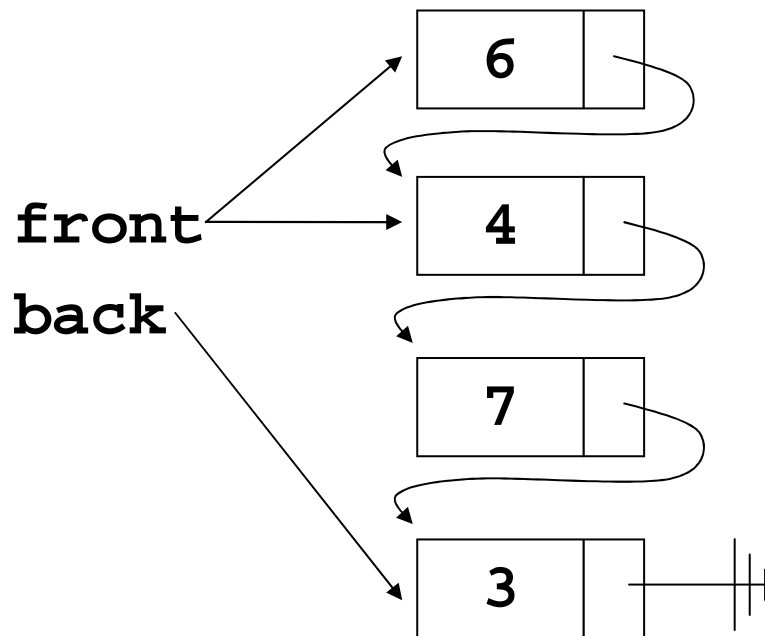





```
//Java Code
Queue q = new Queue();
q.enqueue(6);
q.enqueue(4);
q.enqueue(7);
q.enqueue(3);
```

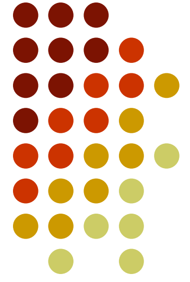
List Queue Example






```
//Java Code
Queue q = new Queue();
q.enqueue(6);
q.enqueue(4);
q.enqueue(7);
q.enqueue(3);
q.dequeue();
```

Queue: Circular Linked List Implementation



- Possible implementations of a queue
  - A circular linked list with one external reference
    - A reference to the back

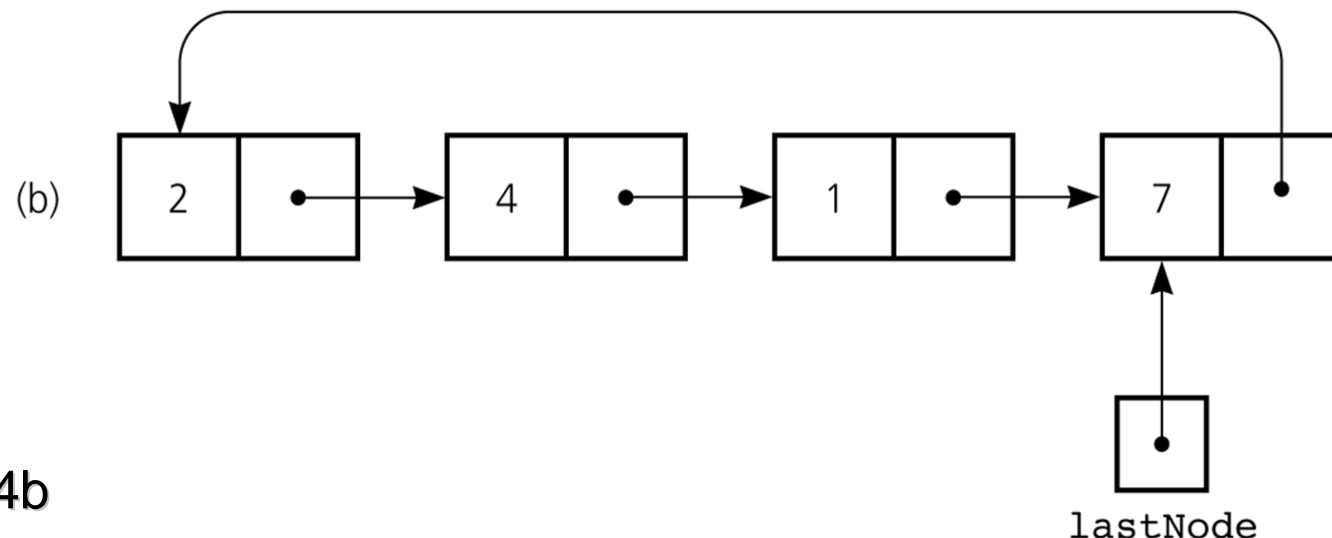
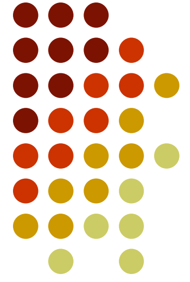


Figure 8-4b

A reference-based implementation of a queue: b) a circular linear linked list with one external reference

Queue: Circular Linked List Implementation



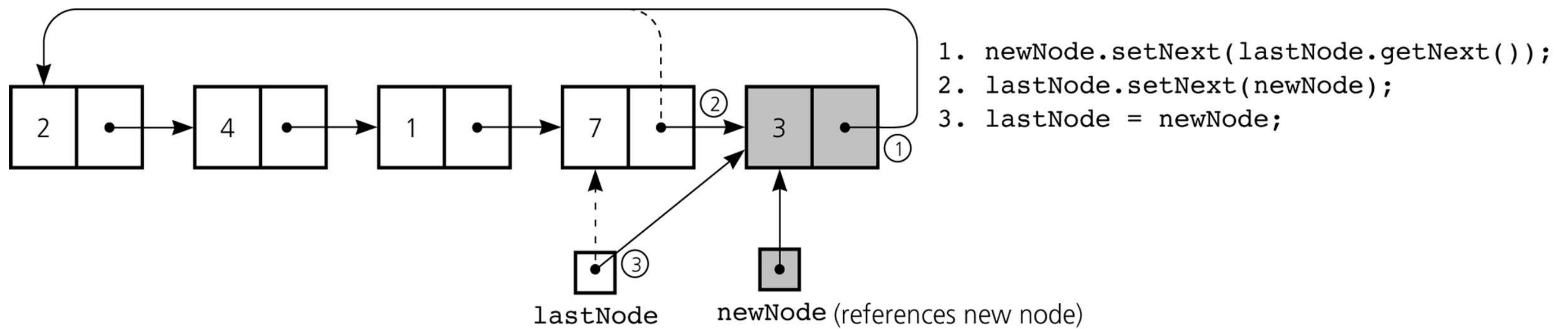
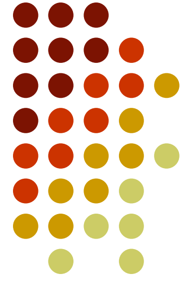


Figure 8-5

Inserting an item into a nonempty queue

Queue: Circular Linked List Implementation



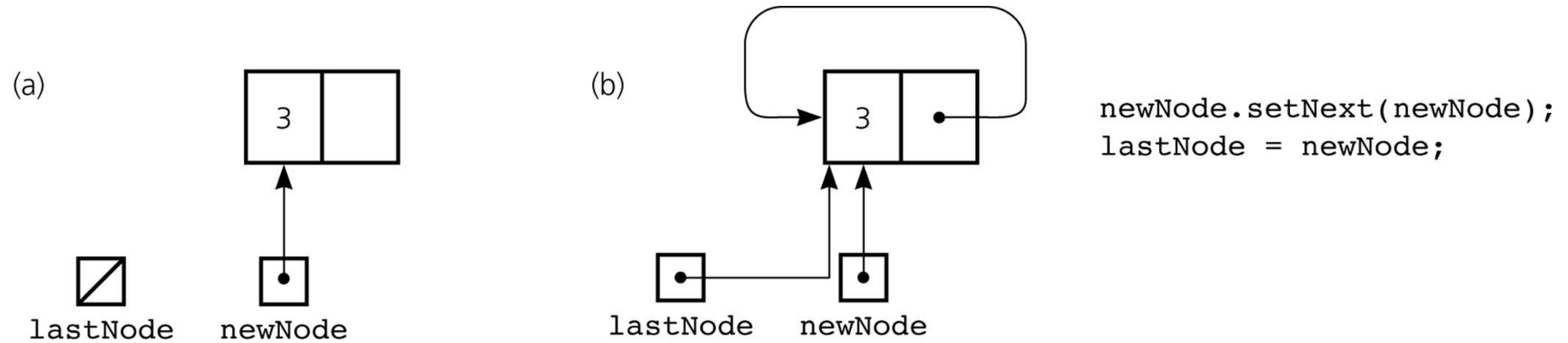
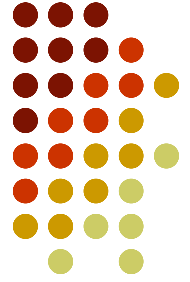


Figure 8-6

Inserting an item into an empty queue: a) before insertion; b) after insertion

Queue: Circular Linked List Implementation



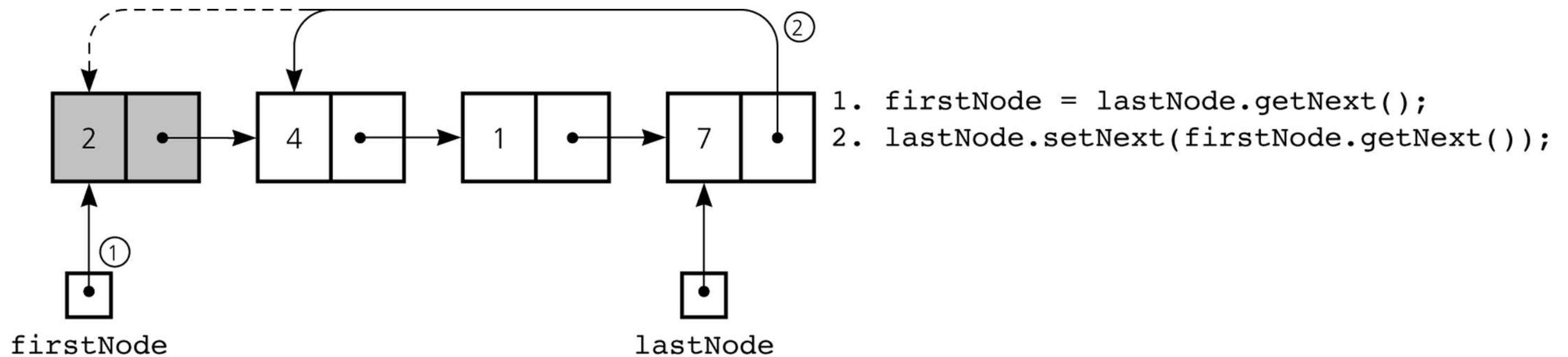
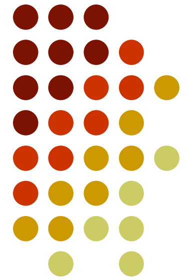


Figure 8-7

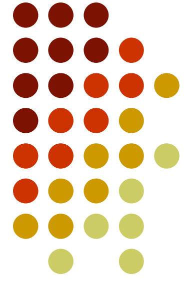
Deleting an item from a queue of more than one item

Queue: ADT List Implementation



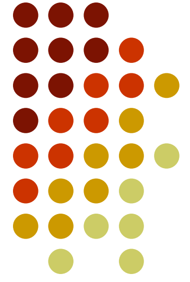
```
public void enqueue(Object newItem) {
   list.add(list.size()+1, newItem);
}  // end enqueue
public Object dequeue() {
   Object temp = list.get(1);
   list.remove(1);
   return temp;
}  // end dequeue
```

Queue: ADT List Implementation

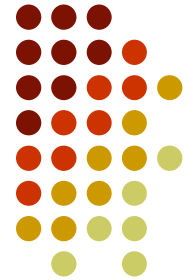


- Efficiency depends on implementation of ADT List – in most common implementations, at least one of operations *enqueue()* and *dequeue()* is not efficient
- On other hand: it was very fast to implement (code is easy, unlikely that errors were introduced when coding).

Application of queues: Recognizing Palindromes



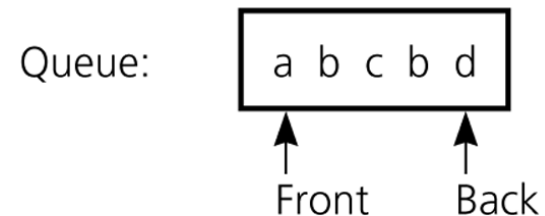
- A palindrome
  - A string of characters that reads the same from left to right as its does from right to left
- To recognize a palindrome, a queue can be used in conjunction with a stack
  - A stack can be used to reverse the order of occurrences
  - A queue can be used to preserve the order of occurrences



Recognizing Palindromes

- A nonrecursive recognition algorithm for palindromes
  - As you traverse the character string from left to right, insert each character into both a queue and a stack
  - Compare the characters at the front of the queue and the top of the stack

String: abcbd



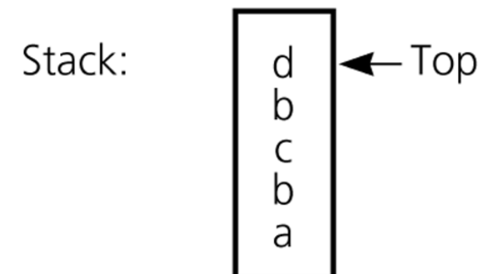
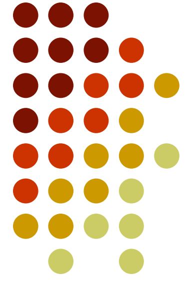


Figure 8-3

The results of inserting a string into both a queue and a stack

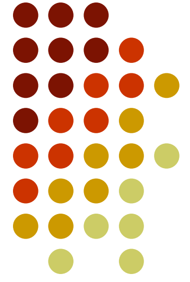
Problems:



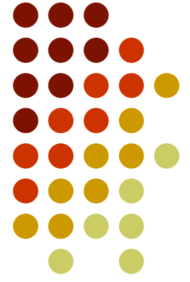
- Recognize palindromes using 3 stacks.
- Simulate (implement) a queue with 2 stacks. How efficient is this implementation?

- Simulate (implement) a stack with 1 queue! (This is the problem which I wanted to discuss but didn’t recall it’s formulation exactly.) How efficient is this implementation?

Recursion

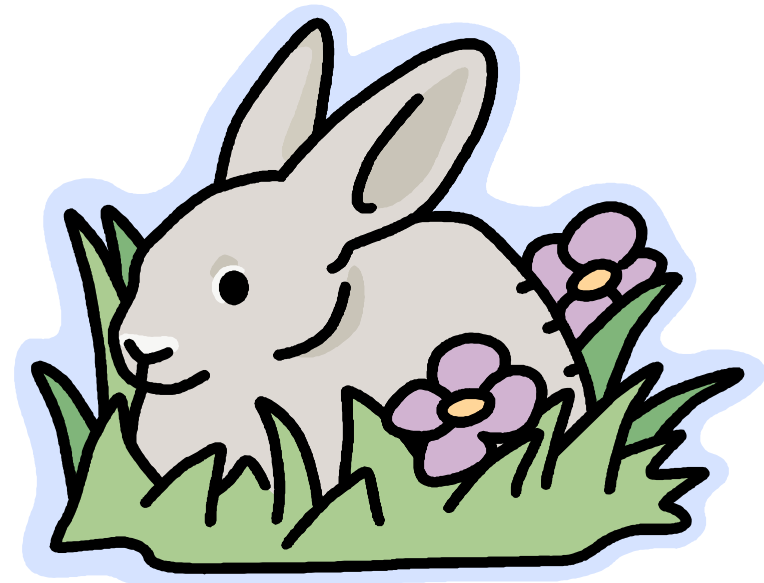


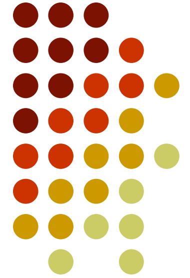
- An extremely powerful problem-solving technique
- Breaks a problem in smaller identical problems
- An alternative to iteration
    - An iterative solution involves loops



Example: Rabbits

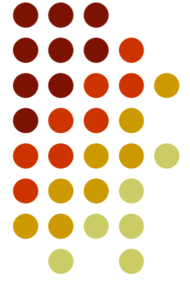
- What happens if you put a pair of rabbits in a field?
  - You get more rabbits!
- Assume that rabbits take one month to reach maturity and that
- Each pair of rabbits produces another pair of rabbits one month after mating.
- **That is:** each pair will produce a new pair 2 months after it was born (and every month after that)





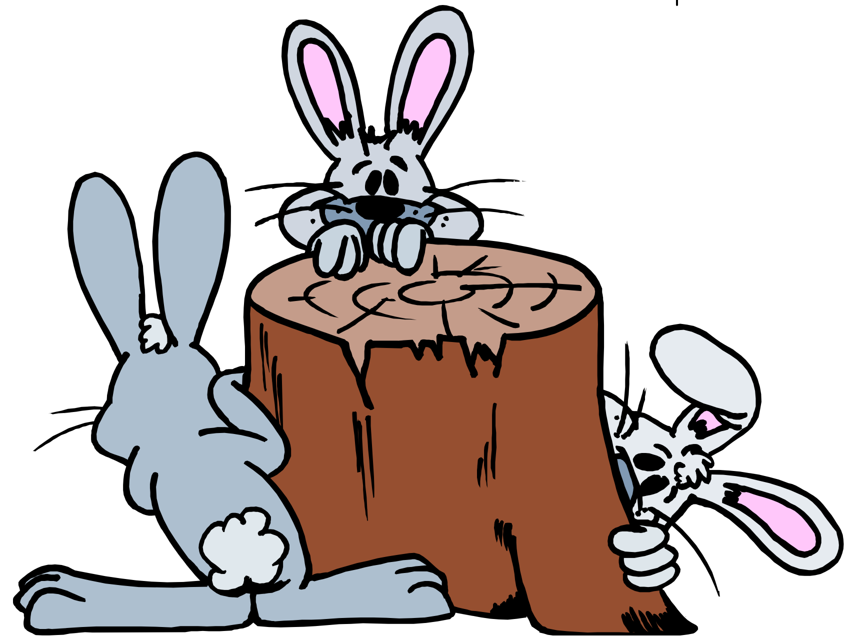
Example: Rabbits

- How many pairs of rabbits are there after five months?
- Month 1: 1 pair
- Month 2: 1 pair
  - after one month the rabbits are mature and can mate
- Month 3: 2 pairs
  - the first pair gives birth to a new pair of rabbits
- Month 4: 3 pairs
  - the first pair gives birth to a new pair, her first children are now mature
- Month 5: 5 pairs



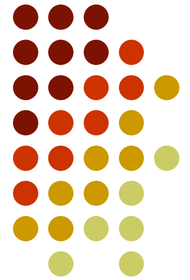
Example: Rabbits

- After 5 months there are 5 pairs of rabbits
  - i.e. the number of pairs at 4 months (**3**) plus the number of pairs at 3 months (**2**)
  - Why?
- We have count existing pairs of rabbits (the same number as previous month) + the new pairs (the same number as 2 months ago, as only those rabbits can now produce children)
- This series of numbers is called the Fibonacci series



rabbit (n) = rabbit(n-1) + rabbit(n-2)

Multiplying Rabbits (The Fibonacci Sequence)



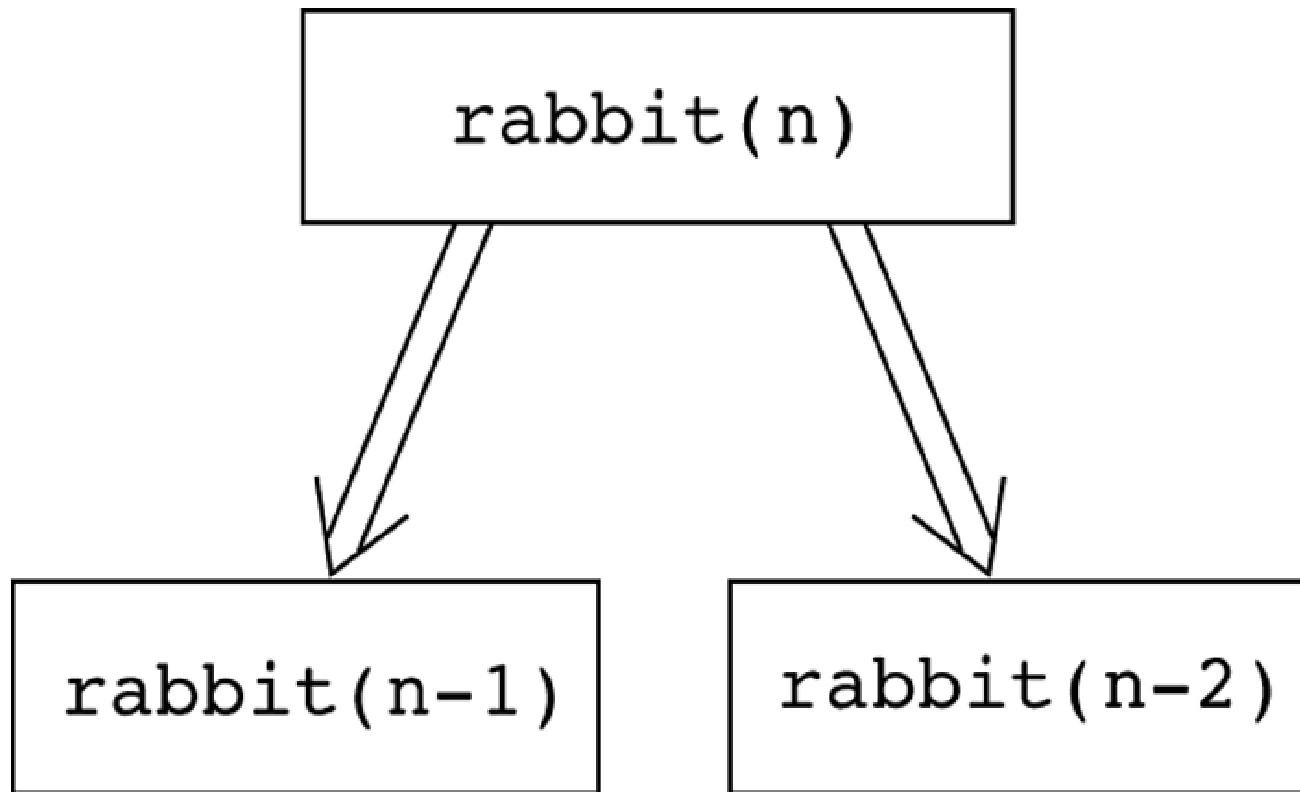
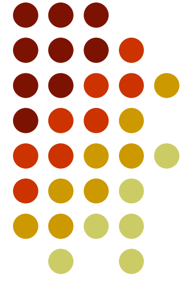


Figure 3-10

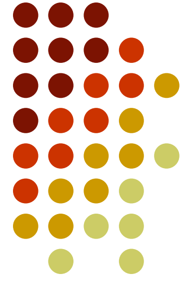
Recursive solution to the rabbit problem

Fibonacci Sequence



- The nth number in the Fibonacci sequence, $fib(n)$, is:
  - 0 if $n = 0$, and 1 if $n = 1$
  - $fib(n - 1) + fib(n - 2)$ for $n > 1$
- So what, say, is $fib(23)$, or how many pairs of rabbits would there be after 23 months?
  - This would be easy if only we knew $fib(22)$ and $fib(21)$
  - If we did then the answer is just $fib(22) + fib(21)$
  - What happens if we write a function to calculate Fibonacci numbers in this way?

Calculating the Fibonacci Sequence



- Here is a function to return nth number in the Fibonacci sequence
  - e.g. fib(1) = 1, fib(3) = 2, fib(5) = 5, fib(8) = 21, …

```
public static int fib(int n){
   if(n == 0 || n == 1){
     return n;
   }
   else{
     return fib(n-1) + fib(n-2);
   }
}
```

The function calls itself!!!

- Notice this looks just like the description of the Fibonacci sequence given previously, but does it work!?