Homework HW 0 (15 pts)

1. [5 pts]: Prove by mathematical induction that for all positive integers *n*,

First, show the base case for this induction (2 pts).

Then state and prove the inductive step (3 pts).

***Solution:***

The base case is n=1. 1\*0 = 0 on the left, and on the right

Inductive step.

Suppose

Then

=

=

=

This proves the induction.

1. [5 pts] Describe how you would implement a stack using two queues. That is, for some object x of the appropriate type, describe how you would implement the standard stack methods push(x), pop(), and isEmpty(). You are given two queues, each of which has operations enqueue(x), dequeue(), and isEmpty().

You may do this with pseudocode or you may do it with an English-language description.

***Solution:***

I’ll do this using the Queue interface methods, for an element x of type E, shown below. I also use a method I call “switchQ1Q2()” that switches the names of queues q1 and q2. I don’t bother to show the pseudocode for this.

boolean add(E x)

E element()

boolean offer( E x )

E peek()

E poll()

E remove()

*// Pseudocode starts here:*

Queue q1, q2;

q1 = new Queue();

q2 = new Queue();

*// push() is just an enqueue. I do all the machinations needed to make pop() work separately.*

E push( E x ) {

q1.add( x );

}

*// if q1 is empty, so is the stack.*

boolean isEmpty() {  
 if ( q1.peek() == null ) {

return true;

}

else {

return false;

}

}

*// To pop the stack, I want to get the last element in the queue on q1. So I dequeue everything on queue 1 and move*

*// it to queue 2. The last item in q1 is the element I want to pop. Then I switch the sense of queues q1 and q2.*

E pop() {

if q1.isEmpty() {

return null;

}

do {

E temp = remove();

while ( !q1.isEmpty() );

switchQ1Q2();

return temp;

}

1. [5 pts] How many dots will the following method print out as a function of n? I’m happy with a big-O estimate of this number. Note that this is written in pseudocode, not any particular programming language (although it follows C/Java-like syntax). Also note that the most important part is to explain your reasoning.

method3( int n ) {

for ( int j = 1; j <= 2^n; j++ ) {

for ( int k = 1; k <= j; k \*= 2 ) {

print(“.”);

}

}

}

***Solution:***

The outer loop runs O(2n) times. For each time through the outer loop, the inner loop runs O(lg j) times. So the total number of iterations (dots) is

Since the value of k doubles every time, the upper limit of the second (inner) summation is actually ⎣lg j⎦, but since we’re interested in big-O estimates, we can ignore the floor function. So

I get the last equality by looking up that