Eddie Christopher Fox III

CS 325 Analysis of Algorithms

April 10, 2017

Assignment 1

(1 pt) Suppose we are comparing implementations of insertion sort and merge sort on the same machine. For inputs of size n, insertion sort runs in $8n^2$ steps, while merge sort runs in 64 n lgn steps. For what values of n does insertion sort run faster than merge sort?

n	8n^2	64n * log(n)
1	8	0
2	32	128
3	72	304
4	128	512
5	200	743
10	800	2,126
20	3,200	5,532
30	7,200	9,421
40	12,800	13,624
45	16,200	15,816
44	15,488	15,373

Between values n = 2- n = 44, insertion sort runs faster than merge sort.

(5 pts) For each of the following pairs of functions, either f(n) is O(g(n)), f(n) is Ω(g(n)), or f(n) = Θ(g(n)). Determine which relationship is correct and explain.

a. f(n) = n0.25; g(n) = n0.5

b. f(n) = n; g(n) = log2 n

c. f(n) = log n; g(n) = ln n

d. f(n) = 1000n2; g(n) = 0.0002n2 – 1000n

e. f(n) = nlog n; g(n) =$n\sqrt{n}$

f. f(n) = en; g(n) = 3n

g. f(n) = 2n; g(n) = 2n+1

h. f(n) = 2n; g(n) =22n

i. f(n) = 2n; g(n) = n!

j. f(n) = lgn; g(n) = $\sqrt{n}$

a. F(n) is O(g(n). The limit of F(n) / g(n) is 0.

b. f(n) is omega(g(n)). Log n is of a lesser complexity than n. N grows faster than Log n.

c. f(n) is theta(g(n)). Log n and ln n have roughly equal complexity and mainly differ by a constant factor. They grow at the same rate.

d. F(n) is theta(g(n)). At high enough values of n, the lower terms become insignificant, and a constant difference doesn’t matter.

e. F(n) is O(g(n)). Both have n times something, but square root grows faster than log, so g(n) will grow faster than f(n).

f. f(n) is O(g(n)). G(n) will grow faster because the number that is being exponentiated is higher. The difference between them grows at faster than a linear rate and can’t be expressed by a constant.

g. F(n) is theta(g(n)). G(n) will be greater than F(n) no matter what value of N, but they will only differ by a constant factor of 2. As in F(n) will be half of what G(n) is, as they have the same growth rate.

h. f(n) is O(g(n)) . G(n) clearly grows much faster than F(n) when G(n) is 2 raised to the f(n) power.

i. f(n) is O(g(n)) because factorial grows faster than exponential.

j. f(n) is O(g(n) because square root grows faster than logarithmic.

(5 points)

a. Describe in words and give pseudocode for an efficient algorithm that determines the maximum and minimum values in a list of n numbers.

Use a for loop that goes through the array and compares pairs of values in the array. The larger of the two is compared with the global maximum and assigned if greater. The smaller of the two is compared with the global minimum and assigned if lesser.

Psudeocode on next page.

Psudeocode:

```
Maximum = Array[0]
Minimum = Array[0]

// Low and high prevent out of bounds

Low = Array.length mod 2
High = floor(Array.length /2) * 2

For (I = low < I high)
If array[i] > array [i+1]
        If array[i] > maximum
                Maximum = array[i]
        If array[i+1] < minimum
                Minimum = array[i+1]
Else
        If array[i] < minimum
                Minimum = array[i]
        If array[i+1] > maximum
                Maximum = array[i+1]
```

b. Show that your algorithm performs at most 1.5n comparisons.

There is N / 2 loops if there is an even amount of elements. or (n-1) /2 loops if there is an odd amount of elements. There are 3 comparisons per loop. 1 to compare array[i] and array[i+1], 1 to compare / assign the maximum, and 1 to assign/compare the minimum. The maximum amount of comparisons would be 3 [the amount of comparisons per loop] * (N / 2) [the number of loops], which is trivially 1.5n comparisons. If there is an odd number, it is even less, with 3 * ([N-1] / 2), which would be a bit less than 1.5n.

c. Demonstrate the execution of the algorithm with the input L = [9, 3, 5, 10, 1, 7, 12].

Max = 9 Min = 9

I = 1

3 is array[i], 5 is array[i+1]

3 is less than 9 and becomes the new min. Max is still 9.

Max = 9 Min = 3

I = 3

10 is array[i], 1 is array[i+1]

10 is greater than 9 and becomes the new max. 1 is less than 3 and becomes the new min.

Max = 10 Min = 1

I = 5

7 is array[i], 12 is array[i+1]

12 is greater than 10 and becomes the new max. Min is still 1.

Final max = 12 Final Min = 1

(4 pts) Let f_1 and f_2 be asymptotically positive non-decreasing functions. Prove or disprove each of the following conjectures. To disprove give a counter example.

a. If $f_1(n) = O(g(n))$ and $f_2(n) = O(g(n))$ then $f_1(n) = \square(f_2(n))$.

b. If $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$ then $f_1(n) + f_2(n) = O(\max\{g_1(n), g_2(n)\})$

A. False.

Say G(n) = N^3, F(1) is N, and F(2) is N^2. It is true that F1 = O(g(n) because N < N^3 and also true that F2 = O(g(n)) because N^2 < N^3, and yet F1 is not theta F2 because N < N^2. They are not equal complexity. It's like saying that 10 and 30 are equal just because both are less than 50.

B.

For some constants n1, n2, g1, and g2, the following is true by definition.

F1(n) ≤ c1 * g1(n) for n ≥ n1 and F2(n) ≤ c2 * g2(n).

The functions are asymptotically positive, so F1(n) + F2(n) ≤ c1 * g(n) + c2 * g(2)

Which is ≤ c1 * max [g1(n), g2(n)] + c2 * max[g1(n), g2(n)]

Which is ≤ (c1 + c2) * max[g1(n), g2(n)]

If we assume A = (c1+ c2) and B = max(n1, n2)

Then

$F1(n) * f2(n) \leq K * \max [g1(n), g2(n)]$ for $n \geq B$

Which by definition means

$F1(n) + F2(n) = O(\max [g1(n), g2(n)]$, as was to be shown.