0. Sumtimes Sum Sums Help (20 points)

Much of algorithm analysis involves working with summations. Finding closed-form solutions to these sums often involves simple algebraic manipulation. In this problem, you will gain experience working with common summations.

(a) [5 Points] Find a closed form for \(\sum_{i=0}^{\infty} \frac{1}{4^i}\). Explain how you got your answer.

(b) [5 Points] Find a closed form for \(\sum_{i=0}^{\infty} \frac{i}{4^i}\). Explain how you got your answer.

(c) [10 Points] Prove that \(\sum_{i=1}^{n} i^3 = \left(\sum_{i=1}^{n} i\right)^2\).

1. Exactly Exactly Exactly Exactly... (15 points)

Consider the following recurrence:

\[ T(n) = \begin{cases} 6 & \text{if } n = 1 \\ 1 + 2T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) & \text{if } n > 1 \end{cases} \]

Note that \(\lfloor n \rfloor\) is the floor function which rounds \(n\) down to the nearest integer.

(a) [5 Points] Determine \(T(n)\) for \(n = 1, 2, \ldots, 8\)

(b) [10 Points] Find an exact closed form for the recurrence. Show your work; don’t just give the final answer.

2. Hell-O! (23 points)

Use the formal definitions of Big-Oh, Big-Omega, and Big-Theta to prove or disprove each of the following statements. You should assume that the domain and co-domain of all functions in this question are the natural numbers.

(a) [5 Points] If we have an algorithm that runs in \(O(n)\) time and make some changes that cause it to run 10 times slower for all inputs, it will still run in \(O(n)\) time.

(b) [6 Points] Let \(M(n) = f(n)h(n)\). If \(f(n) \in O(g(n))\) and \(h(n) \in O(k(n))\), then \(M(n) \in O(g(n)k(n))\).

(c) [6 Points] \(2^{n^{1/3}} \in \Theta(2^n)\)

(d) [6 Points] \(2^{n^3} \in \Theta(2^n)\)
3. Is Your Program Running? Better Catch It! (20 points)

For each of the following, determine the asymptotic worst-case runtime in terms of \( n \).

(a) [5 Points]

```java
1 int sum = 0;
2 for (int i = 0; i < n; i++) {
3   for (int j = 0; j < n; j++) {
4     sum++;
5   }
6 }
```

(b) [5 Points]

```java
1 int sum = 0;
2 for (int i = 0; i < n; i++) {
3   for (int j = 0; j < n * n; j++) {
4     sum++;
5   }
6 }
```

(c) [5 Points]

```java
1 int sum = 0;
2 for (int i = 0; i < n; i++) {
3   for (int j = 0; j < i; j++) {
4     sum++;
5   }
6 }
```

(d) [5 Points]

```java
1 int sum = 0;
2 for (int i = 0; i < n; i++) {
3   for (int j = 0; j < i * i; j++) {
4     for (int k = 0; k < j; k++) {
5       sum++;
6     }
7   }
8 }
```

4. For The Longest Time... (4 points)

An algorithm takes 50 steps for input size 100. How many steps will it take for input size 500 if the running time is the following (assume low-order terms are negligible):

(a) [1 Point] linear

(b) [1 Point] \( O(n \lg n) \)

(c) [1 Point] quadratic

(d) [1 Point] cubic

5. Budgeting Time (18 points)

For each function \( f(n) \) and time \( t \), determine the largest size \( n \) of a problem that can be solved in time \( t \), assuming that the algorithm to solve the problem takes \( f(n) \) \text{ microseconds} (1 second equals 1 million microseconds).

To do this, you will write the following methods:

```java
public static String budgetTimeA(long microseconds); // 500 \lg n
public static double budgetTimeB(long microseconds); // 1000n
public static double budgetTimeC(long microseconds); // 100n \lg n
public static double budgetTimeD(long microseconds); // 10n^2
public static double budgetTimeE(long microseconds); // 2n^3
public static double budgetTimeF(long microseconds); // (2^n)/20
```

Note that for budgetTimeA, you will return a String—not a number. That String should represent the answer using “mathematical notation” (e.g. \( 1g(2^{-5}/5-4+(3\sqrt{4})) \) would be a valid result). Because this is a String, you will have to do part A analytically at the very least.

To write and submit your answer for this question, go to:

https://grinch.cs.washington.edu/cse332/t1/