Analysis of Algorithms: Solutions 3

```
                                                           X
                                                           X
                                                           X
number of                                                  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     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
```

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

Problem 1
Determine asymptotic upper and lower bounds for each of the following recurrences. Make your bounds as tight as possible.

(a) $T(n) = 16T(n/4) + n$.

$$
\begin{aligned}
T(n) &= n + 16T(\frac{n}{4}) \\
&= n + 16(\frac{n}{4} + 16T(\frac{n}{4^2})) \\
&= n + 16\frac{n}{4} + 16^2 T(\frac{n}{4^2}) \\
&= n + 16\frac{n}{4} + 16^2(\frac{n}{4^2} + 16T(\frac{n}{4^3})) \\
&= n + 16\frac{n}{4} + 16^2\frac{n}{4^2} + 16^3 T(\frac{n}{4^3}) \\
&\quad \cdots \\
&= n + 16\frac{n}{4} + 16^2\frac{n}{4^2} + 16^3\frac{n}{4^3} + 16^4\frac{n}{4^4} + \ldots + 16^{\log_4 n}\frac{n}{4^{\log_4 n}} \\
&= n + 4n + 4^2 n + 4^3 n + 4^4 n + \ldots + 4^{\log_4 n} n \\
&= n(1 + 4 + 4^2 + 4^3 + 4^4 + \ldots + 4^{\log_4 n}) \\
&= n\frac{4^{\log_4 n+1} - 1}{4-1} \\
&= n\frac{4n-1}{3} \\
&= \Theta(n^2)
\end{aligned}
$$

(b) $T(n) = 16T(n/4) + n^2$.

$$\begin{aligned}
T(n) &= n^2 + 16T(\frac{n}{4}) \\
&= n^2 + 16((\frac{n}{4})^2 + 16T(\frac{n}{4^2})) \\
&= n^2 + 16(\frac{n}{4})^2 + 16^2T(\frac{n}{4^2}) \\
&= n^2 + 16(\frac{n}{4})^2 + 16^2((\frac{n}{4^2})^2 + 16T(\frac{n}{4^3})) \\
&= n^2 + 16(\frac{n}{4})^2 + 16^2(\frac{n}{4^2})^2 + 16^3T(\frac{n}{4^3}) \\
&\cdots \\
&= n^2 + 16(\frac{n}{4})^2 + 16^2(\frac{n}{4^2})^2 + 16^3(\frac{n}{4^3})^2 + 16^4(\frac{n}{4^4})^2 + ... + 16^{\log_4 n}(\frac{n}{4^{\log_4 n}})^2 \\
&= \underbrace{n^2 + n^2 + n^2 + n^2 + n^2 + ... + n^2}_{\log_4 n+1} \\
&= n^2(\log_4 n + 1) \\
&= \Theta(n^2 \lg n)
\end{aligned}$$

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

$$\begin{aligned}
T(n) &= \sqrt{n} + 2T(\frac{n}{4}) \\
&= \sqrt{n} + 2(\sqrt{\frac{n}{4}} + 2T(\frac{n}{4^2})) \\
&= \sqrt{n} + 2\sqrt{\frac{n}{4}} + 2^2T(\frac{n}{4^2}) \\
&= \sqrt{n} + 2\sqrt{\frac{n}{4}} + 2^2(\sqrt{\frac{n}{4^2}} + 2T(\frac{n}{4^3})) \\
&= \sqrt{n} + 2\sqrt{\frac{n}{4}} + 2^2\sqrt{\frac{n}{4^2}} + 2^3T(\frac{n}{4^3}) \\
&\cdots \\
&= \sqrt{n} + 2\sqrt{\frac{n}{4}} + 2^2\sqrt{\frac{n}{4^2}} + 2^3\sqrt{\frac{n}{4^3}} + 2^4\sqrt{\frac{n}{4^4}} + ... + 2^{\log_4 n}\sqrt{\frac{n}{4^{\log_4 n}}} \\
&= \underbrace{\sqrt{n} + \sqrt{n} + \sqrt{n} + \sqrt{n} + ... + \sqrt{n}}_{\log_4 n+1} \\
&= \sqrt{n}(\log_4 n + 1) \\
&= \Theta(\sqrt{n} \lg n)
\end{aligned}$$

(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 $T(n) = \Theta(\lg\lg n)$.

Problem 2
Consider the following sorting algorithm:

```
STOOGE-SORT(A, i, j)
1. if A[i] > A[j]
2.      then exchange A[i] ↔ A[j]
3. if i + 1 ≥ j
4.      then return
5. k ← ⌊(j − i + 1)/3⌋
6. STOOGE-SORT(A, i, j − k)     ▷ first two-thirds
7. STOOGE-SORT(A, i + k, j)     ▷ last two-thirds
8. STOOGE-SORT(A, i, j − k)     ▷ 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- 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 $length(A[(j-k+1)..j]) \leq j-i-2k+1$, 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:

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

(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

Consider the following algorithm, which inputs a natural number n and returns a natural number m. The algorithm calls a MIN-OUTSIDE(S) procedure which finds the minimal natural number that does not belong to the set S. Give a much faster algorithm that computes the same value m, without using a matrix.

```
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] ← MIN-OUTSIDE(S)      ▷ find the minimal natural number not in S
m ← A[n, n]
return m
```

The main diagonal of the resulting matrix consists of zeros, whereas all other elements of the matrix are greater than zero. For example, if $n = 8$, then the matrix is as follows:

0	1	2	3	4	5	6	7
1	0	3	2	5	4	7	6
2	3	0	1	6	7	4	5
3	2	1	0	7	6	5	4
4	5	6	7	0	1	2	3
5	4	7	6	1	0	3	2
6	7	4	5	2	3	0	1
7	6	5	4	3	2	1	0

Thus, m is always zero, and we may replace SLOW-COUNTER with the following algorithm:

```
FAST-COUNTER(n)
return 0
```