Homework #3

You should try to solve these problems by yourself. I recommend that you start early and get help in office hours if needed. If you find it helpful to discuss problems with other students, go for it. **You do not need to turn in these problems. The goal is to be ready for the in class quiz that will cover the same or similar problems.**

Problem 1: Asymptotic Analysis

Prove that $o(g(n)) \cap \omega(g(n))$ is the empty set.

Solution

By the definition of little-oh, a function $f_1(n)$ cannot be a subset of $o(g(n))$ and be a polynomial of the same degree as $g(n)$ since $cf_1(n)$ must be greater than $g(n)$ for all $c > 0$. Similarly using the definition of little-omega any function $f_2(n)$ which is $\omega(g(n))$ must be of a lesser degree than $g(n)$. Therefore there cannot exist a function which belongs to both sets.

Problem 2: Algorithm Analysis

Consider the following basic problem. You're given an array A consisting of n integers $A[1], A[2], \ldots, A[n]$. You'd like to output a two-dimensional n-by-n array B in which $B[i,j]$ (for $i < j$) contains the sum of array entries $A[i]$ through $A[j]$—that is, the sum $A[i]+A[i+1]+\cdots+A[j]$. (The value of array entry $B[i,j]$ is left unspecified whenever $i \geq j$, so it doesn't matter what is output for these values.) Below is a simple algorithm to solve this problem:

```
1  for $i \leftarrow 1$ to $n$
2        do for $j \leftarrow i+1$ to $n$
3              Add entries $A[i]$ through $A[j]$
4              Store the result in $B[i,j]$
```

(a) Give a function $f(n)$ that is an asymptotically tight bound on the running time of the algorithm above. Using the pseudocode above, argue that the algorithm is, in fact $\Theta(f(n))$.

Solution

$\Theta(n^3)$. Why? The two for loops each run on the order of n times. Adding $O(n)$ entries takes $O(n)$ time. Therefore the total time is $n^2 * n$ or $\Theta(n^3)$.

(b) Although the algorithm you analyzed above is the most natural way to solve the problem, it contains some highly unnecessary sources of inefficiency. Give a different algorithm to solve this problem with an asymptotically better running time than the provided algorithm.

Solution

```
for $i \leftarrow 1$ to $n$
    do for $j \leftarrow i+1$ to $n$
        do $B[i,j] = B[i,j-1] + A[j]$
```

(c) What is the running time of your new algorithm?

Solution

$\Theta(n^2)$

Problem 3: Algorithms and Decision Trees

You are given 9 identical looking balls and told that one of them is slightly heavier than the others. Your task is to identify the defective ball. All you have is a balanced scale that can tell you which of two balls is heavier.

(a) Show how to identify the heavier ball in just 2 weighings.

Solution

Divide the balls into three sets of three. Compare two sets. If one is heavier, recurse on that set. If they weigh the same, recurse on the third set. Weigh any two of the remaining three balls. If one is heavier that's your ball. If they weigh the same, the left over ball is the heavier one.

(b) Give a decision tree lower bound showing that it is not possible to determine the defective ball in fewer than 2 weighings.

Solution

Any decision in this tree has three possible outcomes: the compared sets weigh the same; set A is heavier or set B is heavier. There are 9 possible outcomes (any one of the balls can be the heavy one. Any tree with branching factor 3 has at least 3^h leaves. So we set 3^h equal to 9 and solve for h. Take the log of both sides; $h = 2$. Thus to be able to reach any of the 9 possible outcomes, I must make at least two weighings.

Problem 4: Efficiency

For a given problem, suppose you have two algorithms: A_1 and A_2 with worst-case time complexity of $T_1(n)$ and $T_2(n)$, respectively. For each part, answer the following four questions:

- Is $T_1(n) = O(T_2(n))$?
- Is $T_1(n) = \Omega(T_2(n))$?
- Is $T_1(n) = \Theta(T_2(n))$?
- If your goal is to pick the fastest algorithm for large n, would you pick A_1 or A_2?

(a) $T_1(n) = 5\sqrt{n} + \log_2 n + 3$ and $T_2(n) = 100\log_2 n + 25$

Solution

- Is $T_1(n) = O(T_2(n))$? **NO**
- Is $T_1(n) = \Omega(T_2(n))$? **YES**
- Is $T_1(n) = \Theta(T_2(n))$? **NO**
- If your goal is to pick the fastest algorithm for large n, would you pick A_1 or A_2? $\mathbf{A_2}$.

(b) $T_1(n) = 5\sqrt{n} + \ln n$ and $T_2(n) = \sqrt{n} \cdot \log_2 n$

Solution

- Is $T_1(n) = O(T_2(n))$? **YES**
- Is $T_1(n) = \Omega(T_2(n))$? **NO**
- Is $T_1(n) = \Theta(T_2(n))$? **NO**
- If your goal is to pick the fastest algorithm for large n, would you pick A_1 or A_2? $\mathbf{A_1}$

(c) $T_1(n) = 5\log_{10} n - 20$ and $T_2(n) = \frac{1}{2}\log_2 n$

Solution

- Is $T_1(n) = O(T_2(n))$? **YES**
- Is $T_1(n) = \Omega(T_2(n))$? **YES**
- Is $T_1(n) = \Theta(T_2(n))$? **YES**
- If your goal is to pick the fastest algorithm for large n, would you pick A_1 or A_2? $\mathbf{A_2}$

(d) $T_1(n) = 10n^{2/4}$ and $T_2(n) = \sqrt{n} \cdot \log_2 n$

Solution

- Is $T_1(n) = O(T_2(n))$? **YES**
- Is $T_1(n) = \Omega(T_2(n))$? **NO**
- Is $T_1(n) = \Theta(T_2(n))$? **NO**
- If your goal is to pick the fastest algorithm for large n, would you pick A_1 or A_2? $\mathbf{A_1}$.

(e) $T_1(n) = 10^{(2\log_{10} n)} - 2n$ and $T_2(n) = 5n^3 + 10n - 3$

Solution

- Is $T_1(n) = O(T_2(n))$? **YES**
- Is $T_1(n) = \Omega(T_2(n))$? **NO**
- Is $T_1(n) = \Theta(T_2(n))$? **NO**
- If your goal is to pick the fastest algorithm for large n, would you pick A_1 or A_2? $\mathbf{A_1}$