**NAME:**

**COMP231 – Data Structures and Problem Solving**

**Spring 2018**

**Midterm Exam #1**

**March 8, 2018**

1. Consider the following method:

boolean isIdentity(int[][] x) {

**for**(**int** i = 0; i < x.length; ++i){

**for**(**int** j = 0; j < x[0].length; ++j){

**if**(i == j){

**if**(x[i][j] != 1){

**return** **false**;

}

}**else**{

**if**(x[i][j] != 0){

**return** **false**;

}

}

}

}

**return** **true**;

}

[2] a. What is the problem size, n, for this algorithm?

The length of one side of the array. (Note that this was a bit a of bad question since I didn’t specify the array was square.)

[4] b. Describe the input that will evoke the best-case running time for this algorithm.

The best case will be if x[0][0] !=1

[4] c. Describe the input that will evoke the worst-case running time for this

algorithm.

The worst case is when x[i][j] == 1 for i==j and x[i][j] == 0 for i!=j

[5] d. Counting array accesses as the basic operation, give an equation T(n) for the worst-case operation count for the method.

T(n) = 2n^2 + n

2. Consider the following method:

public static boolean functionX(int[] x) {

**int** count = 0;

**for**(**int** i = 0; i < x.length-1; ++i){

**if**( x[i] != x[i+1]){

count++;

}

**else**{

**break**;

}

}

return count;

}

[1] a. What is the problem size, n, for this algorithm?

The length of the array x.

[4] b. Give the best Ω bound for the running time of the functionX method.

Omega(1)

[4] c. Give the best Ο bound for the running time of the functionX method.

O(n)

3. Consider the following recursive method:

public static int functionY (int n) {

**if**(n==0){

**return** 0;

}**else** **if**(n%2 == 1){

**return** 1 + functionY(n/2);

}**else**{

**return** functionY(n/2);

}

}

Assuming that arithmetic or relational operators (==, + , -, etc.) each require 1 basic operation and that an if statement, a method call and a return statement each require 3 basic operations, give a recurrence relation for the basic operation count for the above method.

[2] T(0) = 4

[4] T(n) = T(n/2) + 16

4. Consider the following recurrence relation giving the basic operation count for a recursive method:

T(n) = T(n-1) + n - 1

T(0) = 7

[3] a. Show the first step in the use of the expansion technique for solving this

recurrence.

T(n) = T(n-2) + n-1 + n – 1= T(n-2) +2n -2

[4] b. Continue using the technique of expansion to give a closed form expression for

T(n).

T(n) = T(n-3) + n-1 + 2n -2 = T(n-3)+3n-3

T(n) = T(n-k) + kn -k

T(n) = T(0) + n^2 +n = 7 + n^2 + n

[4] 5. Consider the following expression showing a basic operation count:

T(n) = 2 + 18 + 32 + 50 + … + 2(n-2)2 + 2(n-1)2 + 2n2

Express this operation count using summation notation. Your answer should contain

a summation. Do not reduce to a closed-form expression.

Sum \_{i=1} ^{n} 2i^2

[4] 6. Consider the following expression showing a basic operation count:

T(n) =

Give a closed form expression for T(n).

5((n-1)\*n/2 -1)-2\*(n-1-2)

[ 7] 7. Consider the following basic operation count for an algorithm:

T(n) = 5n2+ 14n lg n + 25

Use the definition of Ο to show that T(n) is in Ο(n2)

Lim \_{n->infty} \frac{5n^2+14nlgn+25}{n^2} =

Lim \_{n->infty} 5+14lgn/n+25/n^2 =

5

5 is constant or 0, so T(n) \in O(n^2)

[8] 8. Assuming that A and B are two algorithms for solving the same problem, circle T or

F to indicate whether each of the following statements is true or false.

1. T / **F** If A is in Ω(n) and B is in Ω(nlg n) then A will always run faster than B
2. T / **F** If A is in Ω(n) and B is in O(lg n), then A will always run slower than B.
3. **T** / F If A is in Ω(n) and B is in O(lg n), then there must be an instance where A runs slower than B.
4. T / **F** If A is in Ο(n2) and B is in Ο(n lg n) then there must be an instance of the problem for which B will run faster than A.
5. T / **F** If A is in Ο(n) and B is in Ω(lg n) then there must be an instance of the problem for which B will run faster than A.
6. **T** / F If A is in Θ(n) and B never runs more slowly than A, then B is in Ο(n).
7. **T** / F If A is in Ω(n) and also in Ο(n lg n) and B is in Θ(n2), then there must be an instance of the problem for which A will run faster than B.

9. The sumNoConsecutiveEvens method is to answer the following question. Given an array of integers, is it possible to choose some of the elements such that they sum to a given target value, with the following additional constraint: When an even value is chosen, the next value cannot also be even. For example:

[2, 5, 10, 4] with target 12 → false  
[2, 5, 10, 4] with target 14 → false  
[2, 5, 10, 4] with target 7 → true

[2, 5, 10, 4] with target 9 → true

[2, 5, 10, 4] with target 11 → true

[5] a. Complete the sumNoConsecutiveEvens method shown below assuming that the sumNoConsecutiveEvensHelper method defined in part b works.

public static boolean sumNoConsecutiveEvens (int[] vals, int targetSum) {

return sumNoConsecutiveEvensHelper(vals, targetSum, 0, 0, false);

}

[15] b. The sumNoConsecutiveEvensHelper method below solves a transformed version of the original problem. Complete the implementation of sumNoConsecutiveEvensHelper so that the sumNoConsecutiveEvens method from part a will answer the original question.

public static boolean sumNoConsecutiveEvensHelper (int[] vals, int targetSum,

int curSum,

int curIndex, boolean prevWasEven) {

if( curIndex == vals.length ){

return curSum == targetSum;

}

boolean leave = sumNoConsecutiveEvensHelper(vals, targetSum, curSum, curIndex+1, prevWasEven);

boolean take = false;

if( vals[curIndex]%2 ==1){

take = sumNoConsecutiveEvensHelper(vals, targetSum, curSum + vals[curIndex], curIndex+1, false);

}else if( !prevWasEven ){

take = sumNoConsecutiveEvensHelper(vals, targetSum, curSum + vals[curIndex], curIndex+1, true);

}

return take|| leave;

10. Recall our definition of DLLNode (shown below):

private class DLLNode {

public E element;

public DLLNode prev;

public DLLNode next;

public DLLNode(E element, DLLNode prev, DLLNode next) {

this.element = element;

this.prev = prev;

this.next = next;

}

}

Also recall that in our CS232DoublyLinkedList class the fields head and tail refer to dummy nodes at the start and end of the list, and that the field size holds the number of elements in the list.

[6] a. Give a snippet of code that will count the size of the list.

Some people wrote “return size;” which I ended up giving credit for.

int count = 0;

DLLNode cur = head;

While( cur.next!=tail){

cur = cur.next;

count++;

}

[7] b. Assuming that the variable succ refers to a DLLNode in a doubly linked list (not

including head or tail) and that insNode refers to a DLLNode, give a snippet of

code that will insert insNode into the list after the node referred to by succ.

insNode.next = succ.next;

insNode.prev = succ;

succ.next.prev = insNode;

succ.next = insNode;

[7] c. Given an input value v of type E, give a snippet of code that removes all nodes with value equal (.equals) to v.

DLLNode cur = head.next;

while(cur.next!=tail){

if(cur.value.equals(v)){

cur.prev.next = cur.next;

cur.next.prev = cur.prev;

}

cur = cur.next;

}

11. For the tree given above, answer the following questions (2pts each):

What type of node is Q? \_\_\_\_\_\_internal\_\_\_\_\_

What type of node is A? \_\_\_\_root\_\_\_\_\_\_\_

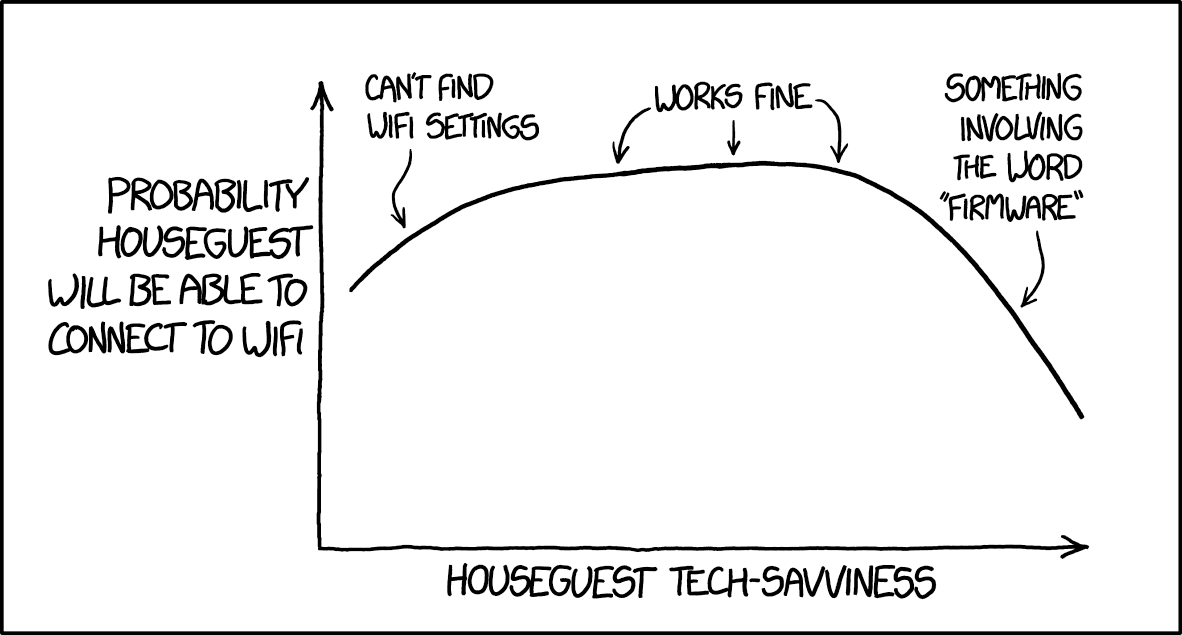
What is the left child of Y? \_\_\_\_\_\_\_\_Q\_\_\_\_

How many nodes does the subtree rooted at B have? \_\_\_\_6\_\_\_\_\_\_\_

What is the distance from O to A? \_\_\_\_\_\_\_3\_\_\_\_\_\_\_

What is the width of the tree? \_\_\_\_\_\_4\_\_\_\_\_\_\_\_

What is the in-order traversal of the tree? \_\_\_\_\_\_\_FDBZIEAQOY\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



Bonus: Guess your percentage score on this exam (out of 100, before any bonus is applied to your score)

Bonus: What does the function in #3 compute?

Bonus: What is the O running time of the function in #3? (Be specific about what you are using as your input size.) What would be the best and worst cases?

Bonus: What is the Θ running time of the function in #3 on the input 57?

Bonus: What does function in problem 2 compute?

Bonus: Write a recursive method that sets all odd index elements of a linked list to 0.