# COMP10020
# Introduction to Programming II
# **Designing Algorithms**

Dr. Brian Mac Namee
brian.macnamee@ucd.ie
School of Computer Science
University College Dublin

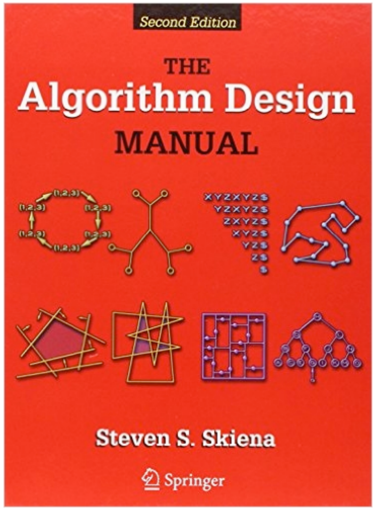
# WHAT IS AN ALGORITHM?

# What Is An Algorithm?

*"An algorithm is a procedure to accomplish a specific task. It is the idea behind any computer program."*

The Algorithm Design Manual

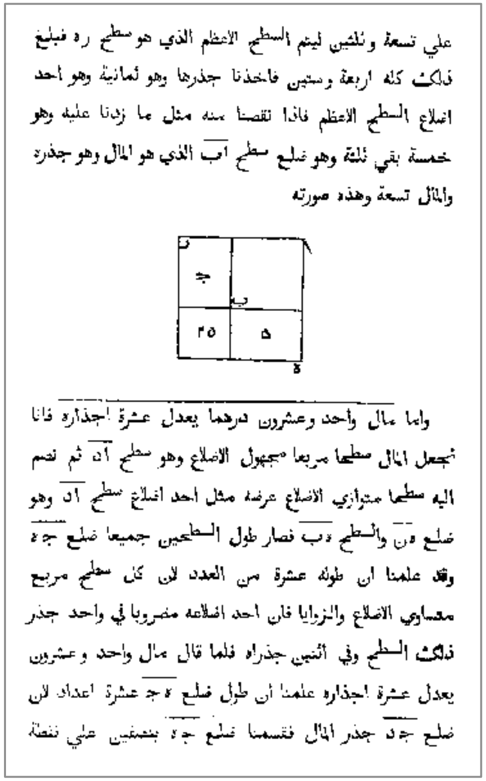
Steven Skiena



https://www8.cs.umu.se/kurser/TDBA77/VT06/algorithms/BOOK/BOOK.HTM
http://www.amazon.com/Algorithm-Design-Manual-Steven-Skiena/dp/1849967202/

# Historical Note

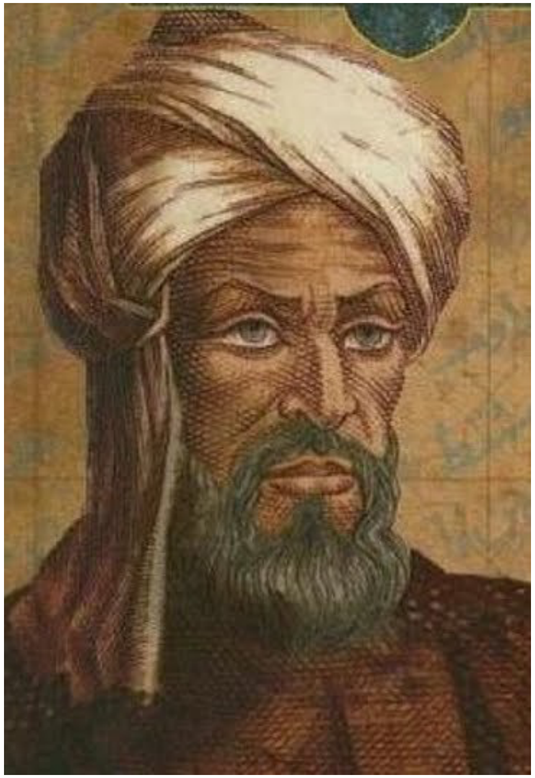
The word *algorithm* is a distortion of al-Khwārizmī, a Persian mathematician who wrote an influential treatise about algebraic methods



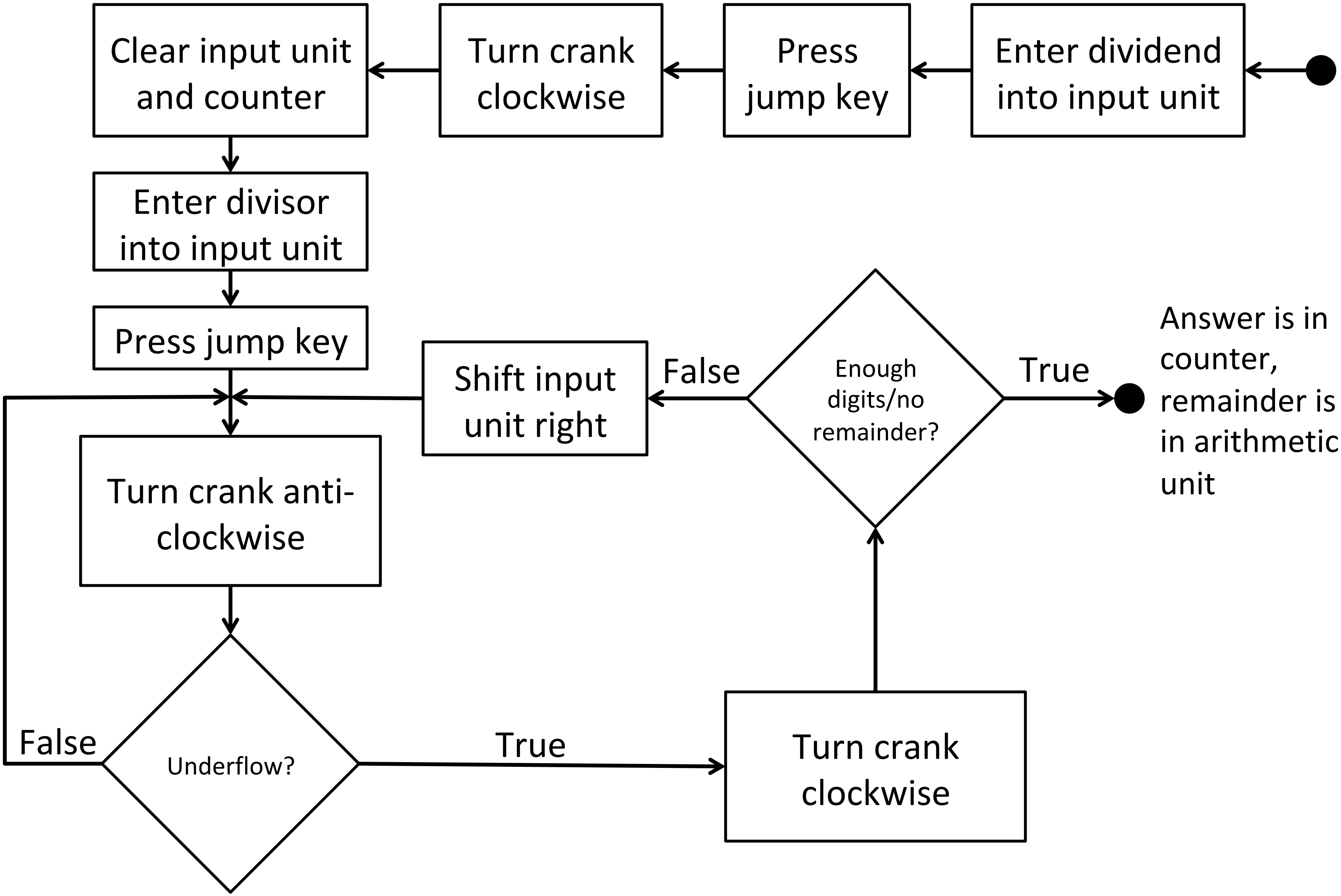
على تسعة وثلثين ليتم السطح الاعظم الذي هو سطح ره فبلغ
ذلك كله اربعة وستين فاخذنا جذرها وهو ثمانية وهو احد
اضلاع السطح الاعظم فاذا نقصنا منه مثل ما زدنا عليه وهو
خمسة بقي ثلثة وهو ضلع سطح اب الذي هو المال وهو جذره
والمال تسعة وهذه صورته

واما مال واحد وعشرون درهما يعدل عشرة اجذاره فانا
نجعل المال سطحا مربعا مجهول الاضلاع وهو سطح اد ثم نضم
اليه سطحا متوازي الاضلاع عرضه مثل احد اضلاع سطح اد وهو
ضلع هن والسطح هب فصار طول السطحين جميعا ضلع جه
وقد علمنا ان طوله عشرة من العدد لان كل سطح مربع
متساوي الاضلاع والزوايا فان احد اضلاعه مضروبا في واحد جذر
ذلك السطح وفي اثنين جذراه فلما قال مال واحد وعشرون
يعدل عشرة اجذاره علمنا ان طول ضلع هج عشرة اعداد لان
ضلع جد جذر المال فقسمنا ضلع جه بنصفين على نقطة

كتاب الخوارزمي
انشاء والمصنف الشيخ الاجل ابي عبد الله
محمد بن موسى الخوارزمي رضي الله عنه وانابه ورحمة
فيه لا ستر ذنوبه وخطاياه العبد الفقير
الى الله الغني به خطاب بن محمد بن علي
ابن جنيس بن علي بن محمد بن علي بن احمد بن
جعفر بن الحسن بن يحيى بن ابرهيم بن محمد بن
ابرهيم بن احمد بن المغيرة بن عمران بن عاصم بن
الوليد بن عتبة بن ربيعة بن عبد شمس بن
عبد مناف
نفعه الله بالعلم والعمل
الصالح
وحسبنا الله ونعم الوكيل



# Division



Enter dividend into input unit

Press jump key

Turn crank clockwise

Clear input unit and counter

Enter divisor into input unit

Press jump key

Turn crank anti-clockwise

Underflow?

Turn crank clockwise

Enough digits/no remainder?

Shift input unit right

Answer is in counter, remainder is in arithmetic unit

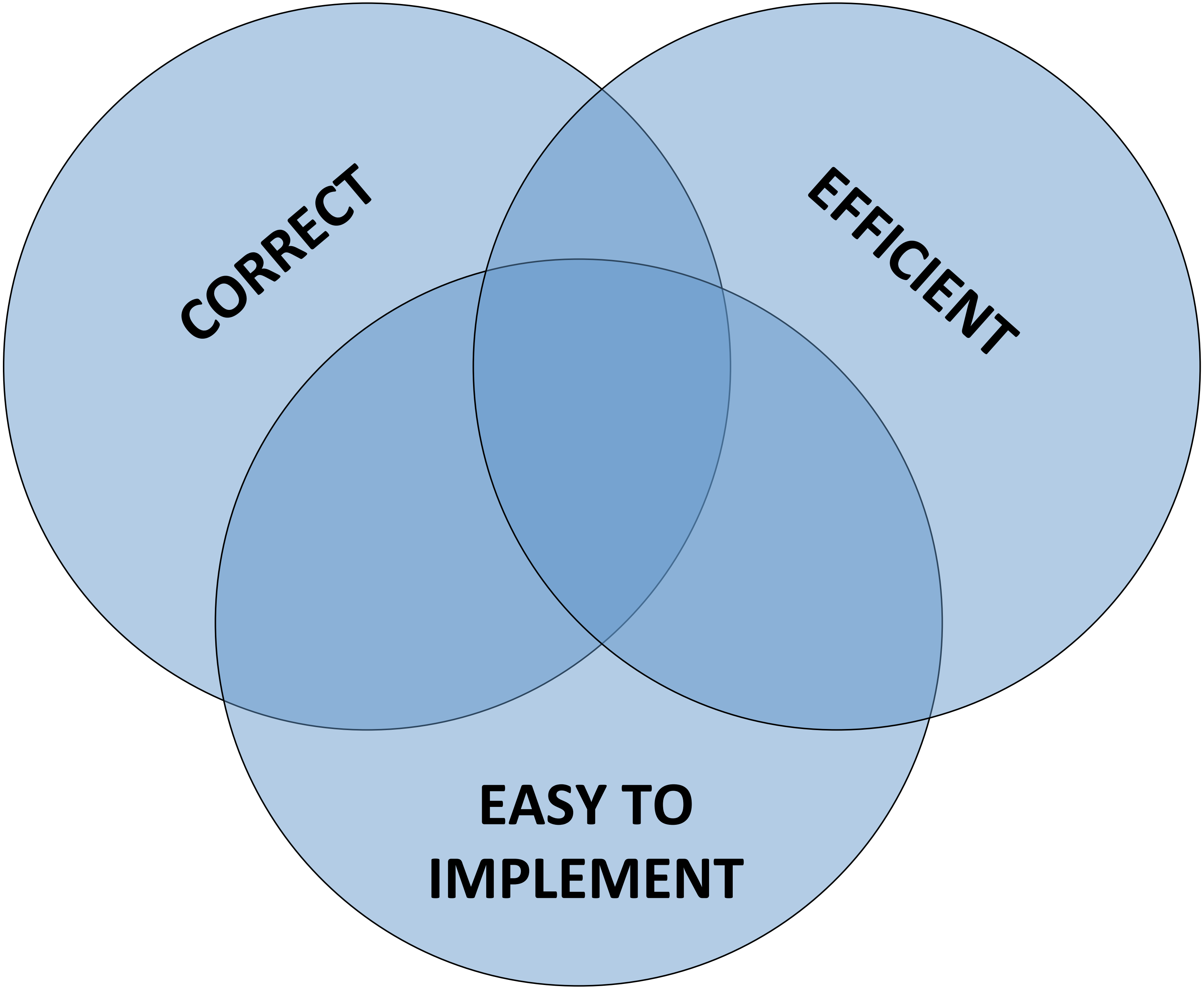
Prof. Dr.-Ing. Christian-M. Hamann Online Museum http://public.beuth-hochschule.de/hamann/calc/index.html

# Characteristics of a Good Algorithm?

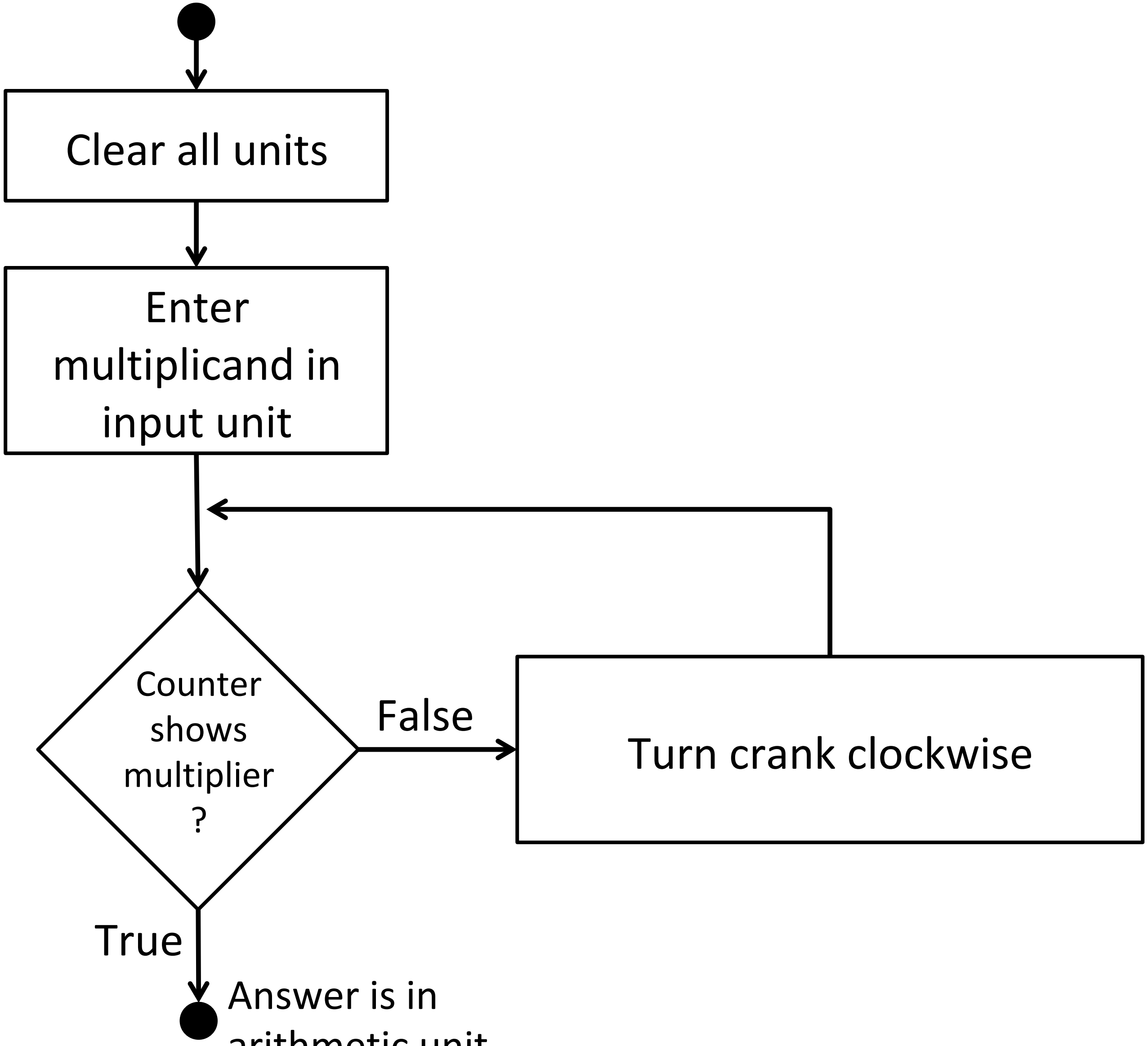
We strive for algorithms that are:

- correct
- efficient
- easy to implement

All three goals are desirable, but they may not be simultaneously achievable - often one or more of them are ignored


CORRECT
EFFICIENT
EASY TO IMPLEMENT

# Multiplication



Clear all units

Enter multiplicand in input unit

Counter shows multiplier ?

False

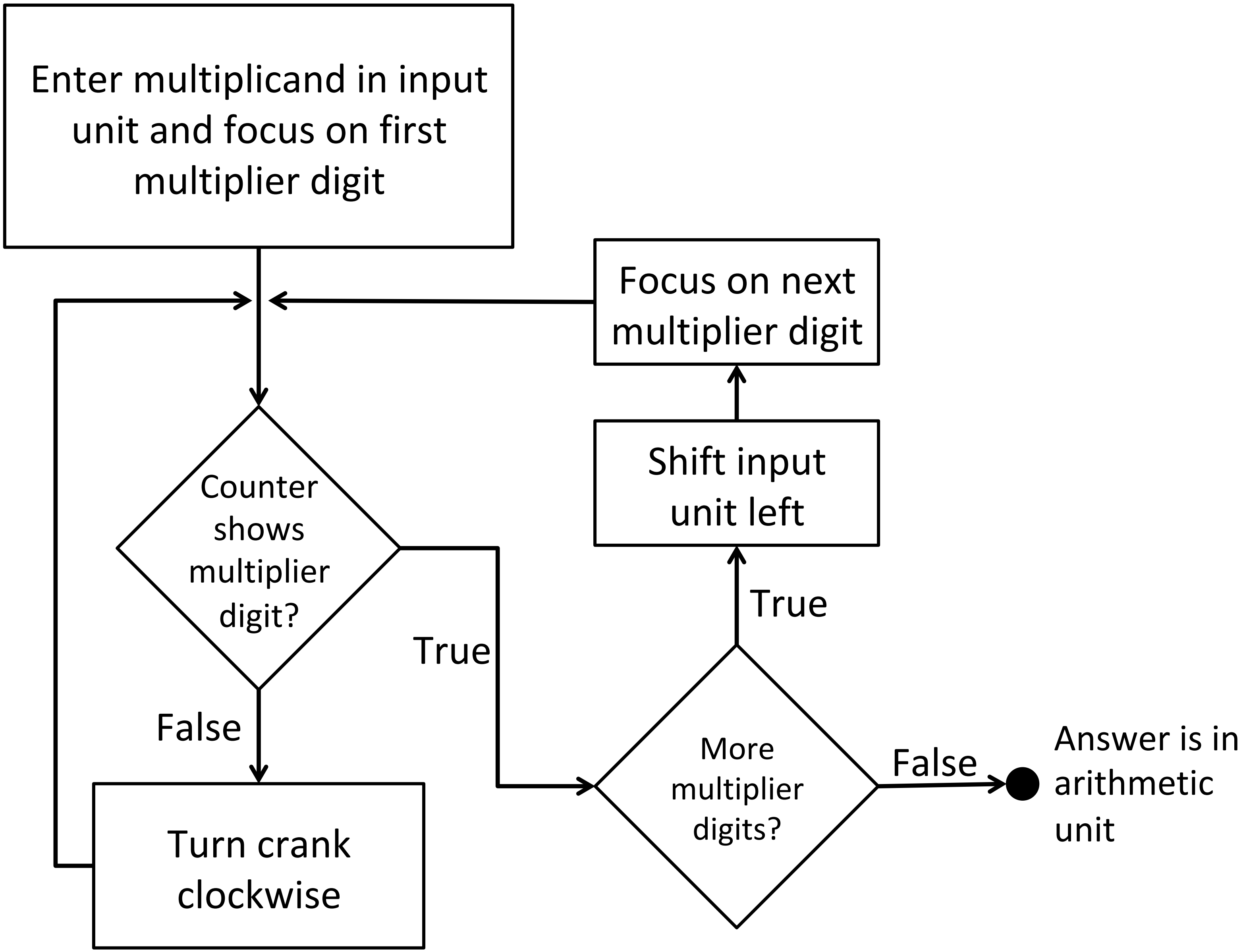
Turn crank clockwise

True

Answer is in arithmetic unit

Prof. Dr.-Ing. Christian-M. Hamann Online Museum [http://public.beuth-hochschule.de/hamann/calc/index.html](http://public.beuth-hochschule.de/hamann/calc/index.html)

# Multiplication (Faster!)



Enter multiplicand in input unit and focus on first multiplier digit

Counter shows multiplier digit?

False

Turn crank clockwise

True

More multiplier digits?

True

Shift input unit left

Focus on next multiplier digit

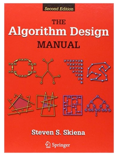
False

Answer is in arithmetic unit

Prof. Dr.-Ing. Christian-M. Hamann Online Museum http://public.beuth-hochschule.de/hamann/calc/index.html

# HOW TO DESIGN ALGORITHMS

# How to Design Algorithms

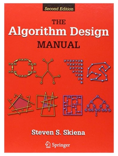


The key to algorithm design (or any other problem-solving task) is to proceed by asking yourself a sequence of questions to guide your thought process

What follows is a sequence of questions to guide your search for the right algorithm for a problem

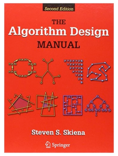
- To use it effectively, you must not only ask the questions, but answer them.
- The key is working through the answers carefully, by writing them down in a log.
- The correct answer to, ``Can I do it this way?'' is never ``no,'' but ``no, because ....''

# How to Design Algorithms



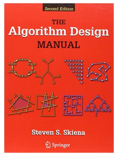
1. Do I really understand the problem?
2. Can I find a simple algorithm or heuristic for the problem?
3. Is my problem in the catalog of well known algorithmic problems (e.g. those in The Algorithm Design Manual)?
4. Are there special cases of the problem that I know how to solve exactly?
5. Am I still stumped?

# How to Design Algorithms



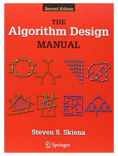
1. Do I really understand the problem?
   - What exactly does the input consist of?
   - What exactly are the desired results or output?
   - Can I construct an example input small enough to solve by hand? What happens when I try to solve it?
   - How important is it to my application that I always find an exact, optimal answer? Can I settle for something that is usually pretty good?
   - How large will a typical instance of my problem be? Will I be working on 10 items? 1,000 items? 1,000,000 items?

# How to Design Algorithms



- How important is speed in my application? Must the problem be solved within one second? One minute? One hour? One day?
- How much time and effort can I invest in implementing my algorithm? Will I be limited to simple algorithms that can be coded up in a day, or do I have the freedom to experiment with a couple of approaches and see which is best?
- Am I trying to solve a numerical problem? A graph algorithm problem? A geometric problem? A string problem? A set problem? Might my problem be formulated in more than one way? Which formulation seems easiest?

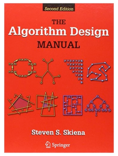
# How to Design Algorithms



2. Can I find a simple algorithm or heuristic for the problem?

  - Can I find an algorithm to solve my problem *correctly* by searching through all subsets or arrangements and picking the best one?
  - Can I solve my problem by repeatedly trying some simple rule, like picking the biggest item first? The smallest item first? A random item first?

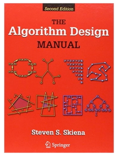
# How to Design Algorithms



3. Is my problem in the catalog of well known algorithmic problems (e.g. those in The Algorithm Design Manual)?

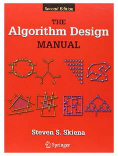
- If it is, what is known about the problem?
- Are there certain operations being repeatedly done on the same data, such as searching it for some element, or finding the largest/smallest remaining element?
- Is there a set of items that can be sorted by size or some key? Does this sorted order make it easier to find the answer?

# How to Design Algorithms



- Is there a way to split the problem into two smaller problems, perhaps by doing a binary search? How about partitioning the elements into big and small, or left and right? Does this suggest a divide-and-conquer algorithm?
- Does my problem seem something like satisfiability, the traveling salesman problem, or some other **NP-complete** problem? If so, might the problem be NP-complete and thus not have an efficient algorithm?

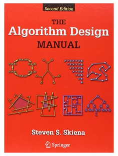
# How to Design Algorithms



4. Are there special cases of the problem that I know how to solve exactly?

- Can I solve the problem efficiently when I ignore some of the input parameters?
- What happens when I set some of the input parameters to trivial values, such as 0 or 1? Does the problem become easier to solve?
- Can I simplify the problem to the point where I *can* solve it efficiently? Is the problem now trivial or still interesting?

# How to Design Algorithms



Second Edition

THE

Algorithm Design

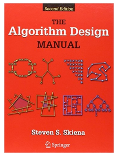
MANUAL

Steven S. Skiena

Springer

- Once I know how to solve a certain special case, why can't this be generalized to a wider class of inputs?
- Is my problem a special case of a more general well-known problem?

# How to Design Algorithms



5. Am I still stumped?
   - Why don't I go back to the beginning and work through these questions again? Did any of my answers change during my latest trip through the list?

# PLAYING CARDS EXERCISE

# Playing Cards Exercise

**Ordering**

Black < Red

A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K

# SUMMARY

# Summary

Writing computer programmes to solve interesting problems requires writing algorithms

Developing effective algorithms is an art as well as a science

In developing algorithms we try to achieve (or sometimes balance):

- correct
- efficient
- easy to implement