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

1) (5 pts) An algorithm processes n values in $O(2^n)$ time. When run on an input of size n = 20, the algorithm takes 800 ms. How long, **in seconds**, is the algorithm expected to take when run on an input of size n = 23?

Let $T(n) = c2^n$ be the amount of time the algorithm takes on an input of size n for some constant c. Using the given information, we have:

$$T(20) = c2^{20} = 800ms$$
$$c = \frac{800ms}{2^{20}}$$

Now, let's solve for T(23):

$$T(23) = c2^{23} = \frac{800ms}{2^{20}} \times 2^{23} = 800ms \times 8 = 6400ms = \mathbf{6.4\ seconds}$$

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:

$$\sum_{i=n}^{2n}(\sum_{j=n}^{2n}(2i+1))$$

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

Grading: 1 pt factor out constant, 2 pts sum of 1, 2 pts split sum, 1 pt eval each sum, 1 pt simplify

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

$$T(n) = 3T(n-1) + 1, for\ all\ integers\ n > 0$$
$$T(0) = 1$$

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

$$T(n) = 3T(n-1) + 1$$

Now, let's substitute for T(n-1):

$$T(n) = 3(3T(n-2) + 1) + 1$$
$$T(n) = 9T(n-2) + (3+1)$$

Now, let's substitute for T(n-2)

$$T(n) = 9(3T(n-3) + 1) + (3+1)$$
$$T(n) = 27T(n-3) + (9+3+1)$$

After k iterations, we have:

$$T(n) = 3^k T(n-k) + \sum_{i=0}^{k-1} 3^i$$

Let k = n and substitute:

$$T(n) = 3^n T(n-n) + \sum_{i=0}^{n-1} 3^i$$

$$T(n) = 3^n + \sum_{i=0}^{n-1} 3^i = \sum_{i=0}^{n} 3^i = \frac{3^{n+1}-1}{3-1} = \mathbf{\frac{3^{n+1}-1}{2}}$$

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	12	13	6	8	5	3	9
1st iteration	12	6	8	5	3	9	13
2nd iteration	6	8	5	3	9	12	13
3rd iteration	6	5	3	8	9	12	13
4th iteration	5	3	6	8	9	12	13
5th iteration	3	5	6	8	9	12	13
Last iteration	3	5	6	8	9	12	13

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	20	16	40	90	33	18	47	8	6	27	55
After Partition	18	16	6	8	20	33	47	90	40	27	55

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 is the average case run time analysis of Merge Sort easier to work out mathematically than the average case run time analysis of Quick Sort?

The size of the inputs to the recursive calls are guaranteed to be the same size each time (half the size of the input array), where as in quick sort, the size of the inputs to the recursive calls can be anything from 0 to n-1, where n is the size of the input array, with each size occurring with some probability.

Recurrence relations with a simple fixed structure (and not a summation of recurrence terms) are easier to solve.

Grading: Give full credit if there's awareness of some sort that Merge Sort behaves more predictably, in terms of size of recursive calls.