COP 5407: Solutions to Midterm Review Problems; Spring 2017

---

1. State and prove the correct relationship $(O, o, \Omega, \omega, \Theta)$ between the functions $f(n) = n^2 \log n + 2n$ and $g(n) = n^3 + 4n \log n$.

   Claim: $f(n) = O(g(n))$.

   We prove this claim by showing that there exists $c, n_0 > 0$

   $$\begin{aligned} n^2 \log n + 2n &\leq c(n^3 + 4n \log n), \forall n \geq n_0, \\ i.e., n \log n + 2 &\leq c(n^2 + 4 \log n), \forall n \geq n_0, \end{aligned}$$

   which is true for $n_0 = 2$ and $c = 1$.

   Claim: $f(n) = o(g(n))$.

   We prove this claim by showing that $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$. This is easily shown by dividing the numerator and denominator by n^3, giving us

   $$\lim_{n\to\infty} \frac{n^2 \log n + 2n}{n^3 + 4n \log n} = \frac{\frac{\log n}{n} + \frac{2}{n}}{1 + \frac{4 \log n}{n^2}} = 0$$

   Note that both terms in the numerator go to 0 as n tends to ∞, while in the denominator, one term goes to 0, while the other is 1. Hence the claim.

   Claim: $f(n)$ has no other relationship (Ω, ω, Θ) with $g(n)$.

   Since $f(n) = o(g(n))$, we know that $f(n)$ is asymptotically slower than $g(n)$. Thus, neither $f(n) = \Omega(g(n))$ nor $f(n) = \omega(g(n))$ can possibly be true. Since $f(n) = \Omega(g(n))$ is false, $f(n) = \Theta(g(n))$ is also false.

2. Solve the following recurrence relations using any of the 3 methods we have discussed in class:

   (a) $T(n) = 2T(2n/3) + O(n)$

   Solution: We replace $O(n)$ by cn for some positive constant c. We will now argue that Case 1 of Master theorem applies here. First, $a = 2, b = 3/2 = 1.5$, and $\log_{1.5} 2 > 1 + \epsilon$, for some positive ϵ. Thus, $f(n) = cn^1 = O(n^{\log_{1.5} 2 - \epsilon})$ and $T(n) = \Theta(n^{\log_{1.5} 2})$.

   (b) $T(n) = \frac{2}{3}T(2n) + O(n)$

   Solution: We replace $O(n)$ by cn for some positive constant c. We will now argue that Case 3 of Master theorem applies here. First, $a = 3/2 = 1.5, b = 2$, and $\log_2 1.5 < 1 - \epsilon$, for some positive ϵ. Furthermore, $af(n/b) \leq df(n)$ is satisfied because $\frac{3n}{4} \leq dn$, for $d = 3/4 < 1$. Thus, $T(n) = \Theta(n)$.

(c) $T(n) = 2T(2n/3) + O(n^2)$

(d) $T(n) = 2T(n/2) + O(n^2)$

(e) $T(n) = 2T(n/4) + 1$

Solution: Case 1 of Master theorem applies here because $a = 2, b = 4$, $n^{\log_4 2} = n^{0.5} = \sqrt{n}$, and $f(n) = 1 = O(n^{0.5-\epsilon})$. Thus $T(n) = \Theta(\sqrt{n})$.

(f) $T(n) = 2T(n/4) + \sqrt{n}$

Solution: Case 2 of Master theorem applies here because $a = 2, b = 4$, and $n^{\log_4 2} = n^{0.5} = \sqrt{n} = f(n)$. Thus $T(n) = \Theta(\sqrt{n} \log n)$.

(g) $T(n) = 2T(n/4) + n$

Solution: Case 3 of Master theorem applies here because $a = 2, b = 4$, $n^{\log_4 2} = n^{0.5} = \sqrt{n}$, and $f(n) = n^1 = \Omega(n^{0.5+\epsilon})$. Furthermore, $af(n/b) \leq cf(n)$ is satisfied because $2n/4 \leq cn$, for $c = 1/2 < 1$. Thus $T(n) = \Theta(n)$.

(h) $T(n) = 2T(n/4) + n^2$

Solution: Case 3 of Master theorem applies here because $a = 2, b = 4$, $n^{\log_4 2} = n^{0.5} = \sqrt{n}$, and $f(n) = n^2 = \Omega(n^{0.5+\epsilon})$. Furthermore, $af(n/b) \leq cf(n)$ is satisfied because $2(n/4)^2 \leq cn^2$, for $c = 1/8 < 1$. Thus $T(n) = \Theta(n^2)$.

3. The standard implementation of INSERTIONSORT (shown below) operates by inserting (in iteration j) $A[j]$ into its appropriate location in the sorted subarray $A[1 \ldots j-1]$. However, the right location is computed by a linear search (while-loop in lines 4 through 7), which has a worst-case time complexity linear in its length ($O(j)$). Can INSERTIONSORT be speeded up by replacing the linear search by a binary search with a logarithmic worst-case time complexity? What would be the time complexity of the modified INSERTIONSORT?

Algorithm 1 INSERTIONSORT(A)

```
1: for j ← 2 to length[A] do
2:     key ← A[j]
3:     i ← j − 1                      ▷ Insert A[j] into sorted subarray A[1 . . . j − 1]
4:     while i > 0 and A[i] > key do
5:         A[i + 1] ← A[i]
6:         i ← i − 1
7:     end while
8:     A[i + 1] ← key
9: end for
```

Solution: By replacing linear search with binary search, we would reduce the worst-case number of comparisons made by the algorithm from $O(n^2)$ to $O(n \log n)$. This is because the number of comparisons is $O(\log j)$ for the j-th iteration, which when summed over all iterations gives $O(n \log n)$. However, the number of data movements is exactly the same as that incurred by InsertionSort. Thus the overall worst-case time complexity remains the same.