COP 3502 Section 2: Algorithm Analysis Homework - Solutions

Assigned: Friday, June 12, 2020
Due: Friday, June 19, 2020 (via Webcourses Section 2)

Timing Problems

1) An algorithm with an $O(n^2)$ run time takes 24 ms to process input data with size n = 10000. How long will the algorithm take, in seconds, to process input data with size n = 50000?

Solution
Let the time the algorithm takes on an input of size n be modeled as $T(n) = cn^2$, for some constant c. Using the given information, we have:

$T(10000) = c(10000)^2 = 24ms$, thus, $c = \frac{24ms}{10^8}$.

Now, let's find T(50000):

$$T(50000) = c(50000)^2 = \frac{(24ms)50000^2}{10000^2} = (24ms)\left(\frac{50000}{10000}\right)^2 = 24 \times 25ms = \mathbf{600\ ms}.$$

Grading: 3 pts total - 1 pt set up equation, 1 pt solve c, 1 pt solve the question.

2) An algorithm that takes in as input an array with n rows and m columns has a run time of O(nlgm). The algorithm takes 173 ms to run in an input array with 1000 rows and 512 columns. How long will the algorithm take to run on an input array with 1500 rows and 4096 columns? (Note: For ease of calculation, please use a base of 2 for your logarithm.)

Solution
Let the time the algorithm takes on an input array with n rows and m columns be $T(n,m) = cnlg_2m$. Using the given information, we have:

$T(1000,512) = c(1000)(log_2 512) = 173\ ms$, thus, $c = \frac{173\ ms}{9000}$.

We must find T(1500,4096):

$$T(1500,4096) = c(1500)(log_2 4096) = \frac{173ms}{9000} \times 1500 \times 12 = 2 \times 173ms = \mathbf{346ms}$$

Grading: 3 pts total - 1 pt set up equation, 1 pt solve c, 1 pt solve the question.

3) An algorithm has a run time of $O(n^k)$ for some integer k. On an input of size 500, the algorithm takes 16 seconds to run. On an input of size 750, the algorithm takes 81 seconds to run. What is the value of k?

Solution

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

$T(500) = c(500)^k = 16$ sec
$T(750) = c(750)^k = 81$ sec

Divide the second equation by the first to yield: $\frac{c(750)^k}{c(500)^k} = \frac{81}{16}$.

Simplifying, we get $(\frac{750}{500})^k = (\frac{3}{2})^k = \frac{81}{16}$. Taking log of both sides, we see that **k = 4.** (With these numbers, it's easiest just to notice that 3 to the fourth power is 81 and 2 to the fourth power is 16.

Grading: 3 pts total - 1 pt set up equation, 1 pt divide, 1 pt solve for k.

Summation Problems

4) What is the value of the following summation: $\sum_{i=30}^{75}(3i + 4)$?

Solution

$$\sum_{i=30}^{75}(3i + 4) = 3\sum_{i=30}^{75} i + \sum_{i=30}^{75} 4$$

$$= 3\left(\sum_{i=1}^{75} i - \sum_{i=1}^{29} i\right) + 4(75 - 30 + 1)$$

$$= 3\left(\frac{75 \times 76}{2} - \frac{29 \times 30}{2}\right) + 4(46)$$

$$= 3(75 \times 38 - 29 \times 15) + 184$$

$$= 3(2850 - 435) + 184 = 3(2415) + 184 = \mathbf{7429}$$

Alternate solution is that the sum is an arithmetic series with first term 94, last term 229, with 46 terms, so the sum is $\frac{(94+229)}{2} \times 46 = 323 \times 23 = 7429$.

Grading: 4 pts total - grade proportionately, 4 pts for a valid correct answer, 3 pts if one piece is incorrect, 2 pts if about 1/2 correct, 1 pt if some work but less than half, 0 otherwise.

5) What is the value of the following summation in terms of n: $\sum_{i=n+1}^{3n}(2i-3)$?

Solution

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

$$=2\sum_{i=1}^{3i}i-\sum_{i=1}^{3n}3-(2\sum_{i=1}^{n}i-\sum_{i=1}^{n}3)$$

$$=\frac{2(3n)(3n+1)}{2}-3(3n)-\frac{2n(n+1)}{2}+3n$$

$$=9n^2+3n-9n-n^2-n+3n$$

$$=\boldsymbol{8n^2-4n}$$

Grading: 4 pts total - grade proportionately, 4 pts for a valid correct answer, 3 pts if one piece is incorrect, 2 pts if about 1/2 correct, 1 pt if some work but less than half, 0 otherwise.

6) Determine the value of this summation, in terms of a, b, c and d. Assume that a < b and that all four are positive integers. $\sum_{i=a}^{b}(ci+d)$.

Solution
This one is faster to solve using the arithmetic series formula. The first term of the sequence is ca+d. The last term of the sequence is cb+d. The number of terms is b - a + 1. It follows that the sum is:

$$\frac{(ca+d+cb+d)(b-a+1)}{2}=\frac{(\boldsymbol{c(a+b)+2d})(\boldsymbol{b-a+1})}{\boldsymbol{2}}$$

Note: The equivalent form you get by plugging into the usual summation formulas looks significantly different than this. Both are valid, of course.

Grading: 4 pts total - grade proportionately, 4 pts for a valid correct answer, 3 pts if one piece is incorrect, 2 pts if about 1/2 correct, 1 pt if some work but less than half, 0 otherwise.

Big-Oh Simplification

For each of these problems, take the given function and rewrite it as the most simple Big-Oh class the function belongs to:

7) $f(n) = 3n^{10} - 99n^9 + 3n + 2$

Solution
This function is **$O(n^{10})$,** all terms are polynomial terms with this being the term with the maximal exponent.
Grading: 2 pts if correct answer, 1 pt if close, 0 otherwise.

8) f(n) = 3nlgn + (.0001)n^2 + 20000

Solution
This function is **O(n^2).** In comparing nlgn to n^2, simply note that lg n grows more slowly than n raised to any positive power, so nlgn grows slower than n(n), which is n^2.

Grading: 2 pts if correct answer, 1 pt if close, 0 otherwise.

9) $f(n) = \frac{n^3+3n^2+7}{2n^2-(n+7)(2n+3)+27n}$

Solution
The denominator is unsimplified. Let's simplify it:

$$2n^2 - (n+7)(2n+3) + 27n = 2n^2 - (2n^2 + 17n + 21) + 27n = 2n^2 - 2n^2 - 17n - 21 + 27n$$
$$= 10n - 21$$

It follows that the given function is a cubic function divided by a linear function, which results in polynomial and fraction with the polynomial being a quadratic. It follows that this function is **O(n^2).**

Grading: 2 pts if correct answer, 1 pt if close, 0 otherwise.

10) $f(n) = \lg n + \lg n^2 + \lg(\lg n) + (lg^2 n)$

Solution
The first two terms are equivalent to each other in Big-Oh, due to the log power rule. (The latter term is 2 times the former.) The third term grows SMALLER than the first two, since we are taking the log a second time, which "brings down" the value of the input. The last term is the first term squared. Squaring any increasing term will create a term that grows faster. It follows that this function is **O($\lg^2 n$).**

Grading: 2 pts if correct answer, 1 pt if close, 0 otherwise.

11) $f(n) = n^{1.999} + \frac{n^2}{\lg n}$

Solution
Since log grows SLOWER than any polynomial term raised to any positive power, it follows that the second term grows FASTER than $\frac{n^2}{n^{0.000001}} = n^{1.999999}$. And this function grows faster than the first function, since the exponent is slightly higher. It follows that the dominating term is the second one and the function is **O($\frac{n^2}{\lg n}$).**

Grading: 2 pts if correct answer, 1 pt if close, 0 otherwise.

Code Segment Analysis Problems

Give a Big-Oh run time for each of the following functions, in terms of the variables in the problem. (These will typically be n and m, respectively, and some of the problems will only have 1 variable and others will have two.) Provide a succinct proof/reason for your answer.

12)

```
int f(int n, int m) {
    int sum = 0;
    for (int i=0; i<n; i++)
        sum++;
    for (int i=0; i<m; i++)
        sum++;
    return 0;
}
```

Solution
The first loop runs n times, the second runs m times. Inside both are a constant amount of work. The overall run time is **O(n+m).**

Grading: 1 pt, only if answer is correct. O(max(n,m)) is also acceptable.

13)

```
int f(int n, int m) {
    int sum = 0;
    for (int i=0; i<n; i++)
        for (int j=0; j<m; j++)
            sum++;
    return sum;
}
```

Solution
The loops are nested and the inner loop always runs m times, and this is repeated n times, so the total run time is **O(nm),** since the code inside the innermost loop takes constant time.

Grading: 1 pt, only if answer is correct.

14)

```
int f(int* array, int n, int target) {
    int low = 0, high = n-1;
    while (low < high) {
        int mid = (low+high)/2;
        if (array[mid] < target)
            low = mid+1;
        else if (array[mid] > target)
            high = mid-1;
        else
            return 1;
    }
    return 0;
}
```

Solution

This is a binary search. The difference between low and high starts at n, and gets divided by 2 each time. The search, in the worst case, stops after this difference gets to 0 (one step after it gets to 1). Thus, if we let k equal the number of times the loop runs, we find that $\frac{n}{2^k} = 1$, which means $2^k = n$ and $k = \log_2 n$. Since the work inside of the loop is constant time (no functions or loops, all simple statements), it follows that the run time of this function is **O(lg n)..**

Grading: 2 pts if correct, 1 pt if some work but incorrect, 0 if no work towards solution

15)

```
int f(int** array, int n) {
    int x = 0, y = 0;
    while (x < n) {
        y = 0;
        while (y < n && array[x][y] != 0)
            y++;
        x++;
    }
    return y;
}
```

Solution

In the worst case, the array only has 1s in it (all non-zero values), and in this case, the inner loop will always run n times, and will be repeated exactly n times, for a total run time of **$O(n^2)$.**

Grading: 1 pt, only if answer is correct.

16)

```
int f(int** array, int n) {
    int x = 0, y = 0;
    while (x < n) {
        while (y < n && array[x][y] != 0)
            y++;
        x++;
    }
    return y;
}
```

Solution

Here we don't reset y = 0 before the inner loop. This means that y++ can run at most n times. In addition, x++ can run at most n times. Although no individual loop is strictly bounded, once we have limits for how many times x++ and y++ can run, those limits apply to checking of the boolean conditions as well. Thus, the total worst case run time is **O(n),** a constant times n.

Grading: 2 pts - 1 for answer, 1 for justification

17)

```
int f(int n) {
    int sum = 0;
    while (n > 0) {
        sum += (n%2);
        n /= 2;
    }
    return sum;
}
```

Solution

This is very similar to binary search. We are repeatedly dividing by 2 until n goes to 0 and doing a constant amount of work in the loop. The analysis is identical to the binary search, thus the run time is **O(lg n).**

Grading: 2 pts if correct, 1 pt if some work but incorrect, 0 if no work towards solution