Analysis of Algorithms: Solutions 2

```
            X
            X                    X
number of   X      X      X      X
homeworks   X      X      X      X                           X
            X      X      X      X      X      X             X
            X      X      X      X      X      X             X
     X      X      X      X      X      X      X      X      X      X
     ------------------------------------------------------------------
     0    0.5-1  1.5-2  2.5-3  3.5-4  4.5-5  5.5-6  6.5-7  7.5-8  8.5-9
                                   grades
```

The histogram shows the distribution of grades for the homeworks submitted on time.

Problem 1
Prove the following properties of asymptotic bounds:

(a) If $f(n) = \Theta(g(n))$ and $g(n) = \Theta(h(n))$, then $f(n) = \Theta(h(n))$.

Since $f(n) = \Theta(g(n))$, we conclude that there are some positive constants c_1, c_2, and n_1 such that, for all $n \geq n_1$, we have:

$$c_1 g(n) \leq f(n) \leq c_2 g(n).$$

Similarly, since $g(n) = \Theta(h(n))$, there exist some positive constants c_3, c_4, and n_2 such that, for all $n \geq n_2$:

$$c_3 h(n) \leq g(n) \leq c_4 h(n).$$

We may combine these two inequalities as follows:

$$c_1 c_3 h(n) \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \leq c_2 c_4 h(n).$$

We now define three new constants, c_5, c_6, and n_3:

$c_5 = c_1 c_3$,
$c_6 = c_2 c_4$,
$n_3 = \max(n_1, n_2)$.

Then, the last inequality implies that, for every $n \geq n_3$, we have:

$$c_5 h(n) \leq f(n) \leq c_6 h(n).$$

This inequality means that, by definition, $f(n) = \Theta(h(n))$.

(b) $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

If $f(n) = \Theta(g(n))$, then there are positive constants c_1, c_2, and n_0 such that, for every $n \geq n_0$, we have:

$$c_1 g(n) \leq f(n) \leq c_2 g(n).$$

This inequality immediately implies that $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

To prove the opposite direction, suppose that $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$. By definition of O, there are positive constants c_2 and n_2 such that, for all $n \geq n_2$:

$$f(n) \leq c_2 g(n).$$

Similarly, by definition of Ω, there are constants c_1 and n_1 such that, for all $n \geq n_1$:

$$f(n) \geq c_1 g(n).$$

In order to combine these two inequalities, we define $n_0 = \max(n_1, n_2)$. Then, for all $n \geq n_0$, we have:

$$c_1 g(n) \leq f(n) \leq c_2 g(n),$$

which implies that $f(n) = \Theta(g(n))$.

(c) If $f(n) = o(g(n))$ then $f(n) = O(g(n))$ and $f(n) \neq \Theta(g(n))$.

Since $f(n) = o(g(n))$, we conclude that, for any c, there exists some n_0 such that, for all $n \geq n_0$, we have $f(n) < c \cdot g(n)$.

To show that $f(n) = O(g(n))$, we need to find c_1 and n_1 such that, for all $n \geq n_1$, $f(n) \leq c_1 g(n)$. We may pick any $c_1 > 0$; by definition of o, there exists an adequate n_1.

To show that $f(n) \neq \Theta(g(n))$, we derive a contradiction. Suppose that $f(n) = \Theta(g(n))$. Then, there exist c_2 and n_2 such that, for all $n \geq n_2$, we have $f(n) \geq c_2 g(n)$. On the other hand, since $f(n) = o(g(n))$, there exists n_3 such that, for all $n \geq n_3$, we have $f(n) < c_2 g(n)$. Thus, if $n \geq \max(n_2, n_3)$, then we have both $f(n) \leq c_2 g(n)$ and $f(n) > c_2 g(n)$, which is a contradiction.

Problem 2

Give an example of functions $f(n)$ and $g(n)$ such that $f(n) \neq O(g(n))$ and $f(n) \neq \Omega(g(n))$.

Consider the following two functions:

$$f(n) = \begin{cases} n & \text{if } n \text{ is even;} \\ 1 & \text{if } n \text{ is odd.} \end{cases}$$

$$g(n) = \begin{cases} 1 & \text{if } n \text{ is even;} \\ n & \text{if } n \text{ is odd.} \end{cases}$$

For even n, $f(n)$ grows asymptotically faster than $g(n)$. On the other hand, for odd n, $f(n)$ grows asymptotically slower. Therefore, $g(n)$ is neither asymptotically lower bound nor asymptotically upper bound for $f(n)$.

Problem 3

Suppose that we have four algorithms, called A_0, A_1, A_2, and A_3, whose respective running times are n, n^2, $\lg n$, and 2^n. If we use a certain old computer, then the maximal sizes of problems solvable in an hour by these algorithms are s_0, s_1, s_2, and s_3.

Suppose that we have replaced the old computer with a new one, which is k times faster. Now the maximal size of problems solvable in an hour by A_0 is $k \cdot s_0$. What are the maximal problem sizes for the other three algorithms, if we run them on the new computer?

For A_1: On the old machine, the A_1 algorithm solves a problem of size s_1 in one hour. The running time of this algorithm on a problem of size s_1 is s_1^2; hence, $s_1^2 = 1$ hour.

The new machine is k times faster, which means that the running time of A_1 is n^2/k. We denote the size of the largest problem solvable in one hour by v_1; then, $v_1^2/k = 1$ hour.

We conclude that $v_1^2/k = s_1^2$ and, hence, $v_1 = s_1\sqrt{k}$. Thus, the maximal size of a problem solvable in one hour on the new machine is $s_1\sqrt{k}$.

For A_2: On the old machine, the A_2 algorithm solves a problem of size s_2 in one hour, which means that $\lg s_2 = 1$ hour. If we denote the maximal problem solvable in an hour on a new machine by v_2, then $\lg v_2/k = 1$ hour. We conclude that $\lg v_2/k = \lg s_2$, which implies that $v_2 = s_2^k$. Thus, the maximal problem solvable in one hour on the new machine is of size s_2^k.

For A_3: We denote the maximal problem solvable by A_3 on the new machine by v_3, and use a similar reasoning to obtain the equation $2^{v_3}/k = 2^{s_3}$, which implies that $v_3 = s_3 + \lg k$.

Problem 4

Determine asymptotic upper and lower bounds for each of the following recurrences. Make your bounds as tight as possible.

(a) $T(n) = 2T(n/2) + n^3$.

$$
\begin{aligned}
T(n) &= n^3 + 2T(\frac{n}{2}) \\
&= n^3 + 2((\frac{n}{2})^3 + 2T(\frac{n}{4})) \\
&= n^3 + 2(\frac{n}{2})^3 + 4T(\frac{n}{4}) \\
&= n^3 + 2(\frac{n}{2})^3 + 4((\frac{n}{4})^3 + 2T(\frac{n}{8})) \\
&= n^3 + 2(\frac{n}{2})^3 + 4(\frac{n}{4})^3 + 8T(\frac{n}{8}) \\
&\ \ \ \ldots \\
&= n^3 + 2(\frac{n}{2})^3 + 4(\frac{n}{4})^3 + 8(\frac{n}{8})^3 + 16(\frac{n}{16})^3 + ... \\
&= n^3 + \frac{n^3}{4} + \frac{n^3}{16} + \frac{n^3}{64} + \frac{n^3}{256} + ... \\
&= n^3(1 + \frac{1}{4} + \frac{1}{16} + \frac{1}{64} + \frac{1}{256} + ...) \\
&< n^3(1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + ...) \\
&< 2n^3
\end{aligned}
$$

We conclude that $n^3 < T(n) < 2n^3$, which implies that $T(n) = \Theta(n^3)$.

(b) $T(n) = T(n-1) + n$.

$$\begin{aligned} T(n) &= n + T(n-1) \\ &= n + (n-1) + T(n-2) \\ &= n + (n-1) + (n-2) + T(n-3) \\ &\quad \cdots \\ &= n + (n-1) + (n-2) + (n-3) + ... + 2 + 1 \\ &= \frac{n(n+1)}{2} \\ &= \Theta(n^2) \end{aligned}$$

(c) $T(n) = T(\sqrt{n}) + 1$.

We "unwind" the recurrence until reaching some constant value of n, for example, until $n \leq 2$:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq 2 \\ T(\sqrt{n}) + 1 & \text{if } n > 2 \end{cases}$$

For convenience, assume that $n = 2^{2^k}$, for some natural value k. Then, we can unwind the recurrence as follows:

$$\begin{aligned} T(n) &= 1 + T(\sqrt{2^{2^k}}) \\ &= 1 + T(2^{2^{k-1}}) \\ &= 1 + 1 + T(\sqrt{2^{2^{k-1}}}) \\ &= 1 + 1 + T(2^{2^{k-2}}) \\ &= 1 + 1 + 1 + T(\sqrt{2^{2^{k-2}}}) \\ &= 1 + 1 + 1 + T(2^{2^{k-3}}) \\ &\quad \cdots \\ &= 1 + 1 + 1 + ... + 1 + T(2) \qquad \triangleright \text{ the sum is of length } k \\ &= k + \Theta(1) \\ &= \Theta(k) \end{aligned}$$

Finally, we note that $k = \lg\lg n$ and, hence, $T(n) = \Theta(\lg\lg n)$.