**HW4**

**To Turn in: please submit the questions and your answers below them in a pdf file on canvas.**

**Perform a time-complexity (Big-O) analysis for each of the next three problems (problems 1, 2, and 3). For full credit you should be able to produce a logical justification for your answer (a growth rate function can help demonstrate this – but is NOT required – so at least show in general why the Big-O is what it is). Equations you may need: (1) 1 + 2 + 3 + 4 + …+ n = (1 + n) \* n / 2; (2) 1 + a + a2 + a3 + … + an = (an+1 – 1) / (a-1).**

**1. (40 Points)** **public static void** two(int n)  
 {  
 **if**(n > 0)  
 {  
 System.out.println("n: " +n);  
 two(n - 1);  
 two(n - 1);  
 }  
 **else** **if** (n < 0)  
 {  
 two(n + 1);  
 two(n + 1);  
 System.out.println(″n: ″ + n);  
 }  
 }

**2. (30 Points)**  
**public void** three(**int** n)

{  
 **int** i, j, k;

**for** (i = n/2; i > 0; i = i/2) // log n

**for** (j = 0; j < n; j++) // n

**for** (k = 0; k < n; k++) // n

System.out.println("i: " + i + " j: " + j+" k: " + k);

} // end three

**3. ( 30 points)**

**public** **static** **void** four(**int** n)

{

**if** (n > 1)

{

System.out.println(n);

four(n-1);  
 }

**for** (**int** i = 0; i < n; i++)

System.out.println(i);

}

Problem #1

* If and else if cannot be executed at the same time.
* Draw the recursive tree
* The number of levels in the family tree is “n levels”
* The number of calls @ the lowest level is 2n because after each call they all have two children meaning it doubles and it’ll look like 2\*2\*2\*2… = 2n
* For a negative number n, the algorithm will do the same amount of work as |n| abs(n)
* O(2n)

Problem #2

* The second line of code the first for loop is log n
* The second and third for loop are n (3rd and 4th line of code)
* Multiple them together to it’ll be n\*n\*log(n) = n2 \* log(n) one term
* It will be a 3D space

Problem #3

* Draw recursive tree
* The number of call it has is “n calls” because it keeps going until it reaches 1
* Number of operations in each call
* The for loop depends on the input n, but for each recursive call n is different.
* It will be 1 + 2 + 3 + 4+ … + n - 2 + n - 1 + n

(1+n) \* n = n2 + n

2 = 2 + 2

* n = 1, n – 1 = 2, n – 2 = 3
* O(n2)