EE360C: Algorithms
University of Texas at Austin — Problem Set #3
Dr. Evdokia Nikolova & Dr. Pedro Santacruz — Due: September 25, 2014 (*in class quiz*)

Problem Set #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 these problems in. The goal is to be ready for the in class quiz that will cover the same or similar problems.**

Problem 1: Asymptotic Time Complexity

Consider each of the following pairs of functions. For each pair, either $f(n) = O(g(n))$, $f(n) = \Omega(g(n))$ or $f(n) = \Theta(g(n))$. Determine which of these three options best captures the relationship and (briefly) explain or demonstrate why.

(a) $f(n) = \log n^2$; $g(n) = \log n + 5$

Solution
$f(n) = \Theta(g(n))$. Why? $\log n^2 = 2 \log n$.

(b) $f(n) = \sqrt{n}$; $g(n) = \log n^2$

Solution
$f(n) = \Omega(g(n))$. Why? Polynomials grow faster than logs.

(c) $f(n) = \log^2 n$; $g(n) = \log n$

Solution
$f(n) = \Omega(g(n))$. Why? Polylogarithmic functions grow faster than logs.

(d) $f(n) = n$; $g(n) = \log^2 n$

Solution
$f(n) = \Omega(g(n))$. Why? Polynomials grow faster than logs.

(e) $f(n) = n \log n + n$; $g(n) = \log n$

Solution
$f(n) = \Omega(g(n))$. Why? Polynomials grow faster than logs.

(f) $f(n) = 10$; $g(n) = \log 10$

Solution
$f(n) = \Theta(g(n))$. Why? They're both constants.

(g) $f(n) = 2^n$; $g(n) = 10n^2$

Solution
$f(n) = \Omega(g(n))$. Why? Exponentials grow faster than polynomials.

(h) $f(n) = 2^n$; $g(n) = 3^n$

Solution

$f(n) = O(g(n))$. Why? Consider the limit. $f(n)/g(n) = 0$ in the limit.

Problem 2: Stable Shipping Repair

Peripatetic Shipping Lines, Inc. is a shipping company that owns n shops and provides services to n ports. Each of its ships has a *schedule* that says, on each day of the month, which of the ports it is currently visiting, or whether it is out to sea. (You can assume the "month" here has m days for some $m > n$.) Each ship visits each port for exactly one day during the month. For safety reasons, PSL Inc. has the following strict requirement:

(†) *No two ships can be in the same port on the same day.*

The company wants to perform maintenance on all the ships this month, via the following scheme. They want to *truncate* each ship's schedule; for each ship S_i, there will be some day when it arrives in its scheduled port and simply remains there for the rest of the month (for maintenance). This means that S_i will not visit the remaining ports on its schedule (if any) that month, but this is okay. So the *truncation* of S_i's schedule will simply consist of its original schedule up to a certain specified date on which it is in a port P; the remainder of the truncated schedule simply has it remain in port P.

Now the company's question to you is the following: given the schedule for each ship, find a truncation of each so that condition (†) continues to hold: no two ships are ever in the same port on the same day.

Show that such a set of truncations can always be found, and given an algorithm to find them.

Example. Suppose we have two ships and two ports, and the "month" has four days. Suppose that the first ship's schedule is:

port P_1; at sea; port P_2; at sea

and the second ship's schedule is:

at sea; port P_1; at sea; port P_2

Then the (only) way to choose truncations would be to have the first ship remain in port P_2 starting on day 3 and have the second ship remain in port P_1, starting on day 2.

Solution

For each schedule, we have to choose a *stopping port*: the port in which the ship will spend the rest of the month. Implicitly, these stopping ports will define truncations of the schedules. We will say that an assignment of ships to stopping ports is *acceptable* if the resulting truncations satisfy the conditions of the problem—specifically, condition (†). (Note that because of condition (†), each ship must have a distinct stopping port in any acceptable assignment.

We set up a stable marriage problem involving ships and ports. Each ship ranks each port in chronological order of its visits to them. Each port ranks each ship in reverse chronological order of their visits to it. Now we simply have to show:

> *A stable matching between ships and ports defines an acceptable assignment of stopping ports.*

Proof. If the assignment is not acceptable, then it violates condition (†). That is, some ship S_i passes through port P_k after ship S_j has already stopped there. But in this case, under our preference relation above, ship S_i "prefers" P_k to its actual stopping port, and port P_k "prefers" ship S_i to ship S_j. This contradicts the assumption that we chose a stable matching between ships and ports.

Problem 3: 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 $f_1(n) > cg(n)$ 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 $g(n)$, since $f_2(n) < cg(n)$. There cannot exist a function that satisfies both $f(n) > cg(n)$ and $f(n) < cg(n)$, so $o(g(n)) \cap \omega(g(n))$ is the empty set.

Problem 4: 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 ← 1 to n
2        do for j ← 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

```
1  for i ← 1 to n
2        do for j ← i + 1 to n
3                 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 5: 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 sides is heavier of if the two sides weigh the same.

(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.