Spring 2022 COP 3502 Quiz #3 Version A (Algorithm Analysis, Sorting) Solutions

1) (5 pts) An algorithm processes n values in $O(3^n)$ time. When run on an input of size n = 15, the algorithm takes 100 ms. How long, **in seconds**, is the algorithm expected to take when run on an input of size n = 20?

Solution

Let $T(n) = c3^n$, represent the run time of the algorithm on an input of size n, for some constant c. Using the given information, we have: $T(15) = c(3^{15}) = 100$ ms → $c = \frac{100ms}{3^{15}}$.

Now, we must solve for T(20): $T(20) = \frac{100ms}{3^{15}} \times 3^{20} = (100ms) \times 3^5 = (.1 \text{ sec}) \times 243 = \mathbf{24.3 \text{ sec.}}$

Grading: 1 pt setting up T(n) equation,
1 pt solve c,
1 pt substitute 20,
1 pt plug in,
1 pt convert to seconds

2) (8 pts) Which of the two following expressions is larger, assuming that n is an integer greater than 1? Provide proof of your answer.

(a) $\sum_{i=1}^{n} i2^i$ (b) $(\sum_{i=1}^{n} i)(\sum_{i=1}^{n} 2^i)$

Solution

Choice b is larger.

This choice represents foiling out two expressions with n terms, resulting in n^2 total terms.

Choice a represents n separate terms. More importantly, **each term in the sum for (a) is one of the terms in the sum for (b), but (b) has an additional $n^2 - n$ terms, all of which must be positive, since each term is a product of two positive integers.** It follows that (b) must be larger because (b) equals (a) plus the sum of $n^2 - n$ other positive integers.

To see an illustration, let's plug in n = 2:

Sum (a) for n = 2 equals $1 \times 2^1 + 2 \times 2^2$

Sum (b) for n = 2 equals $(1 + 2) \times (2^1 + 2^2) = 1 \times 2^1 + 1 \times 2^2 + 2 \times 2^1 + 2 \times 2^2$

The terms in green represent the additional terms in the sum for (b) not included in the sum for (a). As n grows larger, the ratio of green terms to red terms increases. (In fact, as n approaches infinity, the ratio of (a) to (b) goes to 0.)

Grading: If they say choice (a) is larger grade must be 0 out of 8, no matter what.

3 pts for saying b is larger.

1 pts for observing that each term in (a) is contained in (b)

2 pts for PROVING that each term in (a) is in (b).

2 pts for PROVING (b) has OTHER terms, all positive that (a) doesn't.

3) (10 pts) Use the iteration technique to solve the following recurrence relation **EXACTLY.**

$$T(n) = 3T(n-1) + 2, for\ all\ integers\ n > 1$$
$$T(1) = 2$$

Please provide your answer as an exact function that is a closed form representation of T(n).

Solution

Iterate the formula three times:

T(n) **= 3T(n-1) + 2**
= 3(3T(n-2) + 2) + 2
= 9T(n-2) + 3 x 2 + 2
= 9T(n-2) + 8
= 9(3T(n-3) + 2) + 8
= 27T(n-3) + 18 + 8
= 27T(n-3) + 26

After k iterations, we have:

$T(n) = 3^kT(n-k) + (3^k - 1)$

Since we know T(1), plug in k = n-1 into this formula to get:

$T(n) = 3^{n-1}T(1) + (3^{n-1} - 1) = 3^{n-1}(2) + 3^{n-1} - 1 = 3^{n-1}(2+1) - 1 = 3^{n-1}(3) - 1 =$ **$3^n - 1$.**

Grading: 1 pt first iteration,
2 pts second iteration,
2 pts third iteration,
2 pts after k iterations,
1 pt plug in k = n-1
2 pts algebra to final answer (don't give full credit unless fully simplified)

4) (5 pts) Show the contents of the following array after each iteration of Insertion Sort:

Initial Values	7	1	2	6	5	4	3
1st iteration	1	7	2	6	5	4	3
2nd iteration	1	2	7	6	5	4	3
3rd iteration	1	2	6	7	5	4	3
4th iteration	1	2	5	6	7	4	3
5th iteration	1	2	4	5	6	7	3
Last iteration	1	2	3	4	5	6	7

Grading: 1 pt for each row, row must be fully correct to get the point.

5) (4 pts) Show the result of partitioning the array below, using the leftmost element as the partition element. Please use the in-place partitioning algorithm shown in class.

Initial Values	12	19	16	1	3	22	47	14	4	8	31
After Partition	3	8	4	1	12	22	47	14	16	19	31

Grading: 0 – 2 correct slots = 0
3 – 5 correct slots = 1
6 – 8 correct slots = 2
9 – 10 correct slots = 3
All correct slots = 4

6) (3 pts) What is the reason that Quick Sort and Merge Sort can both be sped up if we change the base case to be a subarray of size 40 and instead run Insertion Sort on this subarray?

Solution

Both algorithms are recursive. The number of recursive calls, particularly when breaking the array down into subarrays of size 0, 1 and 2 is a great amount of extra overhead in function calls for very little actual work. Without any extra function calls on the stack, the number of swaps that an Insertion Sort executes on small arrays amounts to less total work since all of the work is done in place. In fact, the overall superior efficiency of the design of Quick and Merge Sort doesn't overtake Insertion Sort until somewhere in the 20 – 50 range (for integers on a regular laptop).

Grading: Give Partial as Needed, look for clarity as well as understanding that there are many recursive calls for a very small amount of work as the array gets smaller and smaller and that the simplicity and lack of overhead of Insertion Sort is faster for some set of smaller values. Read a few and decide when to award 0, 1, 2 and 3.