CS 6301.002. Implementation of advanced data structures and algorithms Spring 2016
Short Project 1
Due: 1:00 PM, Thu, Jan 28.
Group 10

Problems:
A - SetOperations.java
B - ListAddSubtract.java
C - SortableList.java
D - BinTree.java and StackRecurse.java
E - SinglyLinkedList.java
F - SinglyLinkedListReversal.java

A. Given two linked lists implementing sorted sets, write functions for union, intersection, and set difference of the sets.
Implementation for intersection: Created 2 iterators to iterate over the elements for both the lists. Increment the iterator for the list whose element is smaller. If you encounter two elements that are same, then add them to the output list.
Testing
Input:
1 2 6 9
1 3 6 7 8 10
Output:
1 6
Runtime: O(list1.size + list2.size)

Implementation for union: Created 2 iterators to iterate over the elements for both the lists. Add the elements that is smaller to the output list and increment the iterator for the list to which the element belonged. If you encounter two elements that are same, add it to the list and increment both the iterators.
Testing
Input:
1 2 6 9
1 3 6 7 8 10
Output:
1 2 3 6 7 8 9 10
Runtime: O(list1.size + list2.size)

Implementation for difference: Created 2 iterators to iterate over the elements for both the lists. Add element from the first list if it is smaller. If element from the second list is smaller, just increment the iterator without adding it to the output list. If you encounter two elements that are same increment iterators for both the lists.
Testing:
Input:
1 2 6 9
1 3 6 7 8 10
Output:
2 9
Runtime: O(list1.size + list2.size)

b. Implement numbers as List and write basic operation of add and subtract
Implementation: As discussed in class, created 2 iterators to fetch the next element in both the lists.
Sum is b*(sum/b) + sum % b, where sum/b is the carry-over and sum % b is added to output List.

Runtime: O(max(list1.size, list2.size))
Space: O(1)
Testing:
Input:
1 8 4 3 3 7 4 8 8 7
4 5 5 7 3 4 7
Sum output:
5 3 0 1 7 1 2 9 8 7
Subtract output:
7 2 9 5 9 2 7 7 8 7

C. Implementation of merge sort
Mergesort works even better on linked lists than it does on arrays. It avoids the need for the auxiliary space, and becomes a simple, reliably O(N log N) sorting algorithm.
Runtime: O(nlogn)
Space: O(logn)
Testing:
Input: 13 3 11 6 8 11 0 11 10 3 9 8 13 3 11
Output: 0 3 3 3 6 8 8 9 10 11 11 11 11 13 13

d. Implement a recursive algorithm without recursion, by using a stack to simulate recursion. You may work on any recursive algorithm that has multiple recursive calls such as Merge Sort, Binary tree traversals, Quick sort, or, Linear-time median.
Implementation:
Solution implements the pre-order binary tree traversal on a given tree of fixed height. The algorithm is:
stack

- While stack is not empty:
    - **Read** the top of the stack.
    - Is the element marked True?
        - If so, **Pop** the element and Continue
    - Else:
        - **Print** the element
        - **Mark** it as True
        - **Add** its children to the stack (Right child first)

Test Input:
```
                    1
            2--------/--------5
3-------/--4             6--/------7
```

Test Output: 1234567

e. Implement unzip (invariant)
state s indicates the state of finite state machine.
tails[i] is the tail of elements of chain in state i
headers[i] is the head of elements of chain in state i(i = 0,1,2,...k)
loop runs through entire list using c as next element,
when c is added to state i, tails[0] points to new element and state is changed to state + 1 or 0
Runtime: O(n)
Space: O(1)
Testing:

Input: 1 2 3 4 5 6 7 8 9 10 b=4, n=10
Output: 1 4 7 10 2 5 8 3 6 9

f. Write recursive and nonrecursive functions for the following tasks:
(i) reverse the order of elements of the SinglyLinkedList class
(ii) print the elements of the SinglyLinkedList class, in reverse order.
Write the code and annotate it with proper loop invariants.

Part A:-

Without using recursive function

```
Iterate trough the linked list until the next element is not null.
  while next element not null,
      change next to prev,
      prev to current and
      current to next.
```

Test Input: 1 2 3 4 5 6 7 8 9 10
Test Output: 10 9 8 7 6 5 4 3 2 1

Part B:-

With using recursive function

- Iterate trough the linked list until the next element is not null.
  Divide the list in two parts - first node and rest of the linked
      next= current.next;

- Call reverse for the rest of the linked list.
  Reverse (next1, current);

Test Input: 10 9 8 7 6 5 4 3 2 1
Test Output: 1 2 3 4 5 6 7 8 9 10