Analysis of Algorithms: Solutions 3

```
                              X
                              X
                              X
                              X
number of                     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
      -----------------------------
      1  2  3  4  5  6  7  8  9  10
                 grades
```

Problem 1
Determine asymptotic upper and lower bounds for each of the following recurrences.

(a) $T(n) = 27T(n/3) + n$

$$
\begin{aligned}
T(n) &= n + 27T(\frac{n}{3}) \\
&= n + 27(\frac{n}{3} + 27T(\frac{n}{3^2})) \\
&= n + 27\frac{n}{3} + 27^2T(\frac{n}{3^2}) \\
&= n + 27\frac{n}{3} + 27^2(\frac{n}{3^2} + 27T(\frac{n}{3^3})) \\
&= n + 27\frac{n}{3} + 27^2\frac{n}{3^2} + 27^3T(\frac{n}{3^3}) \\
&\quad \cdots \\
&= n + 27\frac{n}{3} + 27^2\frac{n}{3^2} + 27^3\frac{n}{3^3} + 27^4\frac{n}{3^4} + ... + 27^{\log_3 n}\frac{n}{3^{\log_3 n}} \\
&= n + 9n + 9^2n + 9^3n + 9^4n + ... + 9^{\log_3 n}n \\
&= n(1 + 9 + 9^2 + 9^3 + 9^4 + ... + 9^{\log_3 n}) \\
&= n\frac{9^{\log_3 n+1} - 1}{9 - 1} \\
&= n\frac{9n^2 - 1}{8} \\
&= \Theta(n^3)
\end{aligned}
$$

(b) $T(n) = 27T(n/3) + n^3$

$$\begin{aligned}
T(n) &= n^3 + 27T(\frac{n}{3}) \\
&= n^3 + 27((\frac{n}{3})^3 + 27T(\frac{n}{3^2})) \\
&= n^3 + 27(\frac{n}{3})^3 + 27^2T(\frac{n}{3^2}) \\
&= n^3 + 27(\frac{n}{3})^3 + 27^2((\frac{n}{3^2})^3 + 27T(\frac{n}{3^3})) \\
&= n^3 + 27(\frac{n}{3})^3 + 27^2(\frac{n}{3^2})^3 + 27^3T(\frac{n}{3^3}) \\
&\quad \cdots \\
&= n^3 + 27(\frac{n}{3})^3 + 27^2(\frac{n}{3^2})^3 + 27^3(\frac{n}{3^3})^3 + 27^4(\frac{n}{3^4})^3 + ... + 27^{\log_3 n}(\frac{n}{3^{\log_3 n}})^3 \\
&= \underbrace{n^3 + n^3 + n^3 + n^3 + n^3 + ... + n^3}_{\log_3 n+1} \\
&= n^3(\log_3 n + 1) \\
&= \Theta(n^3 \cdot \lg n)
\end{aligned}$$

(c) $T(n) = 3T(n/9) + \sqrt{n}$

$$\begin{aligned}
T(n) &= \sqrt{n} + 3T(\frac{n}{9}) \\
&= \sqrt{n} + 3(\sqrt{\frac{n}{9}} + 3T(\frac{n}{9^2})) \\
&= \sqrt{n} + 3\sqrt{\frac{n}{9}} + 3^2T(\frac{n}{9^2}) \\
&= \sqrt{n} + 3\sqrt{\frac{n}{9}} + 3^2(\sqrt{\frac{n}{9^2}} + 3T(\frac{n}{9^3})) \\
&= \sqrt{n} + 3\sqrt{\frac{n}{9}} + 3^2\sqrt{\frac{n}{9^2}} + 3^3T(\frac{n}{9^3}) \\
&\quad \cdots \\
&= \sqrt{n} + 3\sqrt{\frac{n}{9}} + 3^2\sqrt{\frac{n}{9^2}} + 3^3\sqrt{\frac{n}{9^3}} + 3^4\sqrt{\frac{n}{9^4}} + ... + 3^{\log_9 n}\sqrt{\frac{n}{9^{\log_9 n}}} \\
&= \underbrace{\sqrt{n} + \sqrt{n} + \sqrt{n} + \sqrt{n} + ... + \sqrt{n}}_{\log_9 n+1} \\
&= \sqrt{n}(\log_9 n + 1) \\
&= \Theta(\sqrt{n} \cdot \lg n)
\end{aligned}$$

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

We "unwind" the recurrence until reaching some constant value of n, say, 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.

$$\begin{aligned}
T(2^{2^k}) &= 1 + T(\sqrt{2^{2^k}}) \\
&= 1 + T(2^{2^{k-1}}) \\
&= 1 + 1 + T(\sqrt{2^{2^{k-1}}}) \\
&= 1 + 1 + T(2^{2^{k-2}}) \\
&= 1 + 1 + 1 + T(\sqrt{2^{2^{k-2}}}) \\
&= 1 + 1 + 1 + T(2^{2^{k-3}}) \\
&\quad \ldots \\
&= \underbrace{1 + 1 + 1 + \ldots + 1}_{k} + T(2) \\
&= k + \Theta(1) \\
&= \Theta(k)
\end{aligned}$$

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

(e) $T(n) = T(n-1) + n^2$

$$\begin{aligned}
T(n) &= n^2 + T(n-1) \\
&= n^2 + ((n-1)^2 + T(n-2)) \\
&= n^2 + (n-1)^2 + T(n-2) \\
&= n^2 + (n-1)^2 + ((n-2)^2 + T(n-3)) \\
&= n^2 + (n-1)^2 + (n-2)^2 + T(n-3) \\
&\quad \ldots \\
&= n^2 + (n-1)^2 + (n-2)^2 + (n-3)^2 + (n-4)^2 + \ldots + 1^2 \\
&= \frac{n(n+1)(2n+1)}{6} \\
&= \Theta(n^3)
\end{aligned}$$

Problem 2
Consider the following sorting algorithm:

Stooge-Sort(A, i, j)
1. **if** $A[i] > A[j]$
2. **then** exchange $A[i] \leftrightarrow A[j]$
3. **if** $i + 1 \geq j$
4. **then return**
5. $k \leftarrow \lfloor (j - i + 1)/3 \rfloor$
6. Stooge-Sort$(A, i, j - k)$ $\quad \triangleright$ first two-thirds
7. Stooge-Sort$(A, i + k, j)$ $\quad \triangleright$ last two-thirds
8. Stooge-Sort$(A, i, j - k)$ $\quad \triangleright$ first two-thirds again

(a) Argue that Stooge-Sort$(A, 1, n)$ correctly sorts the input array $A[1..n]$.

We prove the correctness of the algorithm by induction. Clearly, the algorithm works for one-element and two-element arrays, which provides the induction base. Now suppose that it works for all arrays shorter than $A[i..j]$ and let us show that it also works for $A[i..j]$.

After the execution of Line 6, $A[i..(j - k)]$ is sorted, which means that every element of $A[(i + k)..(j - k)]$ is no smaller than every element of $A[i..(i + k - 1)]$; we write it as $A[(i+k)..(j-k)] \geq A[i..(i+k-1)]$. Thus, $A[(i+k)..j]$ has at least $\mathit{length}(A[(i+k)..(j-k)]) = j - i - 2k + 1$ elements each of which is no smaller than each element of $A[i..(i + k - 1)]$.

After the execution of Line 7, $A[(i + k)..j]$ is sorted, which implies that

(1) $A[(j - k + 1)..j]$ is sorted, and

(2) $A[(j - k + 1)..j] \geq A[(i + k)..(j - k)]$.

Since $A[(i + k)..j]$ has at least $(j - i - 2k + 1)$ elements no smaller than each element of $A[i..(i + k - 1)]$ and $\mathit{length}(A[(j - k + 1)..j]) \leq j - i - 2k + 1$, we conclude that

(3) $A[(j - k + 1)..j] \geq A[i..(i + k - 1)]$.

Putting together (2) and (3), we conclude that

(4) $A[(j - k + 1)..j] \geq A[i..(j - k)]$.

After the execution of Line 8, the array $A[i..(j - k)]$ is sorted. Putting this observation together with (1) and (4), we see that the whole array $A[i..j]$ is sorted.

(b) Give the recurrence for the worst-case running time of STOOGE-SORT and a tight asymptotic (Θ-notation) bound on the worst-case running time.

The algorithm first performs a constant-time computation (Lines 1–5), and then recursively calls itself three times (Lines 6–8), each time on an array whose size is 2/3 of the original array's size. Thus, the recurrence is as follows:

$$T(n) = 3T(\frac{2}{3}n) + \Theta(1).$$

This recurrence describes both the worst-case and best-case running time, since the algorithm's behavior does not depend on the order of elements in the input array. We use the iteration method to solve it:

$$\begin{aligned}
T(n) &= 1 + 3T(\frac{2}{3}n) \\
&= 1 + 3 + 9T(\frac{4}{9}n) \\
&\quad \cdots \\
&= 1 + 3 + 3^2 + ... + 3^{\log_{3/2} n} \\
&= \frac{3^{\log_{3/2} n+1} - 1}{3 - 1} \\
&= \Theta(3^{\log_{3/2} n}) \\
&= \Theta(3^{(\log_3 n)/(\log_3 3/2)}) \\
&= \Theta(n^{1/(\log_3 3/2)}) \\
&= \Theta(n^{2.71}).
\end{aligned}$$

(c) Compare the worst-case running time of STOOGE-SORT with that of INSERTION-SORT and MERGE-SORT. Is it a good algorithm?

STOOGE-SORT is slower than the other sorting algorithms. Even INSERTION-SORT has the complexity $O(n^2)$, which is much better than $\Theta(n^{2.71})$.

Problem 3

The following algorithm inputs a natural number n and returns a natural number m.

```
SLOW-COUNTER(n)
for i ← 1 to n
    do for j ← 1 to n
            do S ← ∅        ▷ make the set S empty
               for k ← 1 to i − 1
                  do S ← S ∪ {A[k, j]}      ▷ add the A[k, j] value to S
               for k ← 1 to j − 1
                  do S ← S ∪ {A[i, k]}      ▷ add the A[i, k] value to S
               A[i, j] ← MAX(S) + 1
m ← A[n, n]
return m
```

Give a much faster algorithm that computes the same value m.

Every element $A[i,j]$ of the resulting matrix is 1 greater than its preceding neighbors $A[i-1,j]$ and $A[i,j-1]$. For example, if $n = 8$, then the matrix is as follows:

1	2	3	4	5	6	7	8
2	3	4	5	6	7	8	9
3	4	5	6	7	8	9	10
4	5	6	7	8	9	10	11
5	6	7	8	9	10	11	12
6	7	8	9	10	11	12	13
7	8	9	10	11	12	13	14
8	9	10	11	12	13	14	15

Thus, m is always $2n-1$, and we may replace SLOW-COUNTER with the following algorithm:

```
FAST-COUNTER(n)
return 2n − 1
```