1. Use a recursion tree to determine an asymptotically tight bound to the recurrence
   \( T(n) = T(n/3) + T(2n/3) + O(n) \). Use the substitution method to verify your answer.

2. Solve the following recurrences by giving tight \( \Theta \)-notation bounds, using the Master method if possible.
   (a) \( T(n) = T(n/3) + T(2n/3) + 2n \)

   \( T(n) < 2T(2n/3) + 2n \) => Case 1, but could also state that the Master method does not apply.

   (b) \( T(n) = 3T(n/5) + \log^2 n \)

   \( \log_5(3) = 0.68 \) => Case 1 since \( \log^2 n < n^{0.68} \) for \( n > 2^5 \). \( T(n) = \Theta(n^{0.68}) \).

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

   \( \log(7) = 2.8 \) => Case 3. Check that \( 7(n/2)^3 < n^3 \) for \( n > 2^3 \). Yes, so \( T(n) = \Theta(n^3) \).
3. Rank the following functions by order of growth; that is find an arrangement \( g_1, g_2, \ldots, g_8 \) of the functions satisfying \( g_1 = \Omega(g_2), \ g_2 = \Omega(g_3), \ldots, \ g_7 = \Omega(g_8) \). Partition your list into equivalence classes such that \( g_i(n) \) and \( g_j(n) \) are in the same class if and only if \( g_i(n) = \Theta(g_j(n)) \).

(a) \( n/100 \)
(b) \( n^2 \)
(c) \( n \log n \)
(d) \( 2 \)
(e) \( 2^n \)
(f) \( 3^{\log_3 n} \)
(g) \( n^2 - 9000n \)
(h) \( 2^{10n} \)

- (e) Note, this is \( g_1 \).
- (b) and (g)
- (c)
- (a), (f) and (h)
- (d)

4. What is the time complexity (as a function of \( n \)) for the following algorithm? Please give a tight bound. Be sure to explain your answer carefully.

\[
\text{MysteryAlg}(n) \\
x = 0 \\
\text{for } i = 1 \text{ to } n \\
\quad \text{for } j = 1 \text{ to } i \\
\quad \quad x = x * 2 \\
\quad \text{end for} \\
\quad k = 1 \\
\quad \text{while } k <= n^2 \\
\quad \quad x = x - 1 \\
\quad \quad k = k + 1 \\
\quad \text{end while} \\
\text{end for} \\
\text{return } x
\]

\[
T(n) = 1 + n(n+1)/2 + n + 2n^3 = \Theta(n^3)
\]
5. Write a recursion formula for the running time \( T(n) \) of the function \( \text{NoNeed2} \), whose code is below. Prove using the iterative method that \( T(n) = \Theta(n^3) \).

```java
NoNeed2(int n) {
    if (n < 3) return;
    for(i = 1; i < n; i++)
        for(j = 1; j < n; j++)
            print("*");
    NoNeed2(n-3);
}
```

\[
T(n) = T(n-3) + \Theta(n^2)
\]

Guess \( T(n) = cn^3 \).

\[
T(n) = n^2 + c(n-3)^3 = n^2 + c(n^3 + 6n^2 - 18n - 27) > cn^3 \text{ – need an adjustment!}
\]

Guess \( T(n) = cn^3 - bn^2 \). Which will work as long as \( b > 6c+1 \).

6. What is the Big-O of the following code?

```java
int total = 0;
for(int i = 0; i < N; i++)
    for(int j = 0; j < 5; j++)
        for(int k = 0; k < N; k++)
            for(int m = 1; m <= N; m = m * 2)
                total++;
```

\[
T(n) = 5n^3 \lg(n)
\]

7. Fill in the following table. Each box should have a “yes” or “no” indicating whether or not the condition at the top of the row is true.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>( A=O(B) )</th>
<th>( A=\Omega(B) )</th>
<th>( A=\Theta(B) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td>1</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>( n^2 )</td>
<td>( n \lg n )</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>( n+n^2 )</td>
<td>( n^3 )</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>( n )</td>
<td>200n + n/2</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>( n^5 )</td>
<td>( \lg n )</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>( n^{10^{10}} )</td>
<td>( 2^n )</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
8. Show Merge sort on the following array. Carefully explain each step.
3, 4, 8, 1, 6, 7, 2, 5

3,4,8,1  6,7,2,5
3,4  1,8  6,7  2,5
1,3,4,8  2,5,6,7
1,2,3,4,5,6,7,8

9. Show a decision tree for finding the minimum of three numbers $a$, $b$, and $c$.

```
a < b
  / \
/a < c / b < c
 /     /
|      |
|      |
a   c   b   c
```

10. Given the structure of a heap as sketched below, where the second-smallest value in the set is marked with a $2$. Mark a $4$ for each node that can possibly contain the fourth-smallest value in the set. Assume that there are no duplicate node values.

The shaded nodes are the answer. Any of the nodes shown below the 2 could be the 4th smallest element. If the left child of the root is the 3rd smallest element, then either of its children, but not its grandchildren, could be the 4th smallest. If the 3rd smallest is under the 2, then the left child of the root could also be the 4th smallest.