Problem 1

Sort the following functions in increasing order of asymptotic growth

$$n, n^3, n\log n, \frac{n}{\log n}, \frac{n}{\log^2 n}, \sqrt{n}, \sqrt{n^3}$$

Problem 2

Sort the following function in decreasing order of asymptotic growth

$$n^{1.5}, 2^n, \frac{n}{2^n}, \frac{2^n}{n^{10}}, n!, 1.5^n, 2^{\log n}$$

Problem 3

Which of the following is largest asymptotically

$$\log_3 n, \log_2 n, \log_{10} n$$

Problem 4

Prove that the sum $S(n)$ of the first n natural numbers is $n(n+1)/2$.

Problem 5

Imagine a program A running with time complexity $\Theta(f(n))$, taking t seconds for an input of size m. What would your estimation be for the execution time for an input of size $2m$ for the following functions: n, $n\log n$, n^2 or n^3.

Problem 6

1. For each of the following pseudo-code fragments. give an upper bound for their running time using the big-Oh notation.
2. The upper bound captures the worst case running time of an algorithm. Some instances might require the algorithm to perform more steps than others. Some instances might allow the algorithm to terminate faster. A lower bound for the running time is a "best-case" running time in the worst-case. If an algorithm is $\Omega(f(n))$ for example, then there exists an instance that will take *at least* $f(n)$ steps.

   For the second algorithm shown below, give a lower bound for the running time.

Algorithm 1 Stars

```
1: for $i = 1, \ldots, n$ do
2:     print "*" $i$ times
3: end for
```

Algorithm 2 Check Numbers

```
procedure CheckNumbers($A$,$B$)
                                        ▷ $A$ and $B$ are two lists of integers
    count = 0
    for $i = 1, \ldots, n$ do
        for $j = i \ldots m$ do
            if $A[i] \geq B[j]$ then
                count = count +1
                break
            end if
        end for
    end for
end procedure
```

Problem 7
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:

Algorithm 3 summing-up

```
function summing-up(($A$))
    for $i = 0, \ldots, n-1$ do
        for $j = i, \ldots, n-1$ do
            add up entries $A[i]$ through $A[j]$ and divide by $j - i + 1$
            store result in $C[i][j]$
        end for
    end for
    return $C$
end function
```

1. Using the O-notation, upperbound the running time of summing-up.
2. Using the Ω-notation, lowerbound the running time of summing-up.