**COMS W3134 Fall 2014  
Homework 2  
Due: 5:00pm on Wednesday, October 8, 2014**

**Written**

For the written section of this assignment, you type up your answers and submit a computer document to us. You can submit MS Word doc files, pdf files, or txt files.

1. Weiss, Exercise 3.1

public static void printLots(Collection<Integer> L, Collection<Integer> P)

{

Integer position = 0;

Integer element = 0;

Iterator<Integer> iterator = P.iterator();

while(iterator.hasNext())

{

position = (Integer)iterator.next();

Iterator<Integer> iterator2 = L.iterator();

for(int i = 0; i <= position; i++)

{

if (iterator2.hasNext())

element = iterator2.next();

}

System.out.println(element);

}

}

1. Weiss, Exercise 3.2

Swap two adjacent elements by adjusting only the links (and not the data) using:

a. Singly linked lists.

Node<AnyType> swap(Node<AnyType> first) {

if(first == null || first.next == null)

return first;

Node<AnyType> second = first.next;

Node<AnyType> third = second.next;

first.next = third;

second.next = first;

return second;

}

b. Doubly linked lists.

public void swap(Node node)

{

node = current.prev;

node.next = current.next;

current.next = node.next.next;

node.next.next = current;

node.next.prev = node;

current.prev = node.next;

current.next.prev = current;

}

1. Weiss, Exercise 3.8

The following routine removes the first half of the list passed as a parameter:

public static void removeFirstHalf( List<?> lst )

{

int theSize = lst.size( ) / 2;

for( int i = 0; i < theSize; i++ )

lst.remove( 0 );

}

1. Why is theSize saved prior to entering the for loop?
   1. theSize is saved prior to entering the loop in order to ensure that only half of the list is removed. If theSize is inside of the for loop, then the size of the array will continually be cut in half for each iteration, which will then change the output entirely.
2. What is the running time of removeFirstHalf if lst is an ArrayList?
   1. O(N2) because the remove method for a ArrayList costs O(N) and multiplying that with the cost of the for loop O(N) gives us a quadratic running time.
3. What is the running time of removeFirstHalf if lst is a LinkedList?
   1. O(N2) because the remove method for an ArrayList costs O(N) and multiplying that with the cost of the for loop O(N) gives us a quadratic running time.
4. Does using an iterator make removeHalf faster for either type of List?
   1. An iterator would make the LinkedList faster and would change its running time to O(N), because the running time for the remove method for a LinkedList iterator is O(1). An iterator for the ArrayList would not improve the running time and would stay at O(N), because the running time for the remove method for an ArrayList iterator is O(N).
5. Weiss, Exercise 3.24

Write routines to implement two stacks using only one array. Your stack routines should not declare an overflow unless every slot in the array is used.

In order to implement two stacks using only one array I created two stacks that increase in opposite directions, so one goes forward and one goes backward. This way the user can insert elements on either side of the stack in whatever ratio as long as the total # of elements pushed does not exceed the size of the array.

import java.util.EmptyStackException;

public class TwoStack<E>

{

private final int DEFAULT\_SIZE = 100;

private int top, top2;

private int size;

private static int[] stack;

public TwoStack()

{

top = 0;

top2 = DEFAULT\_SIZE;

stack = new int[DEFAULT\_SIZE];

}

public TwoStack(int n)

{

size = n;

top = -1;

top2 = size;

stack = new int[size];

}

public void push(int element)

{

if(top < (top2 - 1)) // at least one empty space in array

{

top++;

stack[top] = element;

}

else

System.out.println("Stack Overflow.");

}

public void push2(int element)

{

if(top < (top2 - 1)) // at least one empty space in array

{

top2--;

stack[top2] = element;

}

else

System.out.println("Stack Overflow.");

}

public int pop() throws EmptyStackException

{

if(isEmpty())

{

System.out.println("No elements in this stack");

throw new EmptyStackException();

}

else

{

int result = stack[top];

top--;

return result;

}

}

public int pop2() throws EmptyStackException

{

if(isEmpty())

{

System.out.println("No elements in this stack");

throw new EmptyStackException();

}

else

{

int result = stack[top2];

top2++;

return result;

}

}

public int peek() throws EmptyStackException

{

if(isEmpty())

{

System.out.println("No elements in this stack");

throw new EmptyStackException();

}

else

{

int topElement = stack[top];

return topElement;

}

}

public int peek2() throws EmptyStackException

{

if(isEmpty())

{

System.out.println("No elements in this stack");

throw new EmptyStackException();

}

else

{

int topElement = stack[top2];

return topElement;

}

}

public boolean isEmpty()

{

if(top == -1)

return(true);

if(top2 == stack.length)

return(true);

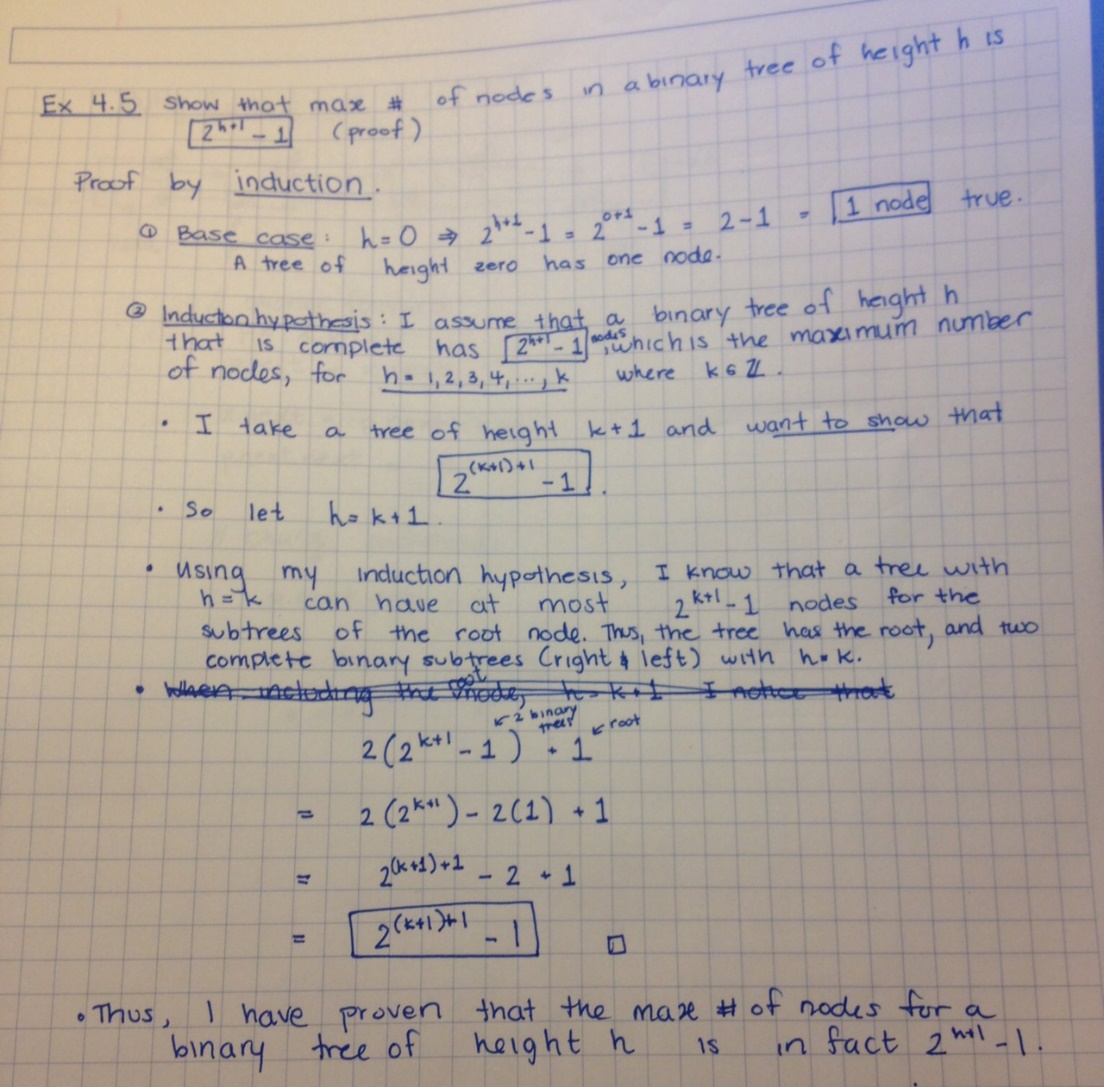
return false;

}

}

1. Weiss, Exercise 4.5 (this is asking for a proof)

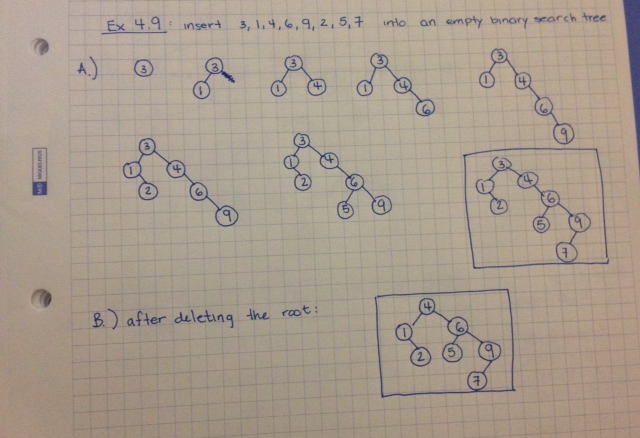
Show that the maximum number of nodes in a binary tree of height h is (2h+1 – 1).



1. Weiss, Exercise 4.9 (using a full deletion)

a. Show the result of inserting 3, 1, 4, 6, 9, 2, 5, 7 into an initially empty binary search tree.

b. Show the result of deleting the root.

****