CLASSIFICATION OF DATA STRUCTURES

Graphs

- Graphs have so many applications in computer science and mathematics that several algorithms have been written to perform the standard graph operations, such as searching the graph and finding the shortest path between the nodes of a graph.
- Note that unlike trees, graphs do not have any root node.
- Rather, every node in the graph can be connected with every another node in the graph.
- When two nodes are connected via an edge, the two nodes are known as neighbors.
- For example, in Fig. 2.8, node A has two neighbors: B and D.

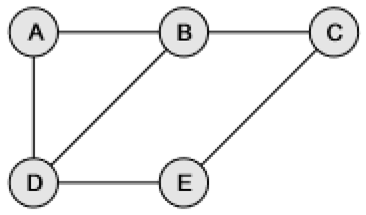


Figure 2.8 Graph

OPERATIONS ON DATA STRUCTURES

- This section discusses the different operations that can be performed on the various data structures previously mentioned.
- *Traversing:* It means to access each data item exactly once so that it can be processed. For example, to print the names of all the students in a class.
- *Searching:* It is used to find the location of one or more data items that satisfy the given constraint. Such a data item may or may not be present in the given collection of data items. For example, to find the names of all the students who secured 100 marks in mathematics.
- *Inserting:* It is used to add new data items to the given list of data items. For example, to add the details of a new student who has recently joined the course.

OPERATIONS ON DATA STRUCTURES

- *Deleting:* It means to remove (delete) a particular data item from the given collection of data items. For example, to delete the name of a student who has left the course.
- *Sorting:* Data items can be arranged in some order like ascending order or descending order depending on the type of application. For example, arranging the names of students in a class in an alphabetical order, or calculating the top three winners by arranging the participants' scores in descending order and then extracting the top three.
- *Merging:* Lists of two sorted data items can be combined to form a single list of sorted data items.
- Many a time, two or more operations are applied simultaneously in a given situation. For example, if we want to delete the details of a student whose name is X, then we first have to search the list of students to find whether the record of X exists or not and if it exists then at which location, so that the details can be deleted from that particular location

ABSTRACT DATA TYPE

- An abstract data type (ADT) is the way we look at a data structure, focusing on what it does and ignoring how it does its job.
- For example, stacks and queues are perfect examples of an ADT.
- We can implement both these ADTs using an array or a linked list.
- The end-user is not concerned about the details of how the methods carry out their tasks.
- They are not concerned about how they work.
- For example, when we use a stack or a queue, the user is concerned only with the type of data and the operations that can be performed on it.

ALGORITHMS

- The typical definition of algorithm is 'a formally defined procedure for performing some calculation'.
- If a procedure is formally defined, then it can be implemented using a formal language, and such a language is known as a programming language.
- In general terms, an algorithm provides a blueprint to write a program to solve a particular problem.
- It is considered to be an effective procedure for solving a problem in finite number of steps.
- That is, a well-defined algorithm always provides an answer and is guaranteed to terminate.

ALGORITHMS

- An algorithm is basically a set of instructions that solve a problem.
- It is not uncommon to have multiple algorithms to tackle the same problem, but the choice of a particular algorithm must depend on the time and space complexity of the algorithm.
- A complex algorithm is often divided into smaller units called modules.
- This process of dividing an algorithm into modules is called modularization.

DIFFERENT APPROACHES TO DESIGNING AN ALGORITHM

- There are two main approaches to design an algorithm—top-down approach and bottom-up approach, as shown in Fig. 2.9.

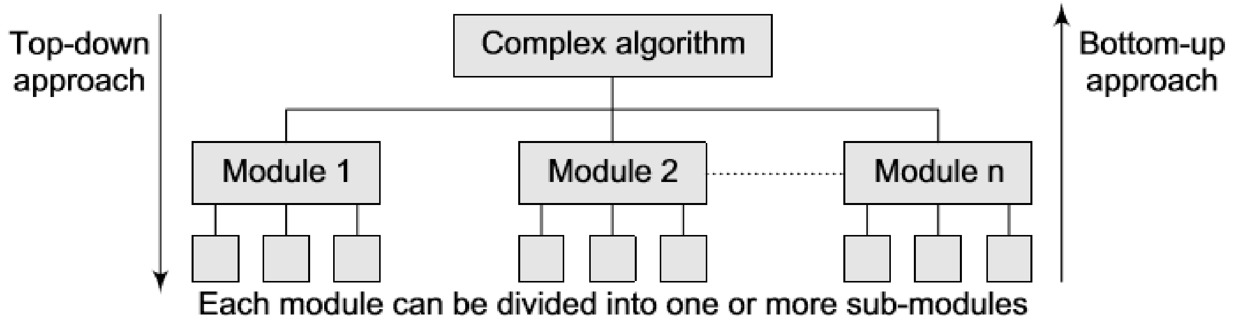


Figure 2.9 Different approaches of designing an algorithm

Top-down approach

- A top-down design approach starts by dividing the complex algorithm into one or more modules.
- These modules can further be decomposed into one or more sub-modules, and this process of decomposition is iterated until the desired level of module complexity is achieved.
- Top-down design method is a form of stepwise refinement where we begin with the topmost module and incrementally add modules that it calls.
- Therefore, in a top-down approach, we start from an abstract design and then at each step, this design is refined into more concrete levels until a level is reached that requires no further refinement.

DIFFERENT APPROACHES TO DESIGNING AN ALGORITHM

- There are two main approaches to design an algorithm—top-down approach and bottom-up approach, as shown in Fig. 2.9.

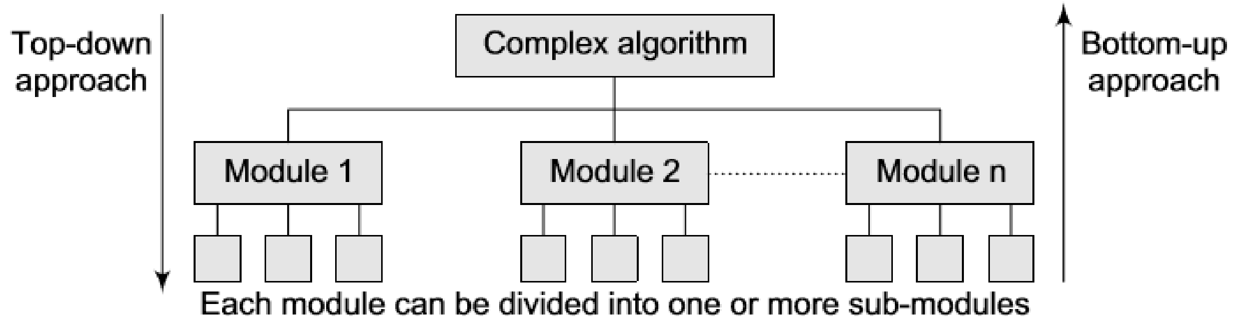


Figure 2.9 Different approaches of designing an algorithm

Bottom-up approach

- A bottom-up approach is just the reverse of top-down approach. In the bottom-up design, we start with designing the most basic or concrete modules and then proceed towards designing higher level modules.
- The higher level modules are implemented by using the operations performed by lower level modules.
- Thus, in this approach sub-modules are grouped together to form a higher level module.
- All the higher level modules are clubbed together to form even higher level modules.
- This process is repeated until the design of the complete algorithm is obtained.

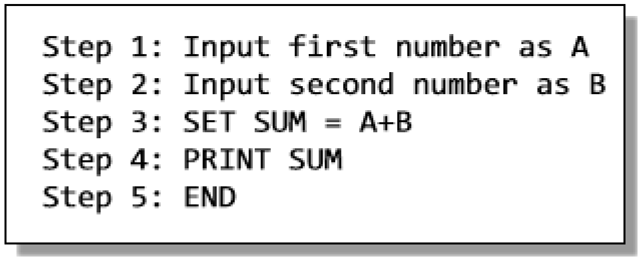
CONTROL STRUCTURES USED IN ALGORITHMS

- An algorithm has a finite number of steps.
- Some steps may involve decision-making and repetition.
- An algorithm may employ one of the following control structures:

  (a) sequence (Figure 2.10),

  (b) decision, and

  (c) repetition.

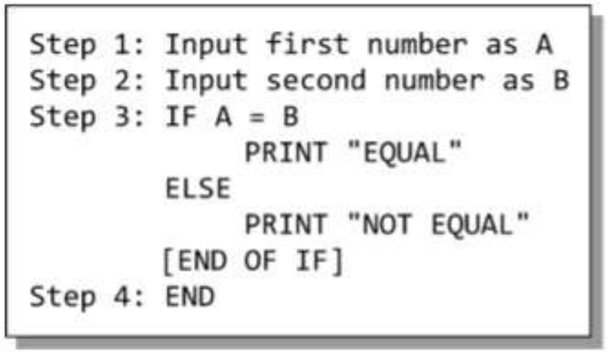


```
Step 1: Input first number as A
Step 2: Input second number as B
Step 3: SET SUM = A+B
Step 4: PRINT SUM
Step 5: END
```

Figure 2.10 Algorithm to add two numbers

CONTROL STRUCTURES USED IN ALGORITHMS

- An algorithm has a finite number of steps.
- Some steps may involve decision-making and repetition.
- An algorithm may employ one of the following control structures:
  - (a) sequence (Figure 2.10),
  - (b) decision (Figure 2.11), and
  - (c) repetition.



```
Step 1: Input first number as A
Step 2: Input second number as B
Step 3: IF A = B
            PRINT "EQUAL"
        ELSE
            PRINT "NOT EQUAL"
        [END OF IF]
Step 4: END
```

Figure 2.11 Algorithm to test for equality of two numbers