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

1) (5 pts) A sorting algorithm sorts n values in $O(n\sqrt{n})$ time. When run on an input of size $n = 10^4$, the algorithm takes 73 ms. How long, **in seconds**, is the algorithm expected to take when run on an input of size $n = 10^6$?

Solution

Let $T(n) = cn\sqrt{n}$, for some constant c, represent the run time of the algorithm running on n values. Using the given information, we have:

$$T(10^4) = c10^4\sqrt{10^4} = 73ms$$

$$c10^4(10^2) = 73ms$$

$$c = \frac{73ms}{10^6}$$

Now, we must solve for $T(10^6)$:

$$T(10^6) = \frac{73ms}{10^6} \times 10^6\sqrt{10^6} = 73ms \times 10^3 = \mathbf{73}\boldsymbol{sec}$$

Notice that there are exactly 1000 ms in one second, so the conversion does itself!

Grading: 1 pt setting up T(n) equation,
1 pt solve c,
1 pt substitute 10^6,
1 pt plug in,
1 pt convert to seconds

2) (8 pts) Provide a closed form solution for the following summation, in terms of the variable n, which represents a positive integer. **Hint: Your answer should be a polynomial of degree 3, in n.**

$$\sum_{i=-n+1}^{n+1} i^3$$

Solution

We split the sum into four different parts: A part with negative terms, the 0 term, a part with corresponding positive terms, and the leftover positive terms. In essence, we partition the range [-n+1,n+1] into the ranges [-n+1,-1], [0], [1,n-1], [n, n+1].

$$\sum_{i=-n+1}^{n+1} i^3 = \left[\sum_{i=-n+1}^{-1} i^3\right] + 0^3 + \left[\sum_{i=1}^{n-1} i^3\right] + \sum_{i=n}^{n+1} i^3$$

Notice that the first sum is summing the cubes of the negative integers upto $-(n-1)^3$. Thus, we can re-express this sum by making the summation index positive and the base of the exponent negative:

$$= \left[\sum_{i=1}^{n-1} (-i)^3\right] + 0^3 + \left[\sum_{i=1}^{n-1} i^3\right] + \sum_{i=n}^{n+1} i^3$$

Since, $(-x)^3 = -x^3$, and we can factor out a constant from a sum, we have

$$= -\left[\sum_{i=1}^{n-1} i^3\right] + 0^3 + \left[\sum_{i=1}^{n-1} i^3\right] + \sum_{i=n}^{n+1} i^3$$

Now, it should be clear that the first three terms sum to 0 because the two sums cancel each other out. Thus, we're left with the last sum, which is two terms. Just plug these in and simplify.

$$= n^3 + (n+1)^3$$

$$= n^3 + n^3 + 3n^2 + 3n + 1$$

$$= \mathbf{2n^3 + 3n^2 + 3n + 1}$$

Grading: 4 pts for noticing that the terms in the sum from i = -n+1 to i = n-1 cancel out. No formal proof is needed, anything intuitive which shows that one term is the negative of the other (writing out terms with … is fine) is fine.

2 pts to plug in last two terms
2 pts to simplify into polynomial form

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

$$T(n) = 2T(n-1) + 6, for\ all\ integers\ n > 1$$
$$T(1) = 2$$

Please provide your answer as an exact function that is a closed form representation of T(n). (Your answer should be of the form $a \times b^n - c$, where a, b and c are positive integers.)

Solution

Iterate the formula three times:

T(n) = **2T(n-1) + 6**
= 2(2T(n-2) + 6) + 6
= 4T(n-2) + 2x6 + 6
= **4T(n-2) + 6(1 + 2)**
= 4(2T(n-3) + 6) + 6(1 + 2)
= 8T(n-3) + 4 x 6 + 6(1 + 2)
= **8T(n-3) + 6(1 + 2 + 4)**

After k iterations, we have:

$T(n) = 2^kT(n-k) + 6(\sum_{i=0}^{k-1} 2^i) = 2^kT(n-k) + 6(2^k - 1)$

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

$T(n) = 2^{n-1}T(1) + 6(2^{n-1} - 1) = 2^{n-1}(2) + 6(2^{n-1}) - 6 = 8 \text{ x } 2^{n-1} - 6 =$ **$4 \text{ x } 2^n - 6$**

Note: Probably a couple of better forms than the one I provided are $2^{n+2} - 6$ OR $2(2^{n+1} - 3)$.

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 (accept any of my 3 final forms)

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

Initial Values	7	1	2	6	5	4	3
1st iteration	1	2	6	5	4	3	7
2nd iteration	1	2	5	4	3	6	7
3rd iteration	1	2	4	3	5	6	7
4th iteration	1	2	3	4	5	6	7
5th iteration	1	2	3	4	5	6	7
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	6	19	16	1	3	12	47	14	4	8	31
After Partition	1	4	3	6	16	12	47	14	19	8	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.