You will submit your solution to this assignment to the Curator System (as `HW03`). Your solution must be either a plain text file (e.g., NotePad++) or a typed MS Word document; submissions in other formats will not be graded.

If you work with a partner, be sure to put both the name and PID of each partner at the beginning of the file you submit.

Except as noted, credit will only be given if you show relevant work.

In all questions about complexity, functions are assumed to be nonnegative.

1. [30 points] The analysis of a certain algorithm leads to the following complexity function (for the average case):

$$T(N) = 43N^2 \log N + 15N^2 + \log N + 2$$

Several computer science students offer their conclusions about the algorithm (quoted below). For each conclusion, state whether it is correct or incorrect, based on the given information, and give a brief justification of your answer; feel free to cite any relevant theorems from the course notes.

We should note first that T(N) is $\Theta(N^2 \log N)$. That should be clear intuitively, and it's easily proved by taking the relevant limit:

$$\lim_{N\to\infty} \frac{43N^2 \log N + 15N^2 + \log N + 2}{N^2 \log N} = \lim_{N\to\infty}\left(43 + \frac{15}{\log N} + \frac{1}{N^2} + \frac{2}{N^2 \log N}\right) = 43$$

Therefore T(N) is both $O(N^2 \log N)$ and $\Omega(N^2 \log N)$.

a) Jimmy Earl Neer of WVU says that the algorithm, on average, is $O(N^2)$.

Incorrect.
We are given that the algorithm is $\Theta(N^2 \log N)$, and that must be (intuitively) "bigger" than N^2. More formally, we could take the relevant limit to show that on average, the algorithm is actually strictly $\Omega(N^2)$.

b) Haskell Hoo V of UVA says no, the algorithm, on average, is $\Omega(N)$.

Correct... although not terribly informative.
From a), the algorithm is, on average, strictly $\Omega(N^2)$, and N^2 is strictly $\Omega(N)$, and Ω is transitive.

c) Jimmy Earl Neer retorts that the algorithm, on average, is actually $\Omega(N^2 \log N)$.

Correct.
This follows trivially from the fact that the algorithm is, on average, $\Theta(N^2 \log N)$.

d) Joe Hokie of VT offers the suggestion that the algorithm, on average, is $\Theta(N \log N)$.

Incorrect.
N^2 log N is strictly Ω(N log N).

e) Haskell Hoo V asserts that, in the worst case, the algorithm must be $O(N^2 \log N)$.

Incorrect.
We are given that the <u>average</u> performance is $\Theta(N^2$ log N). All that implies about the <u>worst</u> case performance cannot be any better than that. It does not imply anything about an upper bound on the worst case (except that it cannot be better than the average case).

f) Joe Hokie replies that, in the worst case, the algorithm could certainly be $\Theta(N^3)$.

Correct.
As said above, all we know about an upper bound on the worst case is that it must be at least as bad as the average case.

g) Haskell Hoo V says that, in the worst case, the algorithm must be $\Omega(\log N)$.

Correct (but weak).
The lower bound for the worst case must be at least as bad as the average case, N^2 log N, which is clearly Ω(log N).

h) Jimmy Earl Neer says that the algorithm's best case performance must be $\Omega(N^2 \log N)$.

Incorrect.
The best case performance must be at least as good as the average case, but we cannot say more than that. So, the best case is actually $O(N^2$ log N).

i) Joe Hokie says, no, the algorithm's best case performance could be $\Omega(N \log N)$.

Correct.
The best case performance cannot be worse than the average case, but it could be much better than that. Since N^2 log N is strictly Ω(N log N), it's certainly possible the best case is much better than N log N, but it is also certainly possible the best case is worse than N log N.

j) Haskell Hoo V then claims that the algorithm's best case performance must be $\Omega(\log N)$.

Incorrect.
The claim may be true, or false. All we can say about the best case performance is that it must be $O(N^2$ log N).

2. [24 points] Suppose that an algorithm takes 60 seconds for an input of 2^8 elements. Estimate how long the same algorithm, running on the same hardware, would take if the input contained 2^{16} elements, and that the algorithm's complexity function is:

a) $\Theta(N)$
b) $\Theta(\log N)$
c) $\Theta(N \log N)$
d) $\Theta(N^2)$

Assume that the low-order terms of the complexity functions are insignificant, and state your answers to the nearest tenth of a second. Be sure to show supporting work.

First, some general observations...

We know the number of instructions that can be executed per second will not change since we are running the same code on the same hardware. The only thing that changes is the number of instructions that will be executed, due to the increase in the size of the input.

Let T(N) be the complexity function for the algorithm. Note that this is independent of the hardware used to execute the algorithm.

Let's call the number of instructions executed per second S. Then, the time required with an input of size 2^8 would be given by $T(2^8)/S$ and we know that equals 60 seconds.

That's enough information to answer the individual questions:

a) Now, if T(N) is Θ(N), then T(N) = kN for some constant N (disregarding lower terms). That means that $T(2^{16}) = 2^{16}k = 256 \times 2^8k = 256 \times T(2^8)$.

Therefore, the running time for an input of size 2^{16}, would be

$$T(2^{16})/S = 256 \times T(2^8)/S = 256 \times 60 = 15360 \text{ seconds}$$

(That's 4 hours 16 minutes.)

b) Following the same approach as above, but knowing T(N) = k log N, we can show that

$$T(2^8) = k \log(2^8) = 8k \log(2) = 8k$$
$$T(2^{16}) = k \log(2^{16}) = 16k \log(2) = 16k$$

Now, we see that $T(2^{16}) = 2 \times T(2^8)$, and therefore the running time would be

$$T(2^{16})/S = 2 \times T(2^8)/S = 2 \times 60 = 120 \text{ seconds}$$

c) Following the same approach as above, but knowing T(N) = k N log N, we can show that

$$T(2^8) = k\,(2^8 \log 2^8) = 8k\,2^8$$

$$T(2^{16}) = k\,(2^{16} \log 2^{16}) = 2 \times 2^8 \times 8k\,2^8 = 512k\,2^8 = 512\,T(2^8)$$

Hence, the running time would be

$$T(2^{16})/S = 512\,T(2^8)/S = 512 \times 60 = 30720 \text{ seconds}$$

(That's 8 hours, 32 minutes.)

d) Again, following the same approach, but knowing $T(N) = kN^2$, we can show that

$$T(2^{16}) = k\,(2^{16})^2 = (2^8)^2 k (2^8)^2 = 2^{16}\,T(2^8)$$

Hence, the running time would be

$$T(2^{14})/S = 2^{16}\,T(2^8)/S = 2^{16} \times 60 = 3932160 \text{ seconds}$$

(That's 65536 minutes, or 45 days, 12 hours, 46 minutes.)

3. [24 points] Use theorems from the course notes to solve the following problems. Show work to support your conclusions.

a) Find the "simplest" function g(n) such that

$$f(n) = 17n^2 + 3n\log n + 1000 \text{ is } \Theta(g(n))$$

Directly from Theorem 13, this is $\Theta(n^2)$.

b) Find the "simplest" function g(n) such that

$$f(n) = 5n\log^2 n + 8n^2 \log n \text{ is } \Theta(g(n))$$

This must be $\Theta(n \log^2 n)$ or $\Theta(n^2 \log n)$... a limit computation settles the issue:

$$\lim_{n\to\infty} \frac{5n\log^2 n + 8n^2 \log n}{n^2 \log n} = \lim_{n\to\infty}\left(\frac{5\log n}{n} + 8\right) = 8 + \lim_{n\to\infty}\left(\frac{5/n\ln 2}{1}\right) = 8$$

So, f(n) is $\Theta(n^2 \log n)$.

c) Find the "simplest" function g(n) such that

$$f(n) = \sqrt{n} + \log\left(8n^2\right) \text{ is } \Theta(g(n))$$

Again, we have two candidates: sqrt(n) and 2 log(8n) (using log properties there).

$$\lim_{n\to\infty} \frac{\sqrt{n} + \log(8n^2)}{\sqrt{n}} = \lim_{n\to\infty}\left(1 + \frac{2\log(8n)}{n^{1/2}}\right) = 1 + \lim_{n\to\infty}\left(\frac{2/n\ln 2}{(1/2)n^{-1/2}}\right) = 1 + \lim_{n\to\infty}\left(\frac{4}{n^{1/2}\ln 2}\right) = 1$$

Therefore, f(n) is Θ(sqrt(n)).

4. [12 points] Using the counting rules from the course notes, find the exact-count complexity function T(n) for the following algorithm. Show details of your analysis, and simplify your answer.

```
for (r = 1; r <= 2*n; r++) {          // 1 before, 3 per pass, 2 to exit
   a[r][1] = 1;                       // 3
   for (c = 2; c <= r; c += 2) {      // 1 before, 3 per pass, 1 to exit
      a[r][c] = c * a[r][c-1];        // 7
   }
}
```

The interesting part of this is "how many passes do we make through the body of the inner loop"?

Suppose we count passes starting with p = 1. On pass p, c = 2 + 2(p-1) and we exit the inner loop when c > 2n. So, we go through the loop as long as 2 + 2(p-1) <= 2n.

Now, algebra is your friend:

$$2 + 2(p-1) \le r \Leftrightarrow 2p \le r \Leftrightarrow p \le \frac{r}{2} \Leftrightarrow p \le \left\lfloor \frac{r}{2} \right\rfloor$$

For our purposes, it is sufficiently accurate to simplify this to p <= r/2.

Based on the analysis of the individual lines, given in the comments above, we have the following operation count if we simplify the limit for the inner loop:

$$T(n) = 1 + \sum_{r=1}^{2n}\left(3 + 3 + 1 + \sum_{p=1}^{\lfloor r/2 \rfloor}\left(3+7\right) + 1\right) + 2 = \sum_{r=1}^{2n}\left(\sum_{p=1}^{\lfloor r/2 \rfloor}(10) + 8\right) + 3$$

$$= \sum_{r=1}^{2n}\left(10\lfloor r/2 \rfloor + 8\right) + 3 \le \sum_{r=1}^{2n}\left(5r + 8\right) + 3 = 5\frac{2n(2n+1)}{2} + 16n + 3$$

$$= 10n^2 + 21n + 3$$

5. [10 points] Use the definitions of O and Ω to prove that:

$$f(n) \text{ is } \mathrm{O}\big(g(n)\big) \text{ if and only if } g(n) \text{ is } \Omega\big(f(n)\big)$$

proof:
Suppose that f(n) is O(g(n)), then there are constants N > 0 and C > 0 such that for all n > N, f(n) <= Cg(n). But, since C > 0, we can divide through and show that for all n > N, g(n) >= (1/C)f(n). By definition, that means that g(n) is Ω(f(n)).

Now, suppose that f(n) is Ω(g(n)), then there are constants N > 0 and C > 0 such that for all n > N, f(n) >= Cg(n). Again, since C > 0, we can divide through and show that for all n > N, g(n) <= (1/C)f(n). Therefore, g(n) is O(f(n)).