Section I: Introduction to Algorithms

What is an algorithm?

An algorithm is a well defined procedure (or set of steps) that takes some input and produces some output.

Key elements of an algorithm:

An algorithm must be well defined. In other words, none of the steps that make up an algorithm is ambiguous.

For Example:

A correct unambiguous algorithm: \( a, b \in \mathbb{R} \)

\[
\begin{align*}
    &\text{if } a = b \\
    &\quad \text{print “Hello World”} \\
    &\text{end}
\end{align*}
\]

An incorrect ambiguous algorithm: \( a, b \in \mathbb{R} \)

\[
\begin{align*}
    &\text{if } a \approx b \\
    &\quad \text{print “Hello World”} \\
    &\text{end}
\end{align*}
\]

In the above example the unambiguous set of instructions provides an exact set of parameters, if \( a \) is equal to \( b \) then “Hello World” will print.

In I500 we will be concerned with the following aspects of algorithms:

1) Algorithm correctness
2) Time Efficiency
3) Space Efficiency (memory requirements)

Frequently asked questions concerning the aforementioned points of interest:

Q1: Can we design a faster algorithm?

Q2: Can we prove correctness?

Q3: Can we prove that no further gains in speed are possible?

There are two approaches available to us with which to address these questions.
Approach #1:
1) Code the algorithm.
2) Run the code.
3) Compare across the proposed algorithms.

Approach #2:
1) Develop the algorithm.
2) Prove the correctness of the algorithm.
3) Estimate the runtime (in machine independent steps) without implementing the algorithm.

Section II: RAM Model

RAM stands for Random Access Machine, this model forms the assumptions on which we build our algorithms and can be understood to be a simplified model of real world computers.

Assumptions of the RAM model:
1) Instructions are executed one at a time.
2) Instructions are grouped into
   a. Arithmetic
   b. Data Management
   c. Control
3) Each instruction takes constant time to run
4) There is a limit on word size to \( c \log_2 n \) bits, where \( n \) is the size of the input and \( c \geq 1 \) is constant.

Section III: The Sorting Problem

Algorithms are used to solve problems; here we are examining the problem of sorting a series of numbers.

Q: How do we know if an algorithm solves our problem?
A: If the algorithm produces the correct solution or desired outcome for any input then it solves the problem.

IMPORTANT NOTE: There may be many algorithms that are capable of solving a particular problem and their time and space complexity may significantly vary.

The sorting problem can be stated via defining an input and output of the algorithm that solves it.

Input: a sequence of \( n \) numbers \( a = (a_1, a_2 \ldots a_n) \)

Output: a permutation of \( a \), called \( a' \), where \( a' = (a_1', a_2' \ldots a_n') \) such that \( a_1' \leq a_2' \leq \ldots \leq a_n' \)

A possible solution is an approach called “Insertion Sort,” the algorithm is as follows.
Insertion Sort

insertion-sort
for \( j \leftarrow 2 \) to \( n \)
    \( \text{key} \leftarrow a(j) \)
    \( i \leftarrow j - 1 \)
    while \( i > \text{key} \) and \( a(i) > \text{key} \)
        \( a(i + 1) \leftarrow a(i) \)
        \( i \leftarrow i - 1 \)
    end
    \( a(i + 1) \leftarrow \text{key} \)
end

Let’s apply the algorithm:
Here is the set of numbers to be sorted:

\[
\begin{array}{cccccc}
7 & 10 & 4 & 3 & 1 & 8 \\
\end{array}
\]

key = 10  \( j = 2 \)

\[
\begin{array}{cccccc}
7 & 10 & 4 & 3 & 1 & 8 \\
\end{array}
\]

i

key = 4  \( j = 3 \)

\[
\begin{array}{cccccc}
7 & 10 & 4 & 3 & 1 & 8 \\
\end{array}
\]

i i

key = 3  \( j = 4 \)

\[
\begin{array}{cccccc}
4 & 7 & 10 & 3 & 1 & 8 \\
\end{array}
\]

i
We have seen the algorithm in action but what is the criterion for determining the correctness of the algorithm?

For algorithms containing loops (for loop in this case) it is good to define a loop invariant – a condition that holds true in every run of the loop (for every \( j \) in the beginning of the for loop).

Loop invariant for insertion sort: array \( a(1) \) till \( a(j - 1) \) is sorted for every \( j \)

This loop invariant can now be tested for correctness before the for loop, within, and after it. This brings three conditions for proving the correctness of the algorithm:

1) initialization
2) maintenance
3) termination

The algorithm generated for the sorting problem fulfills each of these criteria.