**CSE 373 Summer 2017 Homework 2: Asymptotic Analysis**

Due Thursday, July 6th at 5:00pm

Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Students you worked with (but did not share solutions with!):

Student ID number: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**1) Big-O (3 points)**

**For each of the following, show that ƒ ∈ O(g) by finding values for *c* and *n0* such that the definition of big-O holds true, as we did with examples in lecture. Listing valid values for *c* and *n0* is sufficient (no further proof necessary).**

**a)**

**b)**

**c)**

**2) Runtime Analysis (6 points)**

**For each of the following program fragments, determine the asymptotic runtime in terms of n. Explain or show your work (an explanation of your intuition can be sufficient).**

**a)**

public void mysteryOne(int n) {

int y = 1;

for (int j = 0; j < n\*(n-4); j++) {

for (int i = 1; i < n; i \*= 2) {

y \*= y;

}

}

}

**b)**

public void mysteryTwo(int n) {

int x = 0;

for (int i = n/2; i < n; i++) {

if (i % 5 == 0) {

break;

} else {

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

x++;

x \*= 2;

}

}

}

}

**c)**

public void mysteryThree(int n) {

for (int i = n; i >= 0; i--) {  
 helper(i);

}

}

private void helper(int x) {

if (x > 0) {

helper(x – 3);

}

}

**3) More Asymptotic Analysis (3 points)**

**For each of the following, determine if** ƒ∈O(g), ƒ∈Ω(g), ƒ∈Θ(g), ƒ∈o(g), ƒ∈ω(g), **several of these (and which), or none of these. No proofs necessary, listing your answer is sufficient.**

a)

b)

c)

**4) Pseudocode and Recurrence Relations (9 points)**

**a) Write pseudocode for a function that calculates the largest difference between any two numbers in an array of positive integers with a runtime in Θ(*n2*).**

*For example, the largest difference between any two numbers in the following array would be 90.*

*a = [37, 8, 92, 4, 6, 20, 11, 2, 40]*

**b) Can this function be written with a runtime in Θ(n)? If yes, write pseudocode below. If no, why? What would have to be different about the input in order to do so?**

**c) Can this function be written with a runtime in Θ(1)?. If yes, write pseudocode below. If no, why? What would have to be different about the input in order to do so?**

**5) Recursion and Recurrence Relations (5 points)**

**Note:** For this set of problems, let *c* and *d* be constants and let the base case be *T(c) = d.*

1. **What is the recurrence relation for the following method? (The base case is given)**

public void recursiveMethod(int n) {

if (n == 1) {

return;

}

int sum = 0;

for (int i = 0; i < n; i++) {  
 sum += i;

}

if (sum % 2 = 0) {

System.out.println(“The sum is even”);

return recursiveMethod(n/2);

} else {

System.out.println(“The sum is odd”);

return recursiveMethod(n/2);

}

}

1. **Find the *tightest* Big-Oh bound for your recurrence relation from part (a) using the base case *T(c) = b****.* **Justify your answer.**(An example of the *tightest* Big-O bound: for f(n) = 5n, the *tightest* bound is O(n), not O(n2))

**6) Growth Rates (3 points)**

**Order the following functions from slowest to fastest in terms of asymptotic runtime. Be sure to show whether two functions have the same asymptotic runtime.**

Ex: 5n < 3n2 = n2

|  |  |
| --- | --- |
| * n72 * n2 log(n) * 2(n/2) * log(n) * n log(n2) * n6 * n log(log(n)) * n log2(n) | * n * n2 * n log(n) * 2n * log2(n) * 2/n * log(log(n) * 2(1/2) |

**7) Proof by Induction (8 points)**

**Our friend the 3-Eyed Alien doesn’t believe that**  **as approaches infinity. “See,” they say, “for**  = 3 **the opposite is true!” Use induction to prove to our friend that after a certain point,**  **is indeed true.**

*Hint: It may be useful to use the fact that*  *for*  *the base case, along with some clever addition and subtraction after multiplying out the polynomial.*

***Be sure to clearly mark each step, and where you use the inductive hypothesis.***

**8) Amortized Analysis (5 points)**

**Note:** For all parts of this problem, when we ask for Big-O we’re asking for the *tightest* Big-O bound.

(For example, the *tightest* Big-O bound for f(n) = 5n is O(n), not O(n2))

1. **Imagine an array-based implementation of a stack that, instead of doubling its array size, it increases its array size by the fixed amount of 1000 every time it’s full. For this stack, what is the Big-O amortized cost (i.e. average asymptotic running time) to push() times? Justify your answer.**

1. **Based on their amortized costs for push(), which version of an array-stack would you use: one that doubles its array size or one that increases its array size by a fixed amount (i.e. 1000) whenever it’s full, and why?**
2. **Imagine an array-based implementation of a stack that doubles its array size when the array is full, and halves the array size when it’s ¾ empty. What is the Big-O amortized cost of pop()if it’s the th operation done on an initially empty stack? Justify your answer.**