Chapter 2
Algorithm Design

Before we discuss how to design algorithms, we must make up minds about how to represent them, i.e., what notation should we use to express algorithms so that they are clear, precise, and unambiguous.

We don’t want to use a programming language at this stage since we want to be flexible and don’t want to deal with details.

Natural language is not a good choice either, since 1) the verbosity leads to an unstructured description; 2) it is unstable so that it cannot be finitely specified; and 3) the context sensitivity leads to ambiguity;
Happy Birthday, Chris!

Given the following statement:

Bob touches Chris with a bouquet of flowers.

Question: Who has the flower?

Answer: It could be either one.

Hence, the above statement is ambiguous so that a computer is not able to understand it without a clear context.
Pseudocode

*Pseudocode* is a subset of English language constructs that looks like the statements available in most programming languages. It is simple, flexible, and highly readable. With its well-defined structure, it is easier to visualize the organization of a pseudocode algorithm. Finally, it is also easier to transform a pseudocode algorithm into a computer program, since its syntax resembles many programming languages.

We will present the three basic structures, sequential, conditional and, iterative constructs, in pseudocode.
Sequential operations

The fundamental operation sequence involved in every algorithm is *input, computation, output.*

Below shows the instruction for performing a computation and then saving the results.

Set the value of "variable" to "expression"

It *evaluates* the arithmetic expression first and gets a result, which is then stored in the variable. The latter corresponds to a named storage location. For example,

Set the value of Carry to be 0
Input, compute, output

*Input* operations allow the computing agent to receive data from the outside world, which can then be used in the computations, while *output* operations allow the agent to send out results of the computations for future use.

For example, the following is an algorithm, in pseudocode, that computes *average miles per gallon*.

1. Get values for *gallons, start* and *end*

2. Set the value of *distance* to \( (end - start) \)

3. Set the value of *average* to \( \frac{distance}{gallons} \)

4. Print the value of *average*

5. Stop
An example

Write an algorithm that gets three values, $x, y$ and $z$ as input and outputs the average of those three values.

1. Get values for $x, y$ and $z$

2. Set $average$ to $(x + y + z)/3$

3. Print "The average is: ", average.

4. Stop
Another example

Write an algorithm that gets the values of radius $r$ pf a circle as input. Its output is both the circumference and the area of such a circle.

1. Get value for $r$

2. Set $circumference$ to $2\times3.14\times r$

3. Set $area$ to $3.14\times r \times r$

4. Print "Circumference is " $+ circumference$

5. Print "Area is " $+ area$

6. Stop

**Homework:** Exercises 1(a, c), 3 and 4.
Conditional operation

The *conditional operation* lets the algorithm to ask a question, and, based on the answer, selects the next operation to perform. Below is the most commonly used structure.

If "a true-false" condition is true Then
   first set of operations
Else second set of operations.

It *evaluates* the condition first to see if it is *true* or *false*, and then executes the first, or the second set, of operations, accordingly. In either case, continues with the *next* operation.
An example

1. Get values for \textit{gallons}, \textit{start} and \textit{end}

2. Set the value of \textit{distance} to \((\textit{end} - \textit{start})\)

3. Set the value of \textit{average} to \((\textit{distance}/\textit{gallons})\)

4. Print the value of \textit{average}

5. If \textit{average} > 25.0 Then

6. Print the massage “You are getting good gas mileage.”

7. Else Print the massage “You are not getting a good gas mileage.”

8. Stop
Another example

Write an algorithm that inputs your current credit card balance, the total dollar amount of new purchases, and the total amount of all payments. The algorithm computes the new balance, including a 12% interest charge on any unpaid balance.

1. Get values for balance, purchase, and payment

2. Set unpaid to balance – payment

3. If (unpaid > 0)

4. Set charge to unpaid * 0.12

5. Else set charge to 0

6. Set newBalance to unpaid + purchase

7. Print "New balance is " + newBalance

**Homework:** Exercises 5, 6, 7, and 10.
Iterative operation

An iterative operation repeats a block of operations, i.e.,

Repeat step i to step j until a "condition" becomes true
  step i: operation
  step i+1: operation
    ...
  step j: operation

It performs all operations from step i to step j, inclusive, until the condition becomes true. The block from step i to step j is called the loop body, and the condition is called a termination condition, which is used to control the loop.
An example

1. Repeat step 2 to step 10 until response is no
2. Get values for gallons, start and end
3. Set the value of distance to (end − start)
4. Set the value of average to (distance/gallons)
5. Print the value of average
6. If average > 25.0 Then
7. Print the massage “You are getting good gas mileage.”
8. Else Print the massage “You are not getting a good gas mileage.”
9. Print the massage “Do you want to repeat?”
10. Get a value for response from the user
11. Stop
Another example

Write an algorithm that gets as an input $x$ and output three values $x^2$, $\sin x$, and $1/x$. This process is repeated until the input value of $x$ is equal to 999, at while time the program terminates.

1. Get the value for $x$
2. Repeat step 3 to step 7 until $x=999$
3. Set square to $x \times x$
4. Set sine to $\sin(x)$
5. Set reverse to $1/x$
6. Print square, sine and reverse
7. Get the value for $x$
8. Stop

Homework: Exercises 8, 10, and 11.
Variations, etc.

Instead of using the repeat...until format, we can use while...do. Hence,

While "a condition" remains true do
    operation
    ...
    operation

The while loop checks condition first, thus it may not execute even once. On the other hand, the repeat loop executes at least once.
What is the difference between...

the following two pieces?

While not drunk
  Keep on drinking

Repeat
  drinking
until drunk

**Theorem:** If a problem can be solved algorithmically, then it can be solved using only the sequential, conditional, and iterative operations.
Looking, looking and looking.

The first problem we want to discuss in detail is to search for somebody’s telephone number in a phone book. It is already implemented in telephone companies. (?) Let’s assume that we have 10,000 names, \( N_1, \ldots, N_{10000} \), along with that many phone numbers, \( T_1, \ldots, T_{10000} \). We further assume that the names are unique, but NOT need be in the alphabetical order.

The next slide shows an algorithm that lets us input a name and finds out the number.
A first try

Get $Name, N_1, \ldots, N_{10000}, T_1, \ldots, T_{10000}$

If $Name = N_1$ Then write out $T_1$

...$

If Name = N_{10000}$ Then write out $T_{10000}$

Stop

This algorithm is extremely long. (What happens if we have 20 million records?) It is also wrong. (What happens if the name is not in the list?) The first problem can be fixed by using the iterative structure, while the second can also be solved by checking the index at the end.
Sequential search

Get $Name, N_1, \cdots, N_{10000}, T_1, \cdots, T_{10000}$
Set $i$ to 1 and set $Found$ to NO
While both $Found == NO$ and $i \leq 10000$ Do
  If $Name == N_i$ Then
    Print $T_i$
    Set $Name$ to YES
  Else Add 1 to $i$
If ($Found == NO$) Then
Print the massage “Sorry, the name is not in the book.”
Stop

Question: Is this the way we look for a number in the phone book? If it is not, why this way?

Homework: Exercises 12 and 13.
Big, bigger, biggest

The next problem we want to solve is similar to the previous one, in the sense that we still search in a list. This time, we will look for the biggest value, but not a particular one. It is not only useful by itself, but can be used to sort a list of items, etc. Formally, the problem can be specified as follows:

Given a value \( n \geq 2 \), and a list containing exactly \( n \) unique numbers, \( A_1, \ldots, A_n \). find and print out both the largest value in the list and its position.

For example, given 19, 41, 12, 63, 22. The algorithm should print out 63 and 4.
Get some ideas first

Intuitively, we have to search the whole list to find out the biggest value, and we have to save this value and its position somewhere.

With respect to the searching, let’s begin with the first item, check the values one at a time, until we have looked at all the values. As we can’t be sure about the answer until the last step, let’s also keep the biggest value we have seen so far in a pair of variables, and keep on updating this pair during the searching process.

These ideas lead to the algorithm.
The *Find Largest* algorithm

Get \( n \), and \( A_1, \ldots, A_n \)
Set \( \text{Largest} \) to be \( A_1 \)
Set \( \text{Location} \) to 1
Set \( i \) to 2
While \((i \leq n)\) Do
  If \( A_i > \text{Largest} \) Then
    Set \( \text{Largest} \) to \( A_i \)
    Set \( \text{Location} \) to \( i \)
    Add 1 to \( i \)
Print out \( \text{Largest} \) and \( \text{Location} \)

**Question:** How can we modify the algorithm so that we can find the *smallest* value in the list?

**Question:** What happens if \( n == 0 \) or \( n == 1 \)?

**Homework:** Exercises 14, 15, 16 and 17.
Meeting your match

The last problem we want to discuss in this chapter is the *pattern matching* problem. It looks for a special pattern or symbols in a large collection of information. It is implemented as the *Find* feature in every word processor. This technique can be used to look for anything in anywhere.

For example, based on the DNA technique, human gene is composed of four basic chemical compounds, *adenine*, *cytosine*, *thymine*, and *guanine*. If we use the first letter to represent them, then, our very existence can be represented as a very large text file in four letters, e.g., ···ACTCTAGA···. Hence, DNA matching becomes pattern matching.
The problem

Given some text composed of \( n \) characters, \( T_1, \cdots, T_n \), and a pattern of \( m(\leq n) \) characters, \( P_1, \cdots, P_m \). The algorithm must locate every occurrence of the pattern within the text. The output is the locations where each match occurs, i.e., the index where the match begins.

For example, given the text “to be or not to be, that is the question” and the pattern “to”, the output should be 1 and 14.

As the match can occur in any position, so the intuitive thing to do is to put the first symbol at every possible position, then check to see if the rest of the pattern match with the following text.

Homework: Exercise 18 in pp. 62.
The first try

Get values for $n$ and $m$, respectively
Get the text $T_1, \ldots, T_n$ and the pattern $P_1, \ldots, P_m$
Set $k$, the starting location, to be 1
Repeat until the whole text is checked
   Attempt to match every character in the pattern with the text beginning at $k$
   If there is a match Then
      Print $k$
      Add 1 to $k$
Stop

Question: Is the algorithm fully blown up? What does that mean by the condition the whole text is checked?

If we think a bit more deeply, we find out that the last position $k$ could be is $n - m + 1$, otherwise, we don’t have enough symbols in the text to match with the pattern.
The Pastern Matching algorithm

Get values for $n$ and $m$, respectively
Get the text $T_1, \cdots, T_n$ and the pattern $P_1, \cdots, P_m$
Set $k$, the starting location, to be 1
While $k \leq n - m + 1$
  Do Initialize $i$ to 1
  Initialize $mismatch$ to NO
  While ($i \leq m$) and $mismatch==NO$
    If $P_i \neq T_{k+i-1}$ Then
      Set $mismatch$ to YES
    Else Increment $i$ by 1
  If $mismatch==NO$
    Print “There is a match at” $k$
  Increment $k$ by 1
Stop

**Homework:** Exercise 18 and 19.