Sample Questions: Code Run Time Analysis

August 2015 Computer Science A Question 2 (Iterative Code Segment)
Consider the following segment of code, assuming that \( n \) has been previously declared and initialized to some positive value:

```c
int i, j, k;
for (i = 1; i <= n; i++){
    for(k =1; k <= i; k++){
        j = k;
        while(j > 0)
            j--;
    }
}
```

(a) (3 pts) Write a summation (3 nested sums) equal to the number of times the statement \( j--; \) executes, in terms of \( n \).

(b) (7 pts) Determine a closed form solution for the summation above in terms of \( n \).

December 2014 Computer Science A Question 2a (Iterative Code Segment)
Write a summation, but do NOT solve it, that represents the value of the variable \( \text{sum} \) at the end of the following code segment, in terms of the variable \( n \), entered by the user. (Note: your answer should have two summation signs in it and appropriate parentheses that clearly dictate the meaning of the expression you’ve written.)

```c
int i, j, n, sum = 0;
printf("Please enter a positive integer.\n");
scanf("%d", &n);

for (i=n; i<2*n; i++) {
    sum += i;
    for (j=1; j<=i; j++)
        sum += (j*j);
}
```
**August 2014 Computer Science A Question 2b (Iterative Code Segment)**

Determine the run time of the code segment shown below, in terms of n. Provide your answer as a Big-Theta bound.

```c
int n;
scanf("%d", &n);
int i, step = 1, total = 1;
for (i=0; i<n*n; i+= step) {
    total++;
    step += 2;
}
```

**December 2013 Computer Science A Question 2ab (Iterative Code Segment)**

(a) (3 pts) Write a summation that represents the number of times the statement `p++` is executed in the following function:

```c
int foo(int n)
{
    int i, j, p = 0;
    for (i = 1; i < n; i++)
        for (j = i; j <= i + 10; j++)
            p++;
    return p;
}
```

(b) (5 pts) Determine a simplified, closed-form solution for your summation from part (a), in terms of n. You MUST show your work.

**August 2012 Computer Science B Question 1a (Iterative Code Segment)**

(a) (4 pts) Determine, with proof, the run-time of the following function in terms of the formal parameters a and b:

```c
int f(int a, int b) {
    int i,j, sum = 0;
    for (i=0; i<a; i++) {
        j = b;
        while (j > 0) {
            j = j/2;
            sum++;
        }
    }
    return sum;
}
```
August 2015 Computer Science B Question 1 (Recursive Code Segment)
Consider the recursive function diminish shown below:

```java
double diminish(int m, int n)
{
    if (n == 0)
        return m;
    return 1.0/2*diminish(m,n-1)
}
```

(a) (3 pts) Let T(n) represent the run time of the function diminish. Write a recurrence relation that T(n) satisfies.

(b) (6 pts) Using the iteration method, determine a closed-form solution (Big-Oh bound) for T(n).

May 2014 Computer Science A Question 2 (Recursive Code Segment)
Write a recurrence relation that represents the runtime of the following function, then solve it (i.e., derive its closed form) using iterative substitution:

```java
int foo(int n)
{
    if (n == 0 || n == 1)
        return 18;
    else
        return foo(n-2) + foo(n-2);
}
```
Recurrence Relations to Solve

1) \( T(n) = 2T \left( \frac{n}{2} \right) + 1, T(1) - 1 \)

2) \( T(n) = T(n - 1) + n, T(1) = 1 \)

3) \( T(n) = T \left( \frac{n}{2} \right) + n, T(1) = 1 \)

4) \( T(n) = 2T \left( \frac{n}{2} \right) + n, T(1) = 1 \)

Solution to #1 using iteration technique

Original equation: \( T(n) = 2T \left( \frac{n}{2} \right) + 1 \)

Plugging in for \( \frac{n}{2} \), we get \( T \left( \frac{n}{2} \right) = 2T \left( \frac{n}{2^2} \right) + 1 = 2T \left( \frac{n}{4} \right) + 1 \)

Similarly, we find:

\[
T(n) = 2T \left( \frac{n}{2} \right) + 1 \\
= 2 \left( 2T \left( \frac{n}{4} \right) + 1 \right) + 1 \\
= 4T \left( \frac{n}{4} \right) + 2 + 1 \\
= 4T \left( \frac{n}{4} \right) + 3
\]

Repeat, plugging in \( T \left( \frac{n}{4} \right) \):

\[
= 4 \left( 2T \left( \frac{n}{8} \right) + 1 \right) + 3 \\
= 8T \left( \frac{n}{8} \right) + 4 + 3 \\
= 8T \left( \frac{n}{8} \right) + 7
\]

In general, after \( k \) steps, we get:

\[
T(n) = 2^k T \left( \frac{n}{2^k} \right) + (2^k - 1)
\]

If we let \( 2^k = n \) (so that \( k = \log_2 n \)), we get

\[
T(n) = nT \left( \frac{n}{n} \right) + (n - 1) = n(1) + (n - 1) = 2n - 1 = O(n)
\]

Yielding the Big-Oh bound of the recurrence relation.