1. Problem Session 1

For an overview of standard mathematical functions such polynomials, exponentials, logarithms and factorials, and of the relationships between these functions, you may consult Section 3.2 of CLRS. Below, as in CLRS, ln refers to the natural logarithm with base e.

1.1. Suppose we have a computer which can perform 1 million ($= 10^6$) operations per second. The seven formulas below denote the running time of some algorithms (measured in number of operations) depending on the number of elements n we feed to the algorithm. Determine for each algorithm how many elements can be processed in 1 minute.

a) n^2 *b)* $n \ln n$ *c)* 2^n *d)* $n\sqrt{n}$ *e)* n^{100} *f)* 4^n *g)* n

1.2. In each of the following situations, indicate whether $f = \mathcal{O}(g)$, or $f = \Omega(g)$, or both (in which case $f = \Theta(g)$). No proof is required.

a) $f(n) = n - 100$ and $g(n) = n - 200$
b) $f(n) = 14n^2$ and $g(n) = n^2$
c) $f(n) = n^3 - 5n^2 + 3n + 27$ and $g(n) = 2n^3$
d) $f(n) = 3^n$ and $g(n) = n^5$
e) $f(n) = n^{\frac{1}{2}}$ and $g(n) = n^{\frac{2}{3}}$
f) $f(n) = 100n + \ln n$ and $g(n) = n + (\ln n)^2$
g) $f(n) = n \ln n$ and $g(n) = 10n \ln 10n$
h) $f(n) = 2^n$ and $g(n) = n!$

1.3. Recall that in order to prove $f \in \mathcal{O}(g)$ one has to choose $c > 0$ and n_0 and then prove that $f(n) \leq cg(n)$ for all $n \geq n_0$.

Prove that $n \in O(2^n)$. (Hint: use $c = 1$ and $n_0 = 1$, then prove the inequality with induction.)

1.4. Consider the following algorithm.

```
search(int v[], int n, int x) {
    int i = 0
    bool found = false
    while (i < n && !found) {
        if (v[i] == x) {
            found = true
        }
        i++
    }
    if (found) return i-1;
    else return -1;
}
```

a) What is the worst case scenario? How many operations does it take in this case (your answer should depend on n)?
b) What is the best case scenario? How many operations does it take in this case (your answer should depend on n)?

1.5. Consider the following algorithm.

```
naive_sort(int v[], int n) {
    for (int e = n; e > 0; e--) {
        for (int i = 0; i < e-1; i++) {
            if (v[i] > v[i+1]) {
                swap(v[i], v[i+1])
            }
        }
    }
}
```

a) What is the worst case scenario? How many operations does it take in this case (depending on n)?
b) What is the best case scenario? How many operations does it take in this case (depending on n)?
c) Can we improve the number of operations in the best case by changing the algorithm a bit? How? How much do we improve?