Design and Analysis of Algorithms Test 1

Question 1. (15 points) Suppose that you have an $\Theta(n^4)$ algorithm that required 100 seconds to run on a problem size of 10,000. How long would you expect the algorithm to run on a problem size of 100,000?

\[
\begin{align*}
\Theta(n^4) \Rightarrow T(n) &\approx n^4 = (10^4)^4 = (10^{20})^4 = 10^{80} = 10^{16 	imes 5} = 10^{16} \\
T(100,000) &\approx 100,000^4 = (10^5)^4 = 10^{20} = 10^{\frac{10}{2}} \times 10^{\frac{10}{2}} = 10^6 \text{ sec.}
\end{align*}
\]

(11.6 days)

Question 2. (20 points)

for (i = 1; i < n; i++) {
    j = 1;
    while (j < n) {
        for (k = 1; k <= j; k++) {
            // something that takes $O(1)$
        }
        j = j * 2
    }
}

a) Analyze the above algorithm to determine an obvious big-oh notation, $O()$. $O(n^2 \log_2 n)$

b) Analyze (by tracing) the above algorithm to determine its theta notation, $\Theta()$. 

\[
\begin{array}{cccccc}
\theta & i = 1 & i = 2 & i = 3 & i = 4 \\
\hline
j = 1 & j = 2 & j = 3 & j = 4
\end{array}
\]

$x = (n-1)$

$\text{for } i = 1 \text{ to } n$

$\text{for } j = 1 \text{ to } \frac{n}{\sqrt{2}}$

$\text{if } k = 1$

$\text{while } j < n$

$\text{for } k = 1 \text{ to } \frac{n}{\sqrt{2}}$

$n = (n-1)$

$\text{end for } k$

$\text{end while } j$

$\text{end for } i$

$\text{end for } j$

$\text{end for } i$

$\text{end for } x$

$\text{end for } \theta$
Spring 2011

Question 3. (20 points)

```c
int DoWork(int myArray[], int low, int high) {
    int aThird, oneThird, twoThird;
    if (high < low) {
        return 0;
    } else if (low == high) {
        return myArray[high];
    } else if (low+1 == high) {
        return myArray[low] + myArray[high];
    } else {
        aThird = (high - low + 1) / 3;
        oneThird = low + aThird;
        twoThird = low + 2 * aThird;
        return DoWork(myArray, low, oneThird-1) + DoWork(myArray, oneThird, twoThird-1) + DoWork(myArray, twoThird, high);
    } // end if
} // end DoWork
```

For the above recursive algorithm, write the recurrence relation for the worst-case analysis, including base cases. Let "n" be the number of items in myArray.

\[ W(n) = 3W\left(\frac{n}{3}\right) + 2 \quad \text{for } n \]

\[ W(0) = 0 \]
\[ W(1) = 0 \]
\[ W(2) = 1 \]

Question 4. (20 points) The "Master Theorem" (Theorem B.5) for solving most common recurrences is as follows:

Suppose a complexity function \( T(n) \) is eventually nondecreasing and satisfies the recurrence

\[ T(n) = aT(n/b) + cn^k \quad \text{for } n > 1, n \text{ a power of } b \]

\[ T(1) = d, \]

where constant \( a > 0 \), \( b \) is an integer \( \geq 2 \), \( c > 0 \), \( d \geq 0 \), and \( k \) is an integer \( \geq 0 \). Then

\[ T(n) \in \begin{cases} 
\Theta(n^k) & \text{if } a < b^k \\
\Theta(n^k \log n) & \text{if } a = b^k \\
\Theta(n^{\log_b a}) & \text{if } a > b^k 
\end{cases} \]

Use the Master Theorem to solve both of the following recurrence relations:

a) \( T(n) = 2T\left(\frac{n}{4}\right) + 16n^2 \quad T(n) = 10 \)
\[ 2 < 4^2 \]
\[ \Theta(n^2) \]

b) \( T(n) = 5T\left(\frac{n}{5}\right) + 100 \quad T(n) = 1 \)
\[ 5 \geq 5^0 \]
\[ \Theta(n^{\log_5 5}) = \Theta(n^1) \]
Question 5. (15 points) The general idea of Quick sort is as follows. Assume "n" items to sort.
- Select a "random" item (e.g., the first item) in the unsorted part as the pivot item
- Rearrange (called partitioning) the unsorted items such that:

<table>
<thead>
<tr>
<th>low</th>
<th>Pivot Point</th>
<th>high</th>
</tr>
</thead>
<tbody>
<tr>
<td>All items &lt; Pivot</td>
<td>Pivot Item</td>
<td>All items &gt;= Pivot</td>
</tr>
</tbody>
</table>

- Quick sort the unsorted part to the left of the pivot item
- Quick sort the unsorted part to the right of the pivot item

Explain why Quick sort is $\theta(n^2)$ in the worst-case.

In the worst case, the pivot item is always at the "end" of the range of items after partitioning.

1 + 2 + 3 + \ldots + (n-2) + (n-1)
= n \left( \frac{n-1}{2} \right) \in \Theta(n^2)

(Note: Question 6 is on the next page.)
Question 6. (10 points) Suppose we have nine identical looking coins numbered 1 through 9. Exactly one of the coins is counterfeit and heavier than the others. Suppose further that you have a balance scale and are **allowed only two weightings**.

![Balance Scale Diagram](image)

Develop a divide-and-conquer method for finding the heavier counterfeit coin given these constraints. (Hint: think about splitting the original problem into smaller problem(s) of size n/3)

Assume \( n \) is a power of 3. Split \( n \) coins into 3 piles of \( n/3 \) coins each: Piles A, B, C.

Weigh A vs. B

if weigh A = B then

// pile C has the counterfeit coin
return result of counterfeitcoin A, B in pile C

else if A < B then

// pile B has counterfeit coin
return result of counterfeitcoin A in pile B

else

// pile C has counterfeit coin
return result of counterfeitcoin A, B, C in pile C