**Problem 1.** Sort the following functions in increasing order of asymptotic growth

\[ n, n^3, n \log n, n^n, \frac{3^n}{n^2}, n^!, \frac{n^3}{\log n}, 2^n, \sqrt{n}, \frac{n^3}{2 \log n} \]

**Problem 2.** Sort the following functions in increasing order of asymptotic growth

\[ \frac{1}{n \log n}, \log \log n, \log n^!, 2^{\log \log n} \]

**Problem 3.** Consider the following pseudo-code fragment.

```
1: procedure STARS(n)
2:     for i = 1, ..., n do
3:         print "*" i many times
```

a) Using the $O$-notation, upperbound the running time of $\text{STARS}$.  
b) Using the $\Omega$-notation, lowerbound the running time of $\text{STARS}$ to show that your upperbound is in fact asymptotically tight.

**Problem 4.** Given an array $A$ consisting of $n$ integers $A[0], A[1], \ldots, A[n-1]$, we want to compute the upper triangle matrix

\[ C[i][j] = \frac{A[i] + A[i+1] + \cdots + A[j]}{j-i+1} \]

for $0 \leq i \leq j < n$. Consider the following algorithm for computing $C$:

```
1: function SUMMING-UP((A))
2:     for i = 0, ..., n - 1 do
3:         for j = i, ..., n - 1 do
4:             add up entries $A[i]$ through $A[j]$ and divide by $j - i + 1$
5:             store result in $C[i][j]$
6:     return C
```

a) Using the $O$-notation, upperbound the running time of $\text{SUMMING-UP}$.  
b) Using the $\Omega$-notation, lowerbound the running time of $\text{SUMMING-UP}$. 
Problem 5. Come up with a more efficient algorithm for computing the above matrix $C[i][j] = \sum_{k=i}^{j} A[k]$ for $0 \leq i \leq j < n$. Your algorithm should run in $O(n^2)$ time.

Problem 6. Give a formal proof of the transitivity of the $O$-notation. That is, for function $f$, $g$, and $h$ show that if $f(n) = O(g(n))$ and $g(n) = O(h(n))$ then $f(n) = O(h(n))$.

Problem 7. A priority queue is a data structure that supports two operations: INSERT and GET-MIN. Suppose we have at our disposal two implementations: In the first implementation (based on arrays) INSERT takes $O(1)$ time, and GET-MIN takes $O(n)$ time; in the second implementation (based on heaps) both operations run in $O(\log n)$ time. (Here $n$ the size of the queue.)

You need to analyze an algorithm that performs $X$ many INSERT operations followed by $Y$ many GET-MIN operations. Assume we only care about the time spent on priority queue operations. When does it make sense to switch from a heap-based implementation to a an array-based implementation?