1. For each of the following list of functions, order the functions in increasing order of growth rate. Note when two adjacent functions have the same \( \Theta() \) growth rate.

For example, for the functions \( x^3, 3x + 4, 5x, 3x^3, x^2 \), the ordering would be (functions with same \( \Theta() \) running time are in the same column):

<table>
<thead>
<tr>
<th>Function</th>
<th>Growth Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 5x )</td>
<td>( x^2 )</td>
</tr>
<tr>
<td>( 3x + 4 )</td>
<td>( x^3 )</td>
</tr>
</tbody>
</table>

(a) \( n^2, 3n^2 + \lg n, \lg n, \lg \lg n, (\lg n)^2, n \lg n, n \lg n + n, 5n \)
(b) \( 2^n, n^2, n!, n^n, \sum_{i=1}^{n} i, n^{1.5}, n^{1.00004}, n \lg n \)

2. Give \( \Theta() \) running times for each of the following code fragments. HINT – if you are having trouble with finding \( \Theta() \), first find \( O() \), then \( \Omega() \). For partial credit, just find \( O() \).

(a) for (i=0; i<n; i++)
    for (j=n; j>0; j--)
        for (k=0; k<n; k++)

(b) \( tmp = 0; \)
    while (tmp < n) {
        for (tmp2=0; tmp2<n*n; tmp2++)
            \( \text{sum}++ \);
        \( tmp++ \);}

(c) for (i=1; i<n*n i=i+2)
    for (j=1; j<n*n; j=j+2)
        \( \text{sum}++ \);

(d) for (i=1; i<n*n i=i*2)
    for (j=1; j<n*n; j=j+2)
        \( \text{sum}++ \);

(e) for (j=0; j < n*n; j++)
    for (k=0; k<j*j; k++)
        \( \text{sum}++ \);

(f) \( tmp1 = n; \)
    while (tmp1 > 1) {
        \( tmp2 = 0; \)
        while (tmp2 < n/2) {
            \( tmp2++; \)
        }
        \( tmp1 = tmp1/2 \)
    }

(g) for (tmp1=0; tmp1<n; tmp1++)
    for (tmp2=0; tmp2 < tmp1; tmp2++)
    for (tmp3=1; tmp3<n; tmp3 = tmp3*3)
        \( \text{sum}++ \);

3. For each of the following recursive functions:
• Describe what the function computes (careful, some of these are tricky!)
• Give a recurrence relation that describes the running time of the function (Give both base and recursive cases)
• Solve the recurrence to get a Θ running time for the function. Use either the repeated substitution method, or the recursion tree method (which is essentially the same as the repeated substitution method, just a little more graphical)

(a) int recursive1(int n) {
    if (n > 1)
        return recursive1 (n/2) + recursive1(n/2);
    else
        return n;
}
(b) int recursive2(int n) {
    if (n > 1)
        return recursive2(n-1) + recursive2(n-1);
    else
        return n;
}