Part II (8 points)

Your main assignment is to implement a Set ADT in Set.java. Your Set class must use a list to store the elements of the set. Your Sets should behave like mathematical sets, which means they should not contain duplicate items. To make set union and intersection operations run quickly, your Sets will contain only Comparable elements, and you will keep them sorted in order from least to greatest element. (You will want to review the Comparable interface on the Java API Web page.)

You will need to decide on fields and implement the following methods.

- `public Set()` // Constructs an empty Set.
- `public int cardinality()` // Number of elements in this Set.
- `public void insert(Comparable c)` // Insert c into this Set.
- `public void union(Set s)` // Assign this = (this union s).
- `public void intersect(Set s)` // Assign this = (this intersect s).
- `public String toString()` // Express this Set as a String.

Unlike in previous assignments, each method comes with prescribed time bounds that you must meet when your Set uses DLLists (but not when it uses SLLists). For example, `union()` and `intersect()` must run in time proportional to `this.cardinality() + s.cardinality()`. This means you don’t have time to make a pass through "this" list for each element of s; that would take time proportional to `this.cardinality() * s.cardinality()`. You must take advantage of the fact that Sets are sorted to achieve this time bound. This time bound is one reason why Sets may not store duplicate items in their Lists.

On the other hand, `insert()` need not run in constant time. Since each Set uses a sorted representation, `insert()` may need time proportional to the cardinality of the Set to find the right place to insert a new element, and to ensure that the new element doesn’t duplicate an old one.

Another constraint is that `union()` and `intersect()` may NOT change the Set s. Furthermore, `intersect()` may not create any new ListNodes (it only needs to remove ListNodes from "this" List), and `union()` should reuse all the ListNodes in the Set "this", creating new nodes only for elements of s that "this" List lacks. We will deduct points for failing to meet the time bounds or failing to obey these constraints.

Be sure to declare variables of static type List and ListNode in Set.java, not variables of type DLList, DLListNode, SLList, or SLListNode. Set.java should be able to switch between using DLLists and using SLLists by changing one constructor call in the Set() constructor. (In fact, you can use SLList to help you debug Set if you have trouble getting DLList working. But be sure to use a DLList in your final submission unless you can’t get it working.)

Do not modify List.java, ListNode.java, SLListNode.java, or DLListNode.java. Do not modify the prototypes in Set.java, DLList.java, or DLListNode.java.

Afterthought (for your own introspection only)

If you use SLLists instead of DLLists, do your `union()` and `intersect()` methods still run within the time bounds? If not, how easy would it be to fix them so that they do?
Submitting your solution

Change (cd) to your hw5 directory, which should contain the list directory. The list directory should contain DList.java and DListNode.java. You're not allowed to change the other files, so you can’t submit them. You shouldn’t need any other classes, but you can submit them if you want.

Make sure that your code compiles and runs on the _lab_ machines. Then, from your hw5 directory, type "submit hw5". (Note that "submit" will not work if you are inside the list directory!) After submitting, if you realize your solution is flawed, you may fix it and submit again. You may submit as often as you like. Only the last version you submit before the deadline will be graded.