2.1 (1) [20 points]

Although merge sort runs in $\Theta(n \lg n)$ worst-case time and insertion sort runs in $\Theta(n^2)$ worst-case time, the constant factors in insertion sort can make it faster in practice for small problem sizes on many machines. Thus, it makes sense to *coarsen* the leaves of the recursion by using insertion sort within merge sort when subproblems become sufficiently small. Consider a modification to merge sort in which $n/k$ sublists of length $k$ are sorted using insertion sort and then merged using the standard merging mechanism, where $k$ is a value to be determined.

**a.** Show that insertion sort can sort the $n/k$ sublists, each of length $k$, in $\Theta(nk)$ worst-case time.

**b.** Show how to merge the sublists in $\Theta(n \lg(n/k))$ worst-case time.

**c.** Given that the modified algorithm runs in $\Theta(nk + n \lg(n/k))$ worst-case time, what is the largest value of $k$ as a function of $n$ for which the modified algorithm has the same running time as standard merge sort, in terms of $\Theta$-notation?

**d.** How should we choose $k$ in practice?

Answer:
a. Insertion sort takes $\Theta(k^2)$ time per $k$-element list in the worst case. Therefore, sorting $n/k$ lists of $k$ elements each takes $\Theta(k^2 n/k) = \Theta(nk)$ worst-case time.

b. Just extending the 2-list merge to merge all the lists at once would take $\Theta(n \cdot (n/k)) = \Theta(n^2/k)$ time ($n$ from copying each element once into the result list, $n/k$ from examining $n/k$ lists at each step to select next item for result list).

To achieve $\Theta(n \lg(n/k))$-time merging, we merge the lists pairwise, then merge the resulting lists pairwise, and so on, until there’s just one list. The pairwise merging requires $\Theta(n)$ work at each level, since we are still working on $n$ elements, even if they are partitioned among sublists. The number of levels, starting with $n/k$ lists (with $k$ elements each) and finishing with 1 list (with $n$ elements), is $\lceil \lg(n/k) \rceil$. Therefore, the total running time for the merging is $\Theta(n \lg(n/k))$.

c. The modified algorithm has the same asymptotic running time as standard merge sort when $\Theta(nk + n \lg(n/k)) = \Theta(n \lg n)$. The largest asymptotic value of $k$ as a function of $n$ that satisfies this condition is $k = \Theta(\lg n)$.

To see why, first observe that $k$ cannot be more than $\Theta(\lg n)$ (i.e., it can’t have a higher-order term than $\lg n$), for otherwise the left-hand expression wouldn’t be $\Theta(n \lg n)$ (because it would have a higher-order term than $n \lg n$). So all we need to do is verify that $k = \Theta(\lg n)$ works, which we can do by plugging $k = \lg n$ into $\Theta(nk + n \lg(n/k)) = \Theta(nk + n \lg n - n \lg k)$ to get

$$\Theta(n \lg n + n \lg n - n \lg \lg n) = \Theta(2n \lg n - n \lg \lg n),$$

which, by taking just the high-order term and ignoring the constant coefficient, equals $\Theta(n \lg n)$.

d. In practice, $k$ should be the largest list length on which insertion sort is faster than merge sort.

(2) [5 points]

Express the function $n^3/1000 - 100n^2 - 100n + 3$ in terms of $\Theta$-notation

It is $\Theta(n^3)$
3.1.1 (3) [10 points]

Let $f(n)$ and $g(n)$ be asymptotically nonnegative functions. Using the basic definition of $\Theta$-notation, prove that $\max(f(n), g(n)) = \Theta(f(n) + g(n))$.

We need to prove if there exist positive number $c_1$ and $c_2$, let when number $n$ is large enough that:

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

1. to prove $\max(f(n), g(n)) \leq c_2 (f(n) + g(n))$
   
   just let $c_2 = 1$

2. to prove $c_1 (f(n) + g(n)) \leq \max(f(n), g(n))$
   
   Suppose $f(n) \geq g(n)$, let $c_1 = \frac{1}{2}$, thus
   
   $\frac{1}{2} (f(n) + g(n)) \leq \max(f(n), g(n)) = f(n) = \frac{1}{2} (f(n) + f(n))$ is always true

Therefore, there exist $c_1 = 1/2$, $c_2 = 1$, which enables the following formula is true when $n$ is large enough

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

So

$$\max(f(n), g(n)) = \Theta(f(n) + g(n))$$

4.4.3 (4) [10 points]

Use a recursion tree to determine a good asymptotic upper bound on the recurrence $T(n) = 4T(n/2 + 2) + n$. Use the substitution method to verify your answer.

The height of the tree is $\log_2(n)$ and the number of leaves is $4^{\log_2(n)} = n^2$
\[ T(n) = n + 4 \left( \frac{n}{2} + 2 \right) + 4^2 \left( \frac{n}{2^2} + 1 + 2 \right) + \ldots + 4^{\lfloor \log n \rfloor - 1} \left( \frac{n}{2^{\lfloor \log n \rfloor - 1}} + \frac{1}{2^{\lfloor \log n \rfloor - 3}} + \ldots + 1 + 2 \right) \]

\[ = n + 4 \left( \frac{n}{2} + 4 - 2 \right) + 4^2 \left( \frac{n}{2^2} + 4 - 1 \right) + \ldots + 4^{\lfloor \log n \rfloor - 1} \left( \frac{n}{2^{\lfloor \log n \rfloor - 1}} + 4 - \frac{1}{2^{\lfloor \log n \rfloor - 3}} \right) \]

\[ = n + \sum_{k=1}^{\lfloor \log n \rfloor - 1} 4^k \left( \frac{n}{2^k} + 4 - \frac{1}{2^{k-2}} \right) \]

\[ = n + n(n - 2) + \frac{4n^2 - 16}{3} + (4n - 8) \]

\[ = O(n^2) \]

1. Guess the form of the solution.
2. Verify by induction.
3. Solve for constants.

Guess \( T(n) = \Theta(n^2) \)

Assume that \( T(n) \leq cn^2 + bn \), for \( k < n \)

\[ T(n) = 4T \left( \frac{n}{2} \right) + n \leq 4 \left( c \ast \left( \frac{n}{2} + 2 \right)^2 + b \left( \frac{n}{2} + 2 \right) \right) + n \]

\[ = cn^2 + (8c + 2b + 1)n + (16c + 8b) \leq cn^2 + bn \]

Just let \( 8c + 2b + 1 \leq b \) and \( 16c + 8b \leq 0 \)

For example, we let \( c = 1 \) and \( b = -100 \), whenever \( n \geq 1 \)

4.4.4 (5) [10 points]

Use a recursion tree to determine a good asymptotic upper bound on the recurrence \( T(n) = 2T(n - 1) + 1 \). Use the substitution method to verify your answer.
Depth is \( n \) and leaf number is \( 2^{n-1} \)

\[
T(n) = 1 + 2 + \cdots + 2^{n-1} = 2^n - 1 = \Theta(2^n)
\]

Assume that \( T(n) \leq c2^n + bn \), for \( k < n \)

\[
T(n) = 2T(n - 1) + 1 \leq 2(c \cdot 2^{n-1} + b(n - 1)) + 1
\]
\[
= c2^n + (2bn - 2b + 1) \leq c2^n + bn
\]

Just let \( b < 0 \)

For example, we let \( b=-100 \), whenever \( n \) is large enough, \( T(n) \rightarrow O(2^n) \)

4.5.4 (6) [15 points]

**Can the master method be applied to the recurrence \( T(n) = 4T(n/2) + n^2 \log n \)? Why or why not? Give an asymptotic upper bound for this recurrence.**

Please prove the bound. (Clue: try \( \Theta(n^2 \log^2 n) \))

In the given recurrence, \( a=4 \) and \( b=2 \). Hence, \( n^{\log_b a} = n^{\log_4 2} = n^2 \) and \( f(n) = \Theta(n^2 \log n) \).

Now, asymptotically \( f(n) = n^2 \log n \) is definitely larger than \( n^2 \), but it is not polynomially larger than \( n^2 \). So, we cannot apply master method to this recurrence.

Assume that \( T(n) \leq c(n^2 (\log n))^2 \), for \( k < n \)

\[
T(n) = 4T\left(\frac{n}{2}\right) + n^2 \log n \leq 4 \left( c \left( \frac{n}{2} \right)^2 \left( \log \left( \frac{n}{2} \right) \right)^2 \right) + n^2 \log n
\]
\[
\leq cn^2(\log n - 1)^2 + n^2 \log n
\]
\[ \leq cn^2(lgn)^2 + n^2(c + (1 - 2c)lgn) \]

Just let \( c = 1 \), whenever \( n \) is large enough,

\[ T(n) \leq cn^2(lgn)^2 \]

4.1 (6) [30 points]

Give asymptotic upper and lower bounds for \( T(n) \) in each of the following recurrences. Assume that \( T(n) \) is constant for \( n \leq 2 \). Make your bounds as tight as possible, and justify your answers.

a. \( T(n) = 2T(n/2) + n^4 \).

b. \( T(n) = T(7n/10) + n \).

c. \( T(n) = 16T(n/4) + n^2 \).

d. \( T(n) = 7T(n/3) + n^2 \).

e. \( T(n) = 7T(n/2) + n^2 \).

f. \( T(n) = 2T(n/4) + \sqrt{n} \).

a. \( T(n) = 2T(n/2) + n^4 \); \( T(n) = \Theta(n^4) \): in terms of the Master Theorem, \( a = 2, b = 2 \), so \( \log_b a = \log_2 2 = 1 \). \( f(n) = n^4 = n^{\log_b a + 3} \); this is looking like case 3. We need to check \( af\left(\frac{n}{b}\right) \leq cf(n) \) for some \( c < 1 \) and \( n \) large enough. In fact, \( af\left(\frac{n}{b}\right) = 2\left(\frac{n}{2}\right)^4 = \frac{1}{8}n^4 = \frac{1}{8}f(n) \), and case 3 applies. Conclude \( T(n) = \Theta(n^4) \).

b. \( T(n) = T(7n/10) + n \); \( T(n) = \Theta(n) \): in terms of the Master Theorem, \( a = 1, b = \frac{10}{7} \), so \( \log_b a = \log_{10/7} 1 = 0 \). \( f(n) = n = n^{\log_b a + 1} \); this is looking like case 3. We need to check \( af\left(\frac{n}{b}\right) \leq cf(n) \) for some \( c < 1 \) and \( n \) large enough. In fact, \( af\left(\frac{n}{b}\right) = 1\left(\frac{7n}{10}\right)^1 = \frac{7}{10}n = \frac{7}{10}f(n) \), and case 3 applies.
Conclude $T(n) = \Theta(n)$.

c. $T(n) = 16T(n/4) + n^2$: $T(n) = \Theta(n^2 \lg n)$: in terms of the Master Theorem, $a = 16$, $b = 4$, so $\log_b a = \log_4 16 = 2$. $f(n) = n^2 = n^{\log_b a}$; this is case 2.
   Conclude $T(n) = \Theta(n^2 \lg n)$.

d. as $\log_3 7 < 2$, based on master method, as case 3 applies, $T(n) = \Theta(n^2)$

e. $T(n) = 7T(n/2) + n^2$: $T(n) = \Theta(n^{\log_2 7})$: in terms of the Master Theorem, $a = 7$, $b = 2$, so $2 < \log_b a = \log_2 7 < 3$. $f(n) = n^2 = n^{\log_b a - \epsilon}$; this is case 1.
   Conclude $T(n) = \Theta(n^{\log_2 7})$.

f. $T(n) = 2T(n/4) + \sqrt{n}$: $T(n) = \Theta(\sqrt{n} \lg n)$: in terms of the Master Theorem, $a = 2$, $b = 4$, so $\log_b a = \log_4 2 = \frac{1}{2}$. $f(n) = n^{\frac{1}{2}} = n^{\log_b a}$; this is case 2.
   Conclude $T(n) = \Theta(\sqrt{n} \lg n)$. 