# Homework 1

Kyle Krause

January 15, 2025

1. LaTeX **Snippets**

   (a) LaTeX **for Dummies**
   For (non-negative) functions $f(n)$ and $g(n)$, we say "$f(n) = O(g(n))$" if there exist positive constants $c, n_0$ such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$. The key features here are:

   - "there exists a positive constant $c \ldots f(n) \leq c \cdot g(n)$": this says that $f(n)$ is *upper bounded* by some constant multiple of $g(n)$. That is, $O$-notation "hides," or ignores, constant factors. Note also that it captures only an *upper bound*: $f(n)$ might actually be much smaller than $c \cdot g(n)$, or not. A big-$O$ bound by itself says nothing about which is the case.
   - "there exists a positive constant $n_0 \ldots$ for all $n \geq n_0$": this says that the upper bound holds for all $n$ above some constant *"threshold."* However, it says nothing about the relationship between $f(n)$ and $g(n)$ below the threshold; it could be that $f(n)$ greatly exceeds (even a huge multiple of) $g(n)$ in that range. Moreover, the threshold $n_0$ can be *any constant*—it could be one, or ten, or a billion, or a googolplex—and the notation hides its exact value.

   (b) **Fibonacci Numbers.**
   The Fibonacci numbers may be defined by the recurrence relation

$$F_n = F_{n-1} + F_{n-2}$$

   with base cases $F_0 = 0$ and $F_1 = 1$. We can compute the $n$-th Fibonacci number using the following recursive algorithm:

```
Require: a natural number $n$
Ensure: the $n$-th Fibonacci number
function FIB($n$)
    if $n = 0$ then
        return 0
    if $n = 1$ then
        return 1
    return Fib($n-1$) + Fib($n-2$)
```

**Source Code:**

```
\begin{enumerate}
    \item \LaTeX \textbf{ for Dummies} \\
    For (non-negative) functions $f(n)$ and $g(n)$, we say ``$f(n) = O(g(n))$'' if ther

    \begin{itemize}
        \item ``there exists a positive constant $c \dots f(n) \leq c \cdot g(n)$'': th
        \item ``there exists a positive constant $n_0 \dots$ for all $n \geq n_0$'': th
    \end{itemize}



    \item \textbf{Fibonacci Numbers.} \\
    The Fibonacci numbers may be defined by the recurrence relation
    \[
    F_n = F_{n-1} + F_{n-2}
    \]
    with base cases $F_0 = 0$ and $F_1 = 1$. We can compute the $n$-th Fibonacci number

    \begin{minipage}{\linewidth}
    \begin{algorithm}[H]
        \begin{algorithmic}[1]
            \Require{a natural number $n$}
            \Ensure{the $n$-th Fibonacci number}
            \Function{\text{Fib}}{$n$}
                \If{$n = 0$}
                    \State \Return $0$
                \EndIf
                \If{$n = 1$}
                    \State \Return $1$
                \EndIf
                \State \Return $\text{Fib}(n-1) + \text{Fib}(n-2)$
            \EndFunction
        \end{algorithmic}
    \end{algorithm}
    \end{minipage}
\end{enumerate}
```

2. **Read the syllabus!**

   (a) Suppose you attend 9 out of 14 discussion sessions and submit both course evaluation receipts.

| **Component** | **Weight (%)** |
|---|---|
| Homework | 40 |
| Exams | 59 |
| Discussion Attendance | 0 |
| Course Evaluations | 1 |
| **Total** | **100** |

   (b) Homework calculation:
   With homework 12 submitted late, it's true score is $90 \times 95\% = 85.5\%$
   With scores of 100, 80, 90, 0, 95, 70, 85, 88, 92, 50, 94, and 85.5, we drop the two lowest scores of 0 and 70.
   The average of the remaining scores is:

$$\frac{100 + 80 + 90 + 95 + 85 + 88 + 92 + 50 + 94 + 85.5}{10} = 85.95\%$$

   (c) Questions
   Three ways I can ask a question about course content are:

   - During lecture/office hours
   - On Piazza
   - Contacting instructors using individual email addresses

   I am most likely to ask questions on Piazza

3. **Welcome to EECS 376!**

   (a) The function Welcome$(n, k)$ prints out "Welcome to EECS 376!" $n-k$ times, $k$ times. Thus, the number of printed statements can be represented by the function $f(n, k) = (n - k) \times k = -k^2 + nk$. This is a simple quadratic function, with a maximum at $k = \frac{n}{2}$. Thus, the function will print the most statements when $k = \frac{n}{2}$.

   (b) Because the number of printed statements is maximized when $k = \frac{n}{2}$, the worst case performance of this function is when $k = \frac{n}{2}$. In this case, due to the $-k^2$ in the function $f(n, k) = -k^2 + nk$, Welcome() will print $\frac{n}{2} \times \frac{n}{2} = \frac{n^2}{4}$ statements. Thus, the worst case performance of this function is $O(n^2)$.

   (c) This algorithm *is* efficient, because its worst case performance is $O(n^2)$, which is polynomial time, satisfying the requirements for an efficient function.

(d) Regardless of the representative base of numbers $n$ and $k$, the quadratic nature of the function $f(n,k) = -k^2 + nk$ will hold. Note that the change of base formula indicates that logarithms in different bases only differ by a constant factor (which is ignored by big-$O$ notation). Thus, the function will still have a worst case performance of $O(n^2)$.

4. **Lumpy array.**

(a) The number of addition operations for the $Sum(A)$ function is directly proportional to the number of elements in the array $A$. The function iterates through each element of the array and adds it to a running total only once. Thus, the worst case runtime for $Sum(A)$ is $O(n)$. This makes $Sum(A)$ an efficient function, as it runs in polynomial time with respect to n, the number of elements in its input array.

(b) If the addition operation of $Sum(A)$ runs in linear $O(k)$ time, and the number of addition operations for the $Sum(A)$ function is $O(n)$, then the total runtime of $Sum(A)$ is $O(nk)$. This is because the number of addition operations is directly proportional to the number of elements in the array, and the time it takes to perform each addition operation is directly proportional to the size of the numbers being added. This worst-case runtime of $O(nk)$ is still polynomial time with respect to the number of elements in the array, making $Sum(A)$ an efficient function.

5. **Prof. Upsilon and their oopsie claims.**

(a) The claim that "We can upper bound the runtime of $Y$ by $O(n^2)$" is true. As $n$ approaches infinity, $n \log n$ grows asymptotically slower than $n^2$. Observe that $n \log n \leq n^2 \Rightarrow \log n \leq n$ for all $n > 1$.

(b) The claim that "On every input, $Y$ runs faster than $X$." is not necessarily true. This is because with a runtime of $O(n \log n)$, for small enough values of $n$, $n \log n > n^2$. Thus, it is not necessarily true that "On every input, $Y$ runs faster than $X$."

(c) The claim that "For all large enough $n$, there is an input of size $n$ for which $Y$ runs faster than $X$." is true. As $n$ approaches infinity, $n \log n$ grows asymptotically slower than $n^2$. Specifically, observe that $\lim_{n\to\infty} \frac{n \log n}{n^2} = \lim_{n\to\infty} \frac{\log n}{n} = 0$. Thus, for sufficiently large $n$, there will always be inputs where $Y$ runs faster than $X$.

(d) The claim that "For all large enough $n$, there is an input of size $n$ for which $Z$ runs faster than $Y$." is true. As $n$ approaches infinity, $n$ grows asymptotically slower than $n \log n$. Specifically, observe that $\lim_{n\to\infty} \frac{n}{n \log n} = \lim_{n\to\infty} \frac{1}{\log n} = 0$. Thus, for sufficiently large $n$, $Z$ will always run faster than $Y$.

6. **Set of proofs by contra-.**

   (a) If $A \subseteq B$ and $B \subseteq C$, then $A \subseteq C$.
   **Proof:** Assume, for the sake of contradiction, that $A \subseteq B$ and $B \subseteq C$, but $A \nsubseteq C$.

      i. By $A \nsubseteq C$, there exists an element $x \in A$ such that $x \notin C$.
      ii. Since $A \subseteq B$, it must be true that $x \in B$.
      iii. However, since $B \subseteq C$, every element of $B$ must also belong to $C$. Thus, $x \in C$, which contradicts $x \notin C$.
      iv. Therefore, the assumption that $A \nsubseteq C$ is false.

   Thus, $A \subseteq C$, and the transitive property of the subset relation holds.

   (b) DeMorgan's Law states: If $x \in A \cap \overline{B}$, then $x \in \overline{A} \cup \overline{B}$.
   Contrapositive: If $x \notin \overline{A} \cup \overline{B}$, then $x \notin A \cap \overline{B}$.
   **Proof:**

      i. If $x \notin \overline{A} \cup \overline{B}$, then $x \notin \overline{A}$ and $x \notin \overline{B}$.
      ii. From $x \notin \overline{A}$, it follows that $x \in A$. From $x \notin \overline{B}$, it follows that $x \in B$.
      iii. For $x \in A \cap \overline{B}$, $x$ must belong to $A$ and $x \notin B$. However, since $x \in B$, $x \notin A \cap \overline{B}$.
      iv. Thus, if $x \notin \overline{A} \cup \overline{B}$, then $x \notin A \cap \overline{B}$.

7. **Euclid's algorithm, extended.**

   (a) $a = b'$, $b = a' - q \cdot b'$
   **g=gcd(x, y)**
   We havent changed the gcd calculation invariant of the Euclidean algorithm. We have only changed the way we calculate the new values of $a$ and $b$, thus, $g = gcd(x, y)$
   **ax+by=g**
   If y=0, then $g = gcd(x, 0) = x$, which satisfies the equation $ax+by = g$
   Assuming this is true for recursive $ay + br = g$, we can substitute $a = a'$, $b = b'$, and $r = x - qy$ to get:

$$a'y + b'(x - qy) = g$$
$$a'y + b'x - b'qy = g$$
$$b'x + (a' - qb')y = g$$

   Thus, we can set $a = b', b = a' - qb'$, and the equation $ax + by = g$ will still hold.

(b) Running by hand

| | input | | division | | rec ans | | output | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $i$ | $x$ | $y$ | $q$ | $r$ | $a'$ | $b'$ | $a$ | $b$ | $s$ | $s_{i+1}/s_i$ | $\leq 2/3$ ? |
| 0 | 376 | 281 | 1 | 95 | -24 | 71 | 71 | -95 | 657 | 376/657 | Y |
| 0 | 281 | 95 | 2 | 91 | 23 | -24 | -24 | 71 | 376 | 186/376 | Y |
| 0 | 95 | 91 | 1 | 4 | -1 | 23 | 23 | -24 | 186 | 95/186 | Y |
| 0 | 91 | 4 | 22 | 3 | 1 | -1 | -1 | 23 | 95 | 7/95 | Y |
| 0 | 4 | 3 | 1 | 1 | 0 | 1 | 1 | -1 | 7 | 4/7 | Y |
| 0 | 3 | 1 | 3 | 0 | 1 | 0 | 0 | 1 | 4 | 1/4 | Y |
| 0 | 1 | 0 | 0 | 0 | – | – | 1 | 0 | 1 | – | – |

The GCD of 281 and 376 is 1. $ax + by = 376 * 71 + (281 * -95) = 1$.
The output (of a=71 and b=-95) is correct.