**Algorithms (CSCI 323 & 700)**

**Spring 2016**

**Solutions to Homework #5**

1. Stirling’s approximation formula for factorials is n! ~ √(2πn) (n/e)n. Use it to find a better approximation of log(n!) than the “n log n” upper bound obtained in last week’s homework.

**log n! ~ log [√(2πn) (n/e)n] = log √(2πn) + log (n/e)n = ½ (log 2 + log π + log n) + n (log n – log e) = ½ (log 2 + log π + log n) + n log n – n. Note that it is still Θ(n log n).**

1. Use the Master Method to find the order of growth for T(n) = 8T(n/2) + n2.

**a = 8, b = 2, d = 2. log2 8 > 2, so case 1 applies, T(n) = Θ(n log2 8) = Θ(n3)**

1. Use the Master Method to find the order of growth for T(n) = 3T(n/3) + n3.

**a = 3, b = 3, d = 3. log3 3 > 3, so case 3 applies, T(n) = Θ(n3)**

1. Assuming that n = 22^k, solve the recurrence T(2) = 1, T(n) = T(√n) + 1.

**If n = 22^k, then T(n) = T(√(22^k)) + 1 = T(22^k/2) + 1 = T(22^(k-1)) + 1**

**Use a domain transformation S(k) = T(n), so S(k-1) = T(22^(k-1)) so the equations becomes S(k) = S(k-1) + 1.**

**Using telescoping gives S(k) – S(0) = k, or S(k) = k + 1, or T(n) = log2 log2 n + 1 = lg lg n + 1.**

1. Using the methods of last class, solve the recurrence T(1) = 1, T(n) = 1 + 2∑k=0..n-1 [T(k) + T(n-k-1)].

**T(n) = 1 + 2∑k=0..n-1 [T(k) + T(n-k-1)] = 1 + 4∑k=0..n-1 T(k)**

**T(n+1) = 1 + 4∑k=0..n T(k)**

**T(n+1) - T(n) = 0 + 4T(n)**

**T(n+1) = 5T(n)**

**T(n+1) / T(n) = 5**

**(T(n+1) / T(n)) \* (T(n) / T(n-1)) \* … \* (T(2) / T(1)) = T(n+1) / T(1) = T(n+1) = 5n (telescoping by multiplication)**

**Verify: T(2) = 1 + 2∑k=0..n-1 [T(k) + T(n-k-1)] = 1 + 2[(T(0) + T(1)) + (T(1) + T(0))] = 1 + 2(1 + 1) = 5.**

1. Suppose you have n unordered integers from the range {0 … n2 – 1}. What sorting algorithm would you use to order the data as efficiently as possible? What would be its time complexity?

**Use a radix sort, with radix = n, so our elements are two-digit numbers. Use a counting sort on each individual digit. Since the max for digit is n-1 and the number of elements is n, so the time for each digit is Θ(n), for a total of 2Θ(n), or Θ(n).**