Name: Breona Pizzuta Date: 9/23/24

I pledge my honor that I have abided by the Stevens Honor System.

Point values are assigned for each question. Points earned: \_\_\_\_ / 100, = \_\_\_\_ %

1. Find a tight upper bound for . Write your answer here: (4 points)

Prove your answer by giving values for the constants and . Choose the smallest integer value possible for . (4 points)

and

1. Find an asymptotically tight bound for . Write your answer here: (4 points)

Prove your answer by giving values for the constants , , and . Choose the tightest integer values possible for and . (6 points)

We need to show that

For the lower bound we can say = 2 because when

For the upper bound we can say = 3 because when 1

Therefore: ; ;

1. Is Circle your answer: yes / no. (2 points)

If yes, prove your answer by giving values for the constants and . Choose the smallest integer value possible for . If no, derive a contradiction. (4 points)

Or to explain:

So let's assume that there exists positive constants c and n such that₀  
0 ≤ cn² ≤ 3n - 4 ( ∀n ≥ n₀ )  
We also have 3n-4 ≤ 3n( n ≥ 1)

So by combining the two we must have:  
cn² ≤ 3n ( ∀n ≥ max(n₀ , 1))   
cn ≤ 3 ( ∀n ≥ max(n₀ , 1))  
n ≤ 3/c ( ∀n ≥ max(n₀ , 1))  
Since n can grow to infinity, it is impossible to find a positive constant c such that n is bounded above by the constant 3/c. Therefore the c we need to find does not exist. Therefore

1. Write the following asymptotic efficiency classes in **increasing** order of magnitude.

, , , , , , , , (2 points each)

,,,, , , , ,,

1. Determine the largest size *n* of a problem that can be solved in time *t*, assuming that the algorithm takes *f(n)* milliseconds. Write your answer for *n* as an integer. (2 points each)
2. *f(n)* = , *t* = 1 second 1000
3. *f(n)* = , *t* = 1 hour 204094
4. *f(n)* = , *t* = 1 hour 1897
5. *f(n)* = , *t* = 1 day 442
6. *f(n)* = , *t* = 1 minute 8
7. Suppose we are comparing two sorting algorithms and that for all inputs of size the first algorithm runs in seconds, while the second algorithm runs in seconds. For which integer values of does the first algorithm beat the second algorithm?

The first algorithm beat the second algorithm when n= [2,6] (4 points)

Explain in detail how you got your answer or paste code that solves the problem (2 point):

When plotting the two algorithms on a graph, we observe that in the range n=2 to n=6, the curve for the first algorithm () lies below the curve for the second algorithm (). This indicates that the first algorithm is faster for these input sizes. The intersection point at n=6 marks the point where both algorithms take the same time. Beyond n=6, the second algorithm becomes faster as the growth of the first algorithm begins to dominate. This can be shown in the provided graph.

A graph on a graph paper

Description automatically generated

Give the complexity of the following methods. Choose the most appropriate notation from among , , and . (8 points each)

**int** function1(**int** n) {

**int** count = 0;

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

**for** (**int** j = 1; j <= n; j \*= 2) { //log n times

count++;

}

}

**return** count;

}

Answer: θ(n n)

**int** function2(**int** n) {

**int** count = 0;

**for** (**int** i = 1; i \* i \* i <= n; i++) { //cubic iteration n^1/3

count++;

}

**return** count;

}

Answer: θ()

**int** function3(**int** n) {

**int** count = 0;

**for** (**int** i = 1; i <= n; i++) { // n times

**for** (**int** j = 1; j <= n; j++) { // n times

**for** (**int** k = 1; k <= n; k++) { // n times

count++;

}

}

}

**return** count;

}

Answer: θ()

**int** function4(**int** n) {

**int** count = 0;

**for** (**int** i = 1; i <= n; i++) { // n times

**for** (**int** j = 1; j <= n; j++) { // n times

count++;

**break**; //break !!!

}

}

**return** count;

}

Answer: θ (n)

**int** function5(**int** n) {

**int** count = 0;

**for** (**int** i = 1; i <= n; i++) { //n times

count++;

}

**for** (**int** j = 1; j <= n; j++) { //n times

count++;

}

**return** count;

}

Answer: θ(n)