Problem 1

Sort the following functions in increasing order of asymptotic growth

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

Solution: Answer should be:

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

Problem 2

Sort the following function in decreasing order of asymptotic growth

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

Solution: Answer should be:

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

Problem 3

Which of the following is largest asymptotically

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

Solution: They are all equal. This is because:

$$\log_b a = \frac{\log_d a}{\log_d b}$$

Problem 4

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

Solution: Proof is by induction on n. Base case: If $n = 1$ then the claim is trivially true.
Induction hypothesis: Assume that the sum of the first n natural is $n(n+1)/2$.
Induction step: Next we want to prove the statement for $n+1$. From the definition of $S(n)$ we know $S(n+1) = S(n)+n+1$, and from the induction hypothesis we know $S(n) = n(n+1)/2$, and therefore $S(n+1) = S(n) + n + 1 = n(n+1)/2 + n + 1 = (n+2)(n+1)/2$.

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.

Solution: $n \to 2m = 2t$ seconds
$n \log n \to 2m \log 2m = 2t(1 + \log 2/\log m)$ seconds
$n^2 \to 4m^2 = 4t$ seconds
$n^3 \to 8m^3 = 8t$ seconds

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, ..., n do
2:     print "*" i times
3: end for
```

Solution: first interation prints 1 star, second prints two, third prints 3 and so on. Total number of stars is

$$1 + 2 + \cdots + n \sum_{j=1}^{n} j = \frac{n(n+1)}{2} = O(n^2)$$

Algorithm 2 CHECK NUMBERS

```
1: procedure CheckNumbers(A,B)
                                  ▷ A and B are two lists of integers
2:     count = 0
3:     for i = 1, ..., n do
4:         for j = i ... m do
5:             if A[i] ≥ B[j] then
6:                 count = count +1
7:                 break
8:             end if
9:         end for
10:    end for
11: end procedure
```

Solution: An upper bound is $O(n \cdot m)$. The lower bound is $\Omega(n \cdot m)$. Ask the students to come up with a specific example input for this (which is easy).

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

```
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:         end for
7:     end for
8:     return C
9: 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.

Solution:

1. The number of iterations is $n + n - 1 + \cdots + 1 = \binom{n}{2}$, which is bounded by n^2. In the iteration corresponding to indices (i, j) we need to scan $j - i + 1$ entries from A, so it takes $O(j - i + 1) = O(n)$. Thus, the overall time is $O(n^3)$.
2. In an implementation of this algorithm, Line 3 would be computed with a for loop; when $i < \frac{1}{4}n$ and $j > \frac{3}{4}n$, this loop would iterate least $n/2$ times, which takes $\Omega(n)$ time. There are $n^2/16$ pairs (i, j) of this kind, which is $\Omega(n^2)$. Thus, the overall time is $\Omega(n^3)$.