Some Examples

Lists, Stacks, Queues
First, a Summary:

**List ADT**
- *set*: list of things
- *operators*: print, empty, insert, delete, find, findkth, pop, push, top, enqueue, dequeue

**Linked List ADT**
- *usually* restricted to: print, empty, insert, delete, find, findkth

**Stack ADT**
- *usually* restricted to: pop, push, top, empty

**Queue ADT**
- *usually* restricted to: enqueue, dequeue, print, empty

NOT an exhaustive list of the possible operators
Meant only to illustrate that these are all related ADTs
Example 1: Train Sorting (Which ADT Do I Use?)

- In a railroad yard we have a train with 100 cars that must be sorted into their order of delivery. The tracks have the following arrangement. Which of the linked list, queue or stack should I use?

  dead end (called a spur track)
Example 1 (part b)

• For starters, a stack. (We are constrained by the track arrangement.)

Push the cars onto the spur. Pop them out when needed.
Example 1 (part c)

- But what about the train itself?
- A **linked list**! (or doubly linked list)
Example 1 (part d)

- So we can store a car on the spur track
  1. unlink from list (train)
  2. “push” onto the stack (spur)
Example 1 (part e)

- Now can pop into another spot on the train.
- Or can push another car, etc.

Is there a link here or not? (Depends on implementation!)
Example 2: Recursive method calls – stacks

```c
int factorial(int n)
{
    if(n==0 || n==1)
        return 1;
    else
        return n*factorial(n-1);
}
```

Trace…
1. Call factorial(3)
2. return 3*factorial(2) after a call to (and return from) factorial(2).
3. call factorial(2)
4. return 2*factorial(1) after a call to (and return from) factorial(1).
5. call factorial(1)
6. return 1
Example 2 (part b)

- How does code remember where to return after a function call?
- Just push onto a stack each time make another method call.
- Pop off of the stack each time done with a method call.

<table>
<thead>
<tr>
<th>Method Call</th>
<th>Stack Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>First</td>
<td>return 3*factorial(2)</td>
</tr>
<tr>
<td>Second</td>
<td>return 2*factorial(1)</td>
</tr>
<tr>
<td>Third</td>
<td>return 1</td>
</tr>
</tbody>
</table>

stack
Example 3: Linked list

- My old TV would only let me surf up and down from one channel to the next. What kind of data structure do you think they used for the TV software? [doubly linked list]

- My friend’s TV does the same thing, but is fast surfing up the channels and really slow when surfing down the channels. What kind of data structure do you think the TV uses? [linked list]

- I upgraded to a new TV that lets me quickly surf up and down and jump around to any channel. What kind of data structure do you think they used? [array list: fastest for findKth, don’t need insert and delete, fast for up and down.]
Example 4: Train Station (queue)

• Now our train is delivering passengers to a station. The station platform is too small and only one car can be unloaded at a time.

• Use linked list, stack, queue?

• Probably **queue**, but any of them work. Just take the first car, then the second, third, etc.
Example 5: Sets
(Which ADT Do I Use?)

• S’pose have two sets of numbers, not sorted.
• Find the intersection of these numbers.
  • 67,-32, 4, 5, 7, 22, -35
  • 45, 7, 4, 27, -35, -7, -9, 10, 99

• Use a Linked List? Array List? Stack? Queue? None of the above? Why????
• What if I want the answer in a sorted list?
Example 5: Answers

- If just printing out the numbers then an array list should work fine.

- But I’d suggest a linked list if storing the values in a new sorted list – lots of inserts will be too costly in anything else!

- BTW, for practice… Just how costly do you think your algorithm for “intersection” would be?