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

1) (5 pts) A sorting algorithm sorts n values in O(nlgn) time. When run on an input of size $n = 2^{20}$, the algorithm takes 800 ms. How long, **in seconds**, is the algorithm expected to take when run on an input of size $n = 2^{25}$?

Let the algorithm take T(n) = cnlgn time for an input size of n, for some constant c. Using the given information, solve for c. We can choose the base for our logarithm (and keep it the same), so we can choose base 2:

$$T(2^{20}) = c(2^{20})log_2 2^{20} = 800ms$$
$$20c(2^{20}) = 800\ ms$$
$$c = \frac{40ms}{2^{20}}$$

Now, solve for $T(2^{25})$:

$$T(2^{25}) = c(2^{25})log_2 2^{25} = \frac{40ms}{2^{20}}(2^{25})25 = 40(2^5)(25)ms = 32 \times 1000ms = \mathbf{32\ sec}$$

Grading: 2 pts solving for c, 2 pts solving for $T(2^{25})$, 1 pt converting to seconds

2) (8 pts) Give a closed form solution for the following summation in terms of n:

$$\sum_{i=n}^{2n}(\sum_{j=n}^{2n} j)$$

$$\sum_{i=n}^{2n}(\sum_{j=n}^{2n} j) = \sum_{i=n}^{2n}(\left(\sum_{j=1}^{2n} j\right) - (\sum_{i=1}^{n-1} j))$$
$$= \sum_{i=n}^{2n}(\frac{2n(2n+1)}{2} - \frac{(n-1)n}{2})$$
$$= \frac{n}{2}\sum_{i=n}^{2n}(2(2n+1) - (n-1))$$
$$= \frac{n}{2}\sum_{i=n}^{2n}(4n+2-n+1)$$
$$= \frac{n}{2}\sum_{i=n}^{2n}(3n+3)$$
$$= \frac{n}{2}(3n+3)\sum_{i=n}^{2n} 1 = \frac{3n(n+1)}{2}(2n-n+1) = \mathbf{\frac{3n(n+1)^2}{2}}$$

Grading: 1 pt sum split, 1 pt sum to 2n, 1 pt sum to n-1, 3 pts algebra to reduce inner sum, 2 pts to multiply inner sum by (n+1) for outer sum.

3) (10 pts) Use the iteration technique to solve the following recurrence relation:

$$T(n) = 2T(n-1) + 2^n, for\ all\ integers\ n > 0$$
$$T(0) = 1$$

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

$$T(n) = 2T(n-1) + 2^n$$

Plug in for T(n-1) to yield:

$$T(n) = 2(2T(n-2) + 2^{n-1}) + 2^n$$
$$T(n) = 4T(n-2) + 2^n + 2^n$$
$$T(n) = 4T(n-2) + 2(2^n)$$

Now, plug in for T(n-2) to yield:

$$T(n) = 4(2T(n-3) + 2^{n-2}) + 2(2^n)$$
$$T(n) = 8T(n-3) + 2^n + 2(2^n)$$
$$T(n) = 8T(n-3) + 3(2^n)$$

After k iterations, we have:

$$T(n) = 2^k T(n-k) + k(2^n)$$

Set k = n to obtain:

$$T(n) = 2^n T(n-n) + n(2^n)$$
$$T(n) = 2^n T(0) + n(2^n)$$
$$T(n) = 2^n T + n(2^n)$$
$$T(n) = (\boldsymbol{n+1})(\boldsymbol{2^n})$$

Grading: 1 pt copying down first iteration, 2 pts to get to second iteration, 2 pts to get to third iteration, 2 pts for guess to k, 1 pt to plug in k=n, 2 pts to finish it up.

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

Initial Values	8	3	6	1	7	5	2
1st iteration	3	6	1	7	5	2	8
2nd iteration	3	1	6	5	2	7	8
3rd iteration	1	3	5	2	6	7	8
4th iteration	1	3	2	5	6	7	8
5th iteration	1	2	3	5	6	7	8
Last iteration	1	2	3	5	6	7	8

Grading: 1 pt per row, row must be perfect 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	6	9	3	12	5	2	13	18	4	1	7
After Partition	2	1	3	4	5	6	13	18	12	9	7

Grading: 4 pts if all 11 values correct, 3 pts if 9 or 10 are correct, 2 pts if 6, 7 or 8 are correct, 1 pt if 3, 4 or 5 are correct, 0 otherwise.

6) (3 pts) Why does Quick Sort run faster than Merge Sort, on average, in practice?

It runs faster because it does not have the overhead of using extra temporary arrays to copy values and copy back values into the original array. In short, Quick Sort runs in place, and this makes it run faster in practice even though it doesn't always split its input list into two halves for recursive calls.

Grading: Mostly all or nothing. Give full credit if they make any mention of not having to copy stuff to a new place and back or extra temporary memory.