INTRODUCTION TO ALGORITHMS

Objects of the lecture:

After studying this chapter, the students will be able to

- Understand the importance of algorithms
- Trace the origin of algorithm
- Define an algorithm
- Learn the various ways of writing an algorithm
- Understand what the future has in store for us vis-à-vis algorithms
- Understand the concept of designing an algorithm

INTRODUCTION

Algorithms are used everywhere, from a coffee machine to a nuclear power plant. A good algorithm should use the resources such as the CPU usage, time, and memory judiciously. It should also be unambiguous and understandable. The output produced by an algorithm lies in a set called range. The input is taken from a set 'domain' (input constraints). From the domain only the values satisfying given constraints can be taken. These constraints are referred to as input constraints. Input constraints determine the values of xi, i.e., input. The inputs are related to each other as governed by relation corresponding to the task that is to be accomplished. This is referred to as explicit constraint .

Summarizing the importance of algorithms discussed earlier, we can say the following:

- It helps in enhancing the thinking process. They are like brain stimulants that will give a boost to our thinking process.
- It helps in solving many problems in computer science, computational biology, and economics.
- Without the knowledge of algorithms we can become a coder but not a programmer.

- **A good understanding of algorithms will help us to get a job. There is an immense demand of good programmers in the software industry who can analyze the problem well.**
- **The fourth section of the book that introduces genetic algorithms and randomized approach will help us to retain that job.**

ALGORITHM DEFINITION

An algorithm is a sequence of steps that must be carried out in order to accomplish a particular task. Three things are to be considered while writing an algorithm: input, process, and output. The input that we give to an algorithm is processed with the help of the procedure and finally, the algorithm returns the output. It may be stated at this point that an algorithm may not even have an input. An example of such an algorithm is pseudorandom number generator (PRNG). Some random number generators generate a number without a seed. In such cases, the algorithm does not require any input. The processing of the inputs generates an output. This processing is the most important part of the algorithm. While writing an algorithm, the time taken to accomplish the task and the memory usage must also be considered. The prime motto is to solve the problem but efficiency of the process followed should not be compromised. There is a distinction between natural language and algorithmic writing. While speaking or writing a letter, we may use ambiguous terms unknowingly or deliberately. To summarize the main goal of discussion

- An algorithm is a sequence of steps in order to carry out a particular task.
- It can have zero or more inputs.
- It must have at least one output.
- It should be efficient both in terms of memory and time.
- It should be finite.
- Every statement should be unambiguous. The meaning of finite is that the algorithm should have countable number of steps.

It may be stated that a program can run infinitely but an algorithm is always finite. For example, an operating system of a server, in spite of being a program runs 24 × 7 but an algorithm cannot be infinite.

1. WAYS OF WRITING AN ALGORITHM:

There are various ways of writing an algorithm. In this section, three ways have been explained and exemplified taking requisite examples.

A. ENGLISH-LIKE ALGORITHM

An algorithm can be written in many ways. It can be written in simple English but this methodology also has some demerits. Natural languages can be ambiguous and therefore lack the characteristic of being definite. Since each step of an algorithm should be clear and should not have more than one meaning, English language-like algorithms are not considered good for most of the tasks. However, an example of linear search, in which an element is searched at every position of the array and the position is printed if an element is found, is given below. In this algorithm, ‘A’ is the

array in which elements are stored and 'item' is the value which is to be searched. The algorithm assumes that all the elements in 'A' are distinct. Algorithm below depicts the above process.

Algorithm 1.1: English-like algorithm of linear search

Step 1. Compare 'item' with the first element of the array, A.
Step 2. If the two are same, then print the position of the element and exit. Step 3. else repeat the above process with the rest of the elements.
Step 4. If the item is not found at any position, then print 'not found' and exit.

B. FLOWCHART

Flowcharts pictorially depict a process. They are easy to understand and are commonly used in the case of simple problems. The process of linear search, explained in the previous subsection, is depicted in the flowchart illustrated in Fig. 1.1. The conventions of flowcharts are depicted in Table 1.1.

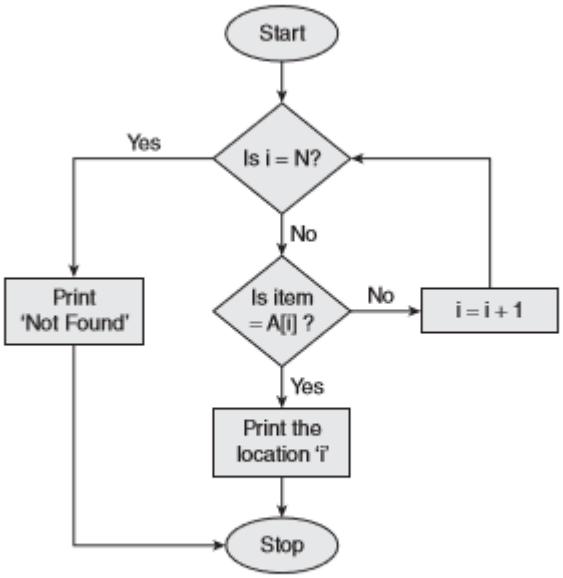
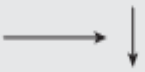
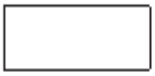


Figure 1.1 Flowchart of linear search

In the flowchart, shown in Fig 1.1, A[ ] is an array containing N elements. The index of the first element is O which is also the initial value of i. Such depictions, though easy to comprehend, are used only for simple straightforward problems. Hence, this lecture neither recommends nor uses the above two types for writing algorithms, except for some cases.

Table 1.1 Flowchart conventions

S. No.	Name	Element Representation	Meaning
1.	Start/End		An oval is used to indicate the beginning and end of an algorithm.
2.	Arrows		An arrow indicates the direction of flow of the algorithm.
3.	Connectors		Circles with arrows connect the disconnected flowchart.
4.	Input/Output		A parallelogram indicates the input or output.
5.	Process		A rectangle indicates a computation.
6.	Decision		A diamond indicates a point where a decision is made.

C. PSEUDOCODE

The pseudocode has an advantage of being easily converted into any programming language. This way of writing algorithm is most acceptable and most widely used. In order to be able to write a pseudocode, one must be familiar with the conventions of writing it. Table 1.2 shows the pseudocode conventions.

Table 1.2 Pseudocode conventions

S. No.	Construct	Meaning
1.	`// Comment`	Single line comments start with //

2.	`/* Comment Line 1`
	`Comment Line 2`
	`o`
	`o`
	`Comment Line n`
	`*/`
		Multi-line comments occur between /* and */

3.	`{`
	`statements`
	`}`
		Blocks are represented using { and }. Blocks can be used to represent compound statements (collection of simple statements) or the procedures.

4.	`;`	Statements are delimited by ;

(Contd)

Table 1.2 (Contd)

S. No.	Construct	Meaning
5.	`<variable> = <Expression>`	This is an assignment statement. The statement indicates that the result of evaluation of the expression will be stored in the variable.
6.	`a > b`	a and b are expressions, and > is a relational operator 'greater than'. The Boolean expression `a > b` returns true if a is greater than b, else returns false.
7.	`a < b`	a and b are expressions, and < is a relational operator 'less than'. The Boolean expression `a < b` returns true if a is less than b, else returns false.
8.	`a <= b`	a and b are expressions, and <= is a relational operator 'less than or equal to'. The Boolean expression `a <= b` returns true if a is less than or equal to b, else returns false.
9.	`a >= b`	a and b are expressions, and >= is a relational operator 'greater than or equal to'. The Boolean expression `a >= b` returns true if a is greater than or equal to b, else returns false.
10.	`a != b`	a and b are expressions, and != is a relational operator 'not equal to'. The Boolean expression `a != b` returns true if a is not equal to b, else returns false.
11.	`a == b`	a and b are expressions, and == is a relational operator 'equal to'. The Boolean expression `a == b` returns true if a is equal to b, else returns false.
12.	`a AND b`	a and b are expressions, and AND is a logical operator. The Boolean expression `a AND b` returns true if both the conditions are true, else it returns false.
13.	`a OR B`	a and b are expressions, and OR is a logical operator. The Boolean expression `a OR b` returns true if any of the condition is true, else it returns false.
14.	`NOT a`	a is an expression, and NOT is a logical operator. The Boolean expression 'NOT a' returns true if the result of a evaluates to False, else returns False.
15.	`if<condition>then<statement>`	The statement indicates the conditional operator if.
16.	`if<condition>then<statement1>` `else<statement2>`	The statement is an enhancement of the above if statement. It can also handle the case wherein the condition is not satisfied.

17.	`Case`
	`{`
	`:<condition 1>: <statement 1>`
	`o`
	`o`
	`:<condition n>: <statement n>`
	`:default: <statement n+1>`
	`}`
		The statement is a depiction of switch case used in C or C++.

(Contd)

Table 1.2 (Contd)

S. No.	Construct	Meaning

18.	`while<condition>do`
	`{`
	`statements`
	`}`
		The statement depicts a while loop

19.	`repeat`
	`statements`
	`until<condition>`
		The statement depicts a do-while loop

20.	`for variable = value1 to value2`
	`{`
	`statements`
	`}`
		The statement depicts a for loop

21.	`Read`	Input instruction
22.	`Print`	Output instruction
23.	`Algorithm<name> (<parameter list>)`	The name of the algorithm is <name> and the arguments are stored in the <parameter list>

Algorithm 1.2 depicts the process of linear search. The name of the algorithm is 'Linear Search'. The element 'item' is to be searched in the array 'A'. The algorithm uses the conventions stated in Table 1.2.

Algorithm 1.2 linear search

```
Algorithm Linear_Search (A, n, item)
{
     for i = 1 to n step 1 do
     {       if(A[i] == item)
             {           print i;
                       exit();
              }
     }
     print "Not Found"
}
```

2. DESIGN AND ANALYSIS VS ANALYSIS AND DESIGN

In order to accomplish a task, a solution needs to be developed. This is called designing of an algorithm. For example, if an array 'A' of length n is given and our requirement is to find out the maximum element of the array, then we can take a variable 'Max' whose initial value is A[1], which is the first element of the array. Now, start traversing the array, compare the value of Max with each element, if we are able to find any element greater than Max, then we can set Max to the value of that element, else continue. The process is depicted in Algorithm 1.3.

Algorithm 1.3 Finding maximum element from an array

```
Algorithm Max(A, n)
{    Max = A[1];
     for i = 2 to n step 1 do
     {
          if(A[i]>Max) then
          {
            Max=A[i];
          }
      }
         Print "The maximum element is A[i]"
 }
```

The next step would be to analyse the time complexity of the algorithm. Table 1.3 shows the number of times each statement is executed. The above analysis gives an idea of maximum amount of resources (in this case time) required to run the algorithm. This is referred to as algorithm design and analysis (ADA) (see Fig. 1.2). However, this may not be the case most of the times. Often, we have to develop software for the client. The client has some set-up and will not want to upgrade his systems in order to install the software. In such cases, we must analyse the hardware and

Table 1.3 Number of times each statement is executed in Algorithm 1.3

`Max = A[1];`	1
`for i := 2 to n step 1 do{`	n
`if(A[i]>max) {`	n–1
`Max=A[i];}}`	less than or equal to (n–1)
`print "The maximum element is A[i]"`	1
	Maximum: (n)

Figure 1.2 Design and analysis

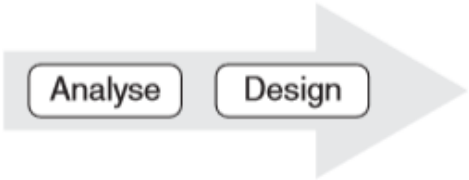


Figure 1.3 Analysis and design

The set-up of the client and then decide on the algorithms we would be using in order to accomplish the tasks. Here, we cannot apply techniques like diploid genetic algorithm on a system that uses a P4, similarly there is no point in using algorithms that are time efficient but probably use extensive resources in a very advanced set-up. The process is referred to as analysis and design. The process is depicted in Fig. 1.3. The general approach being used is design and analysis; however, analysis and design is far more practical and hence implementable.