1 Assorted ADTs

Below are some sketches of ADTs (not real Java code). It's not important to understand the details of how these work right now; just try to understand how each one can be used conceptually.

```
List {
    insert(item, position);      // inserts item into the list at the position
    get(position);              // returns the item in the list at the position
    size();                        // returns the number of items in the list
}

Set {
    add(item);                 // puts item in the set. Does not add duplicates
    contains(item);            // returns whether or not the item is in the set
    items();            // returns a List of all items in some arbitrary order
}

Stack {
    push(item);                                    // puts item onto the stack
    pop();                    // removes and returns the most recently put item
    isEmpty();                              // returns whether the stack is empty
}

Queue {
    enqueue(item);                                 // puts item into the queue
    dequeue();               // removes and returns the least recently put item
    isEmpty();                              // returns whether the queue is empty
}

PriorityQueue {
    enqueue(item, priority);       // puts item into the queue with a priority
    dequeue();            // removes and returns the item with highest priority
    peek();      // returns but does not remove the item with highest priority
}

Map {                                          // like a dictionary from python
    put(key, value);      /* puts key into the map and associates it with the
       given value. If key is already in the map, replaces its existing
       value with the given value */
    get(key);                                 // returns value associated with key
    keys();              // returns a List of all keys in some arbitrary order
}
```

2 Solving Problems with ADTs

Consider the problems below. Which of the ADTs given in the previous section might you use to solve each problem? Although in principle any of the ADTs might be used to solve any of the problems, think about which ones will make code implementation easier or more efficient.

1. Given a sequence of parentheses, determine whether or not they are nested correctly. e.g. `(()(()))` is nested correctly, but `(()(` is not.

2. Given a text file that contains the name and country of birth of each Berkeley student, count the number of different countries that are represented.

3. Given a bunch of strings, partition the strings into groups based on which ones are anagrams of each other. e.g. given {"cat", "love", "act", "bat", "tab", "tac"}, you should return {{"cat", "act", "tac"}, {"love"}, and {"bat", "tab"}}. Hint: assume you can write a helper method that takes in a string and returns the same string but with its characters sorted.

3 More Complicated ADTs

The first page introduced you to some basic ADTs; you can find implementations of these in Java's standard library. But if we want something more complicated, we'll have to build it ourselves.

1. Suppose we want an ADT called `BiDividerMap` with the following functionality (assume `K` is something `Comparable`):

```
put(K, V);                              // put a key, value pair
getByKey(K);                // get the value corresponding to a key
getByValue(V);              // get the key corresponding to a value
numLessThan(K);    // return number of keys in the map less than K
```

   Describe how you could implement this ADT building off the ADTs given on the first page. Do not write code. Then, suppose you decide you want `numLessThan(K)` to run really fast. Can you think of any ways to improve your implementation to account for this?

2. Next, Suppose we would like to invent a new ADT called `MedianFinder` which supports the following operations:

```
add(int x);                  // add the integer into the collection
getMedian();       // returns the median integer in the collection
```

   Again, describe how you could implement this ADT building off of the ADTs from the first page.

   Auxiliary for Adepts: Ensure that `add(int x)` and `getMedian()` each use a number of method calls independent of the items in the `MedianFinder` object.

4 ADTing in Circles

You want to solve a problem using a queue, but unfortunately, you only have access to a class that is a stack. You decide to implement the queue ADT just using stacks. Complete the following class, assuming that you have access to a class called `Stack` which implements the stack ADT. Hint: Consider using two stacks.

```
public class SQueue{
    // add any instance variables you like



    public SQueue(){
        // add any code to the constructor you like




    }

    public void enqueue(int item){
        // your code here









    }

    public int dequeue(){
        // your code here









    }
}
```

Auxiliary for Adepts: Can you do it with only one stack? **Especially Extra:** Are you really getting away with using only one stack?