CMSC 341

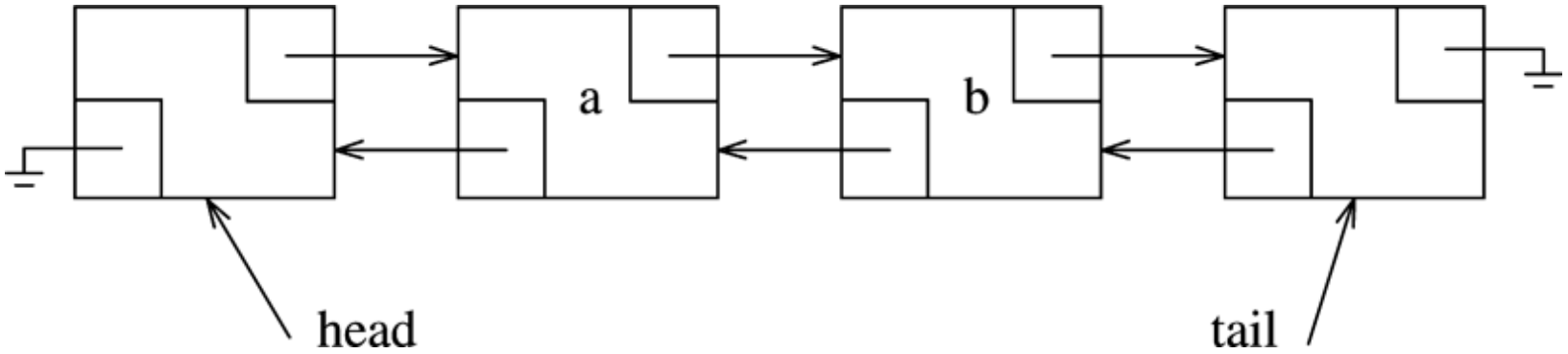
Linked Lists, Stacks and Queues

Goal of the Lecture

- To complete iterator implementation for ArrayList
- Briefly talk about implementing a LinkedList
- Introduce special Lists
  - Stacks (LIFO Data Structure)
  - Queues (FIFO Data Structure)
  - Simple Adapters to implement Stacks and Queues

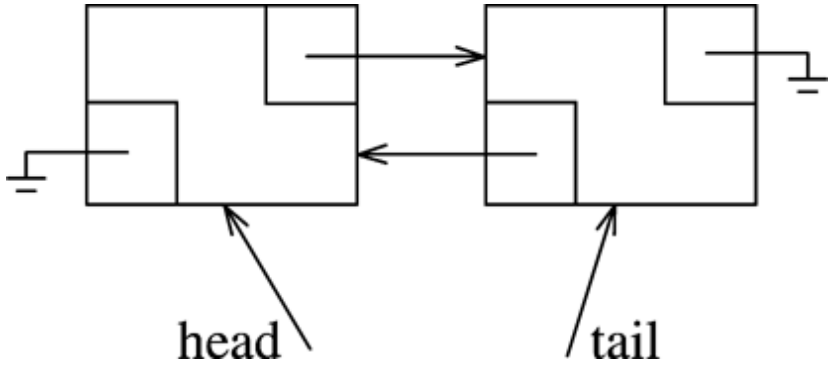
Implementing A Linked List

- To create a doubly linked list as seen below
  - MyLinkedList class
  - Node class
  - LinkedListIterator class
  - Sentinel nodes at head and tail



Empty Linked List

- An empty double linked list with sentinel nodes.



Inner classes

- Inner class objects require the construction of an outer class object before they are instantiated.
- Compiler adds an implicit reference to outer class in an inner class (MyArrayList.this).
- Good for when you need several inner objects to refer to exactly one outer object (as in an Iterator object).

Nested classes

- Considered part of the outer class, thus no issues of visibility.
- Making an inner class private means that only the outer class may access the data fields within the nested class.
- Is Node a prime candidate for nested or inner class? public or private?

Implementation for MyLinkedList

1. Let take a simple implementation of a LinkedList
    1. Look at the Node definition, Add method.
    2. Your Goal: As homework assignment is to implement an Iterator Inner class

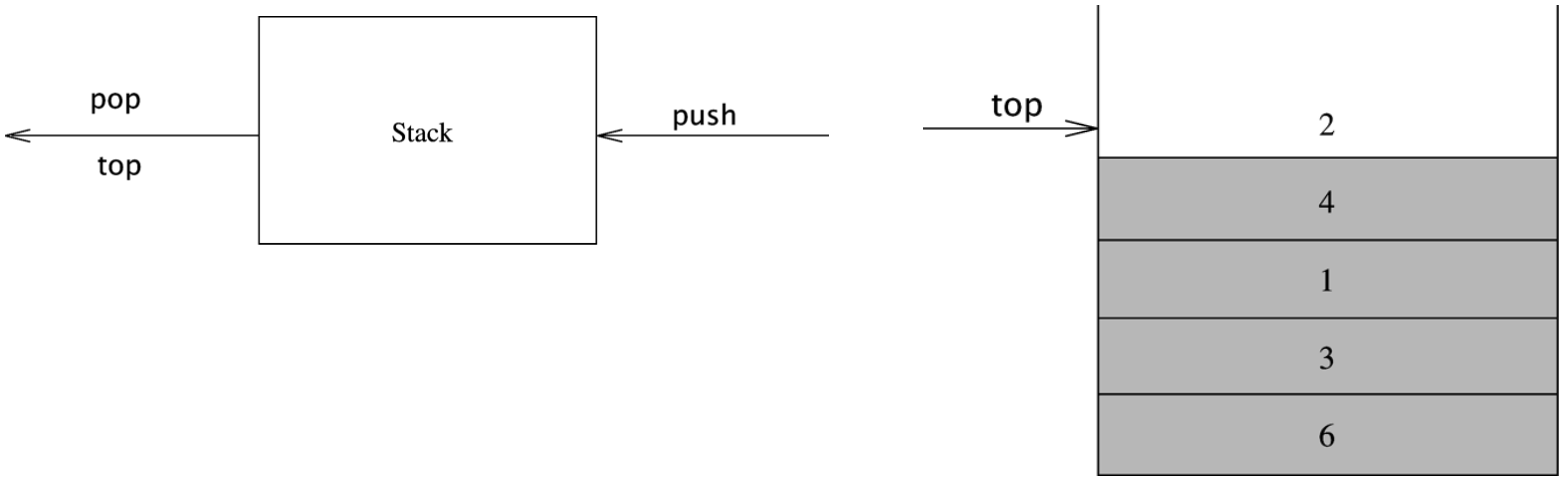
Stacks

- A restricted list where insertions and deletions can only be performed at one location, the end of the list (top).
- LIFO – Last In First Out
  - Laundry Basket – last thing you put in is the first thing you remove
  - Plates – remove from the top of the stack and add to the top of the stack

Stack ADT

- Basic operations are push, pop, and top

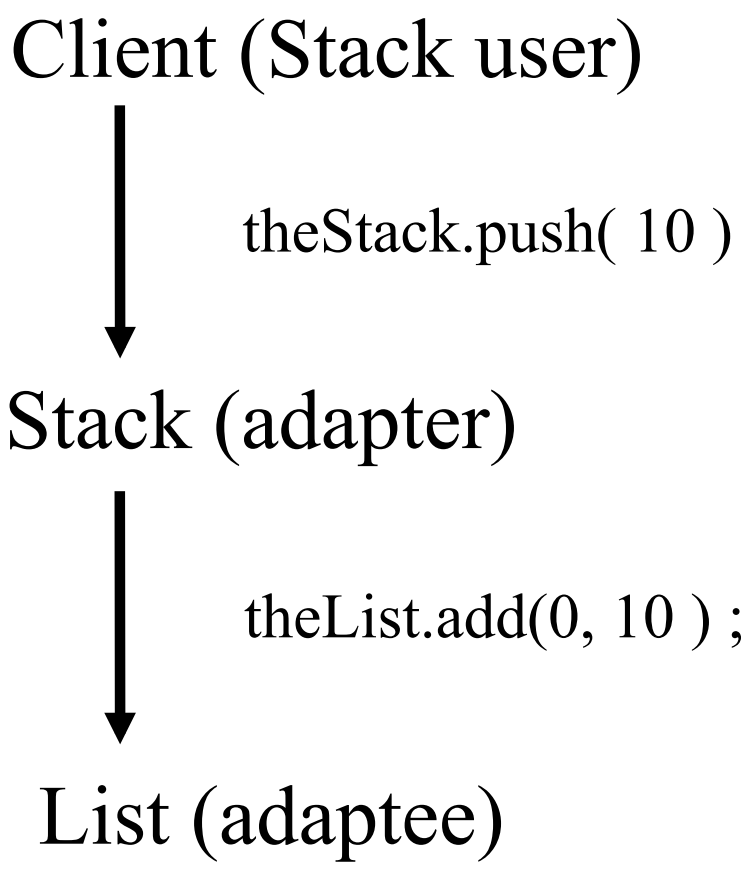
Stack Model



Adapting Lists to Implement Stacks

- Adapter Design Pattern
- Allow a client to use a class whose interface is different from the one expected by the client
- Do not modify client or class, write adapter class that sits between them
- In this case, the List is an adapter for the Stack. The client (user) calls methods of the Stack which in turn calls appropriate List method(s).

Adapter Model for Stack

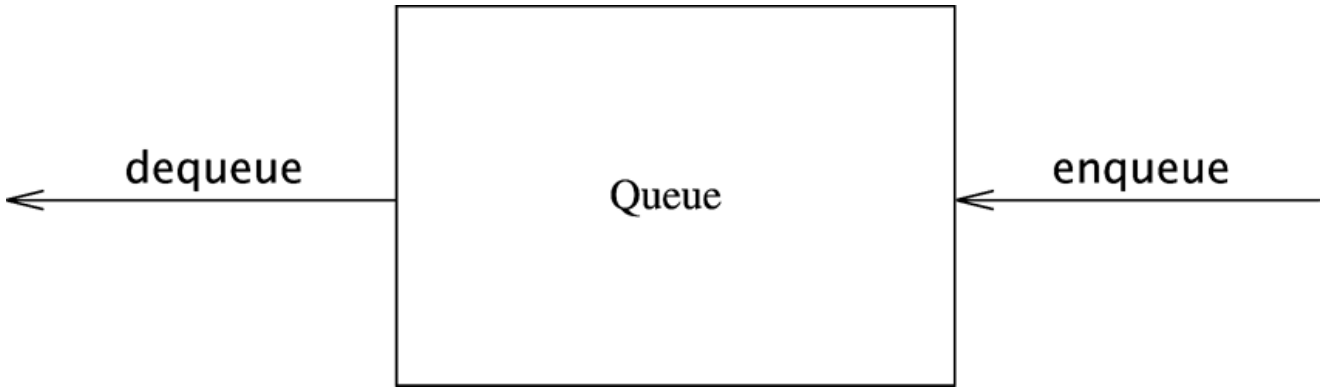


Queues

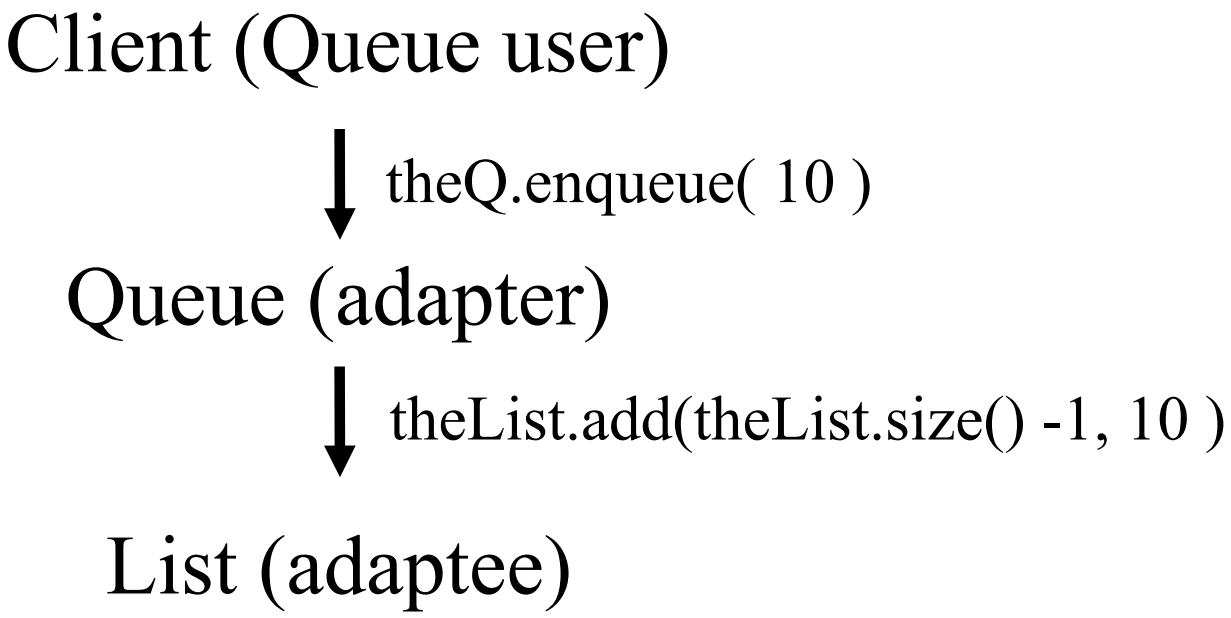
- Restricted List
  - only add to head
  - only remove from tail
- Examples
  - line waiting for service
  - jobs waiting to print
- Implement as an adapter of List

Queue ADT

- Basic Operations are enqueue and dequeue



Adapter Model for Queue



Circular Queue

- Adapter pattern may be impractical
  - Overhead for creating, deleting nodes
  - Max size of queue is often known
- A circular queue is a fixed size array
  - Slots in array reused after elements dequeued

Circular Queue Data

- A fixed size array
- Control Variables

  arraySize

  the fixed size (capacity) of the array

  currentSize

  the current number of items in the queue

  Initialized to 0

  front

  the array index from which the next item will be dequeued.

  Initialized to 0

  back

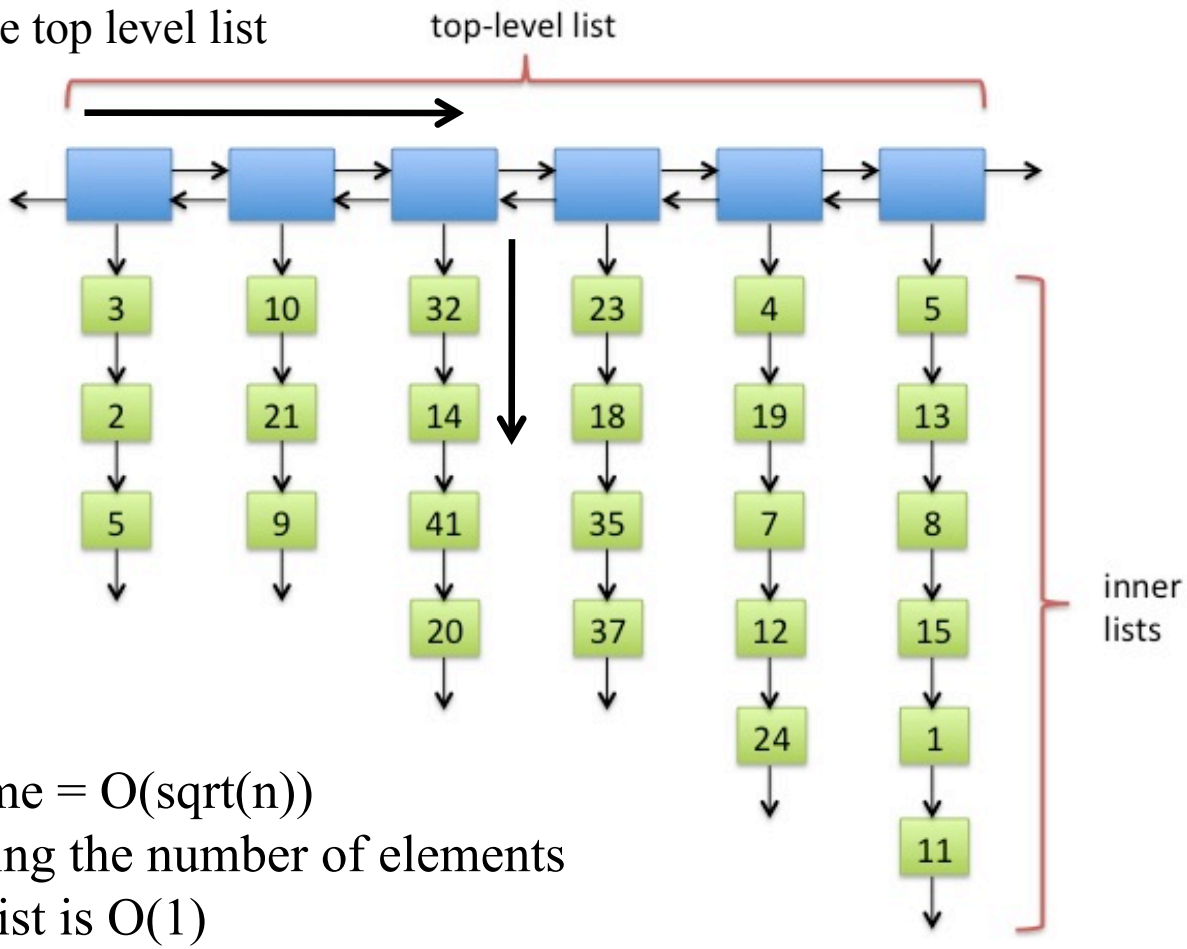
  the array index last item that was enqueued

  Initialized to -1

Project 1 (squarelist)

LinkedList of LinkedList

Finding the top level list



Traversal time = O(sqrt(n))
Only if finding the number of elements
In an inner list is O(1)

Project 1 (squarelist)

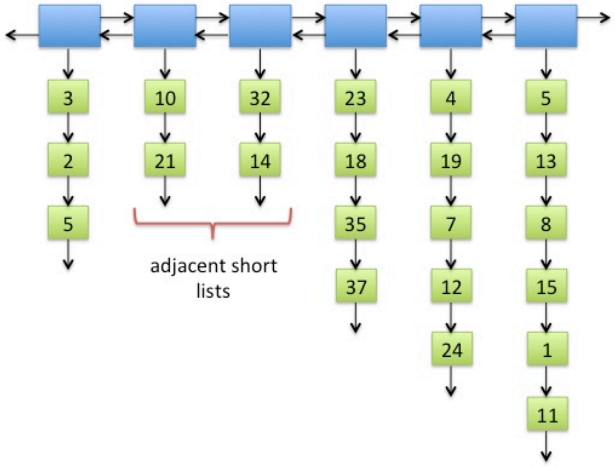
Condition 1:

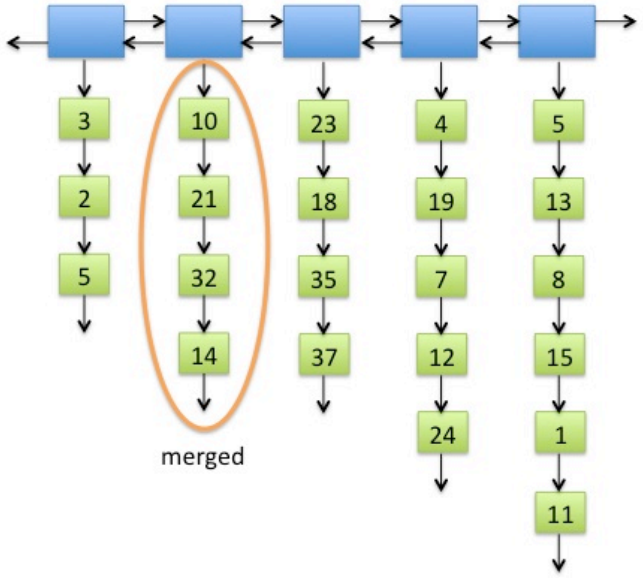
Every inner list has $\leq 2\sqrt{n}$ items.

Condition 2:

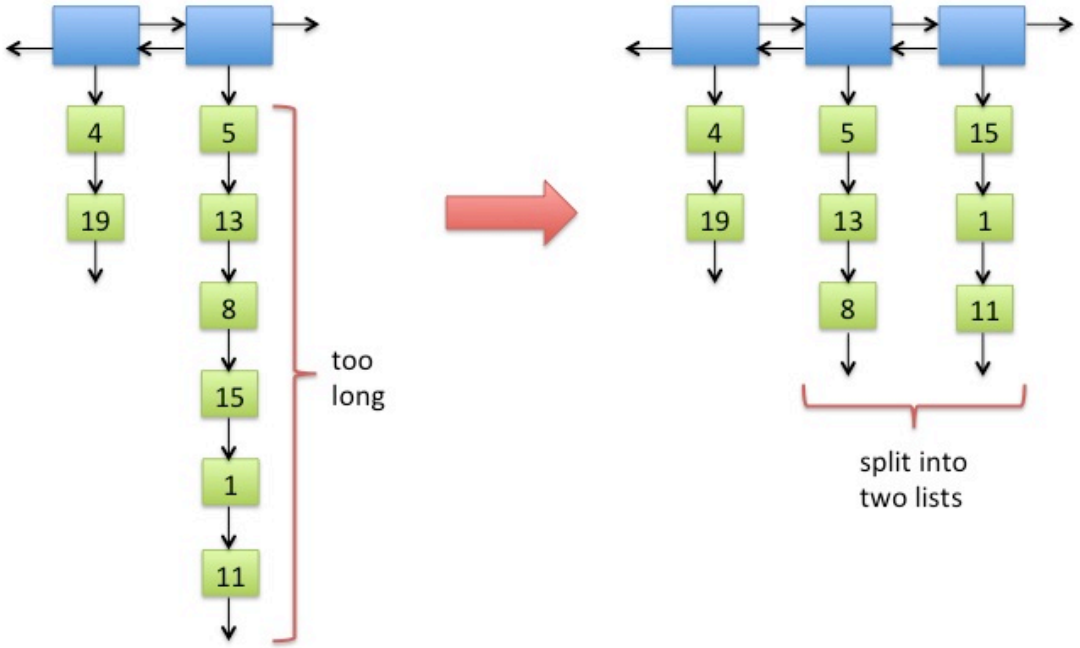
There are no adjacent short inner lists, where *short* is defined as having $\leq \sqrt{n}/2$ items.

Project 1 (squarelist) – Merging list





Project 1 (squarelist) – split long list



Project 1 (squarelist) – consolidation

Consolidate:

1. Traverse the top-level list.
2. Whenever an empty inner list is encountered, remove that inner list.
3. Whenever two adjacent short inner lists are encountered, merge them into a single inner list. (See Figures 2 and 3.)
4. Whenever an inner list is found to have more than 2 $\sqrt{n}$ items, split them into two lists of equal length. (See Figure 4.)

Discuss the implementation