ALGORITHM

Algorithm is a step-by-step procedure, which defines a set of instructions to be executed in a certain order to get the desired output. Algorithms are generally created independent of underlying languages, i.e. an algorithm can be implemented in more than one programming language.

From a data structure point of view, the following are some important categories of algorithms −

- **Search** − Algorithm to search an item in a data structure.
- **Sort** − Algorithm to sort items in a certain order.
- **Insert** − Algorithm to insert item in a data structure.
- **Update** − Algorithm to update an existing item in a data structure.
- **Delete** − Algorithm to delete an existing item from a data structure.

Characteristics of an Algorithm

Not all procedures can be called an algorithm. An algorithm should have the following characteristics −

- **Unambiguous** − Algorithm should be clear and unambiguous. Each of its steps (or phases), and their inputs/outputs should be clear and must lead to only one meaning.
- **Input** − An algorithm should have 0 or more well-defined inputs.
- **Output** − An algorithm should have 1 or more well-defined outputs, and should match the desired output.
- **Finiteness** − Algorithms must terminate after a finite number of steps.
- **Feasibility** − Should be feasible with the available resources.
- **Independent** − An algorithm should have step-by-step directions, which should be independent of any programming code.

Algorithm Analysis

Efficiency of an algorithm can be analyzed at two different stages, before implementation and after implementation. They are the following −

- **A Priori Analysis** − This is a theoretical analysis of an algorithm. Efficiency of an algorithm is measured by assuming that all other factors, for example, process or speed, are constant and have no effect on the implementation.
- **A Posterior Analysis** − This is an empirical analysis of an algorithm. The selected algorithm is implemented using programming language. This is then executed on target computer machine. In this analysis, actual statistics like running time and space required, are collected.

Algorithm analysis deals with the execution or running time of various operations involved. The running time of an operation can be defined as the number of computer instructions executed per operation.

Algorithm Complexity

Suppose **X** is an algorithm and **n** is the size of input data, the time and space used by the algorithm X are the two main factors, which decide the efficiency of X.

- **Time Factor** − Time is measured by counting the number of key operations such as comparisons in the sorting algorithm.
- **Space Factor** − Space is measured by counting the maximum memory space required by the algorithm.

The complexity of an algorithm **f(n)** gives the running time and/or the storage space required by the algorithm in terms of **n** as the size of input data.

Space Complexity

Space complexity of an algorithm represents the amount of memory space required by the algorithm in its life cycle. The space required by an algorithm is equal to the sum of the following two components −

- A fixed part that is a space required to store certain data and variables, that are independent of the size of the problem. For example, simple variables and constants used, program size, etc.
- A variable part is a space required by variables, whose size depends on the size of the problem. For example, dynamic memory allocation, recursion stack space, etc.

Space complexity S(P) of any algorithm P is S(P) = C + SP(I), where C is the fixed part and S(I) is the variable part of the algorithm, which depends on instance characteristic I.

Time Complexity

Time complexity of an algorithm represents the amount of time required by the algorithm to run to completion. Time requirements can be defined as a numerical function T(n), where T(n) can be measured as the number of steps, provided each step consumes constant time.

For example, addition of two n-bit integers takes **n** steps. Consequently, the total computational time is T(n) = c ∗ n, where c is the time taken for the addition of two bits. Here, we observe that T(n) grows linearly as the input size increases.
Asymptotic analysis of an algorithm refers to defining the mathematical boundation/framing of its run-time performance. Using asymptotic analysis, we can very well conclude the best case, average case, and worst case scenario of an algorithm.

Asymptotic analysis refers to computing the running time of any operation in mathematical units of computation. Asymptotic analysis is input bound i.e., if there's no input to the algorithm, it is concluded to work in a constant time. Other than the "input" all other factors are considered constant. For example, the running time of one operation is computed as *f*(n) and may be for another operation it is computed as *g*(n^2). This means the first operation running time will increase linearly with the increase in **n** and the running time of the second operation will increase exponentially when **n** increases. Similarly, the running time of both operations will be nearly the same if **n** is significantly small.

Usually, the time required by an algorithm falls under three types −

- **Best Case** − Minimum time required for program execution.
- **Average Case** − Average time required for program execution.
- **Worst Case** − Maximum time required for program execution.

Asymptotic Notations

Following are the commonly used asymptotic notations to calculate the running time complexity of an algorithm.

- Ο Notation
- Ω Notation
- θ Notation

Big Oh Notation, Ο

The notation Ο(n) is the formal way to express the upper bound of an algorithm's running time. It measures the worst case time complexity or the longest amount of time an algorithm can possibly take to complete.

Omega Notation, Ω

The notation Ω(n) is the formal way to express the lower bound of an algorithm's running time. It measures the best case time complexity or the best amount of time an algorithm can possibly take to complete.

Theta Notation, θ

The notation θ(n) is the formal way to express both the lower bound and the upper bound of an algorithm's running time.

An algorithm is designed to achieve optimum solution for a given problem. **In greedy algorithm approach**, decisions are made from the given solution domain. As being greedy, the closest solution that seems to provide an optimum solution is chosen.
Greedy algorithms try to find a localized optimum solution, which may eventually lead to globally optimized solutions. However, generally greedy algorithms do not provide globally optimized solutions.

Examples

Most networking algorithms use the greedy approach. Here is a list of few of them −

- Travelling Salesman Problem
- Prim's Minimal Spanning Tree Algorithm
- Kruskal's Minimal Spanning Tree Algorithm

- Dijkstra's Minimal Spanning Tree Algorithm
- Graph - Map Coloring
- Graph - Vertex Cover
- Knapsack Problem
- Job Scheduling Problem

There are lots of similar problems that uses the greedy approach to find an optimum solution.

In divide and conquer approach, the problem in hand, is divided into smaller sub-problems and then each problem is solved independently. When we keep on dividing the sub problems into even smaller sub-problems, we may eventually reach a stage where no more division is possible. Those "atomic" smallest possible sub-problem (fractions) are solved. The solution of all sub-problems is finally merged in order to obtain the solution of an original problem.

Broadly, we can understand **divide-and-conquer** approach in a three-step process.

Divide/Break

This step involves breaking the problem into smaller sub-problems. Sub-problems should represent a part of the original problem. This step generally takes a recursive approach to divide the problem until no sub-problem is further divisible. At this stage, sub-problems become atomic in nature but still represent some part of the actual problem.

Conquer/Solve

This step receives a lot of smaller sub-problems to be solved. Generally, at this level, the problems are considered 'solved' on their own.

Merge/Combine

When the smaller sub-problems are solved, this stage recursively combines them until they formulate a solution of the original problem. This algorithmic approach works recursively and conquer & merge steps works so close that they appear as one.

Examples

The following computer algorithms are based on **divide-and-conquer** programming approach −

- Merge Sort
- Quick Sort
- Binary Search
- Strassen's Matrix Multiplication
- Closest pair (points)

There are various ways available to solve any computer problem, but the mentioned are a good example of divide and conquer approach.

Dynamic programming approach is similar to divide and conquer in breaking down the problem into smaller and yet smaller possible sub-problems. But unlike, divide and conquer, these sub-problems are not solved independently. Rather, results of these smaller sub-problems are remembered and used for similar or overlapping sub-problems.

Dynamic programming is used where we have problems, which can be divided into similar sub-problems, so that their results can be re-used. Mostly, these algorithms are used for optimization. Before solving the in-hand sub-problem, dynamic algorithm will try to examine the results of the previously solved sub-problems. The solutions of sub-problems are combined in order to achieve the best solution.
So we can say that −

- The problem should be able to be divided into smaller overlapping sub-problem.
- An optimum solution can be achieved by using an optimum solution of smaller sub-problems.
- Dynamic algorithms use Memorization.

Comparison

In contrast to greedy algorithms, where local optimization is addressed, dynamic algorithms are motivated for an overall optimization of the problem.
In contrast to divide and conquer algorithms, where solutions are combined to achieve an overall solution, dynamic algorithms use the output of a smaller sub-problem and then try to optimize a bigger sub-problem. Dynamic algorithms use Memoization to remember the output of already solved sub-problems.

Example

The following computer problems can be solved using dynamic programming approach −

- Fibonacci number series
- Knapsack problem
- Tower of Hanoi
- All pair shortest path by Floyd-Warshall
- Shortest path by Dijkstra
- Project scheduling

Dynamic programming can be used in both top-down and bottom-up manner. And of course, most of the times, referring to the previous solution output is cheaper than recomputing in terms of CPU cycles.

FLOWCHART

A flowchart is a picture of the separate steps of a process in sequential order. It is a generic tool that can be adapted for a wide variety of purposes, and can be used to describe various processes, such as a manufacturing process, an administrative or service process, or a project plan.

Elements that may be included in a flowchart are a sequence of actions, materials or services entering or leaving the process (inputs and outputs), decisions that must be made, people who become involved, time involved at each step, and/or process measurements.

When to Use a Flowchart

- To develop understanding of how a process is done
- To study a process for improvement
- To communicate to others how a process is done
- When better communication is needed between people involved with the same process
- To document a process
- When planning a project

Commonly Used Symbols in Detailed Flowcharts

▭	Process
→	Direction of flow
◇	Decision
◗	Delay or wait
○	Page Connector
▱	Data or Input/Output
▢	Terminator

A flowchart is a diagram that depicts a process, system or computer algorithm. They are widely used in multiple fields to document, study, plan, improve and communicate often complex processes in clear, easy-to-understand diagrams. Flowcharts, sometimes spelled as flow charts, use rectangles, ovals, diamonds and potentially numerous other shapes to define the type of step, along with connecting arrows to define flow and sequence.

They can range from simple, hand-drawn charts to comprehensive computer-drawn diagrams depicting multiple steps and routes. If we consider all the various forms of flowcharts, they are one of the most common diagrams on the planet, used by both technical and non-technical people in numerous fields.

Flowcharts are sometimes called by more specialized names such as Process Flowchart, Process Map, Functional Flowchart, Business Process Mapping, Business Process Modeling and Notation (BPMN), or Process Flow Diagram (PFD). They are related to other popular diagrams, such as Data Flow Diagrams (DFDs) and Unified Modeling Language (UML) Activity Diagrams.

A flowchart is simply a graphical representation of steps. It shows steps in sequential order and is widely used in presenting the flow of algorithms, workflow or processes. Typically, a flowchart shows the steps as boxes of various kinds, and their order by connecting them with arrows.

How to Create a Flow Chart

- Identify Tasks
- Organize and Document Tasks
- Double-Check the Process
- Challenge the Flow Chart

Types of Flowchart

- **Document Flowcharts:** These "have the purpose of showing existing controls over document-flow through the components of a system. … The chart is read from left to right and documents the flow of documents through the various business units."
- **Data Flowcharts:** These show "the controls governing data flows in a system. … Data flowcharts are used primarily to show the channels that data is transmitted through the system rather than how controls flow."
- **System Flowcharts:** These "show the flow of data to and through the major components of a system such as data entry, programs, storage media, processors, and communication networks."
- **Program Flowcharts:** These show "the controls placed internally to a program within a system.

- **System Flowchart:** Identifies the devices to be used.
- **General Flowchart:** Overview.
- **Detailed Flowchart:** Increased detail.
- **System Flowchart.**
- **Program Flowchart.**
- **Decision Flowchart.**
- **Logic Flowchart.**
- **Systems Flowchart.**
- **Product Flowchart.**
- **Process Flowchart.**
- **Swimlane Diagram, a.k.a Swimlane Flowchart:** To delineate who does what in cross-team processes.
- **Workflow Flowchart:** To document workflows, often involving tasks, documents and information in offices.
- **Event-Driven Process Chain (EPC) Flowchart:** To document or plan a business process.
- **Specification and Description Language (SDL) Flowchart:** To brainstorm computer algorithms using three basic components: system definition, block and process.
- **Data Flow Diagram (DFD):** To map out the flow of information for any system or process.
- **Process Flow Diagram (PFD), a.k.a. Process Flowchart:** To illustrate the relationships between major components at an industrial plant.
- **Business Process Model and Notation (BPMN 2.0):** To model the steps of a planned business process.

PSEUDOCODE

Pseudocode is an informal way of programming description that does not require any strict programming language syntax or underlying technology considerations. It is used for creating an outline or a rough draft of a program. Pseudocode summarizes a program’s flow, but excludes underlying details. System designers write pseudocode to ensure that programmers understand a software project's requirements and align code accordingly.

Pseudocode is not an actual programming language. So it cannot be compiled into an executable program. It uses short terms or simple English language syntaxes to write code for programs before it is actually converted into a specific programming language. This is done to identify top level flow errors, and understand the programming data flows that the final program is going to use. This definitely helps save time during actual programming as conceptual errors have been already corrected. Firstly, program description and functionality is gathered and then pseudocode is used to create statements to achieve the required results for a program.

Detailed pseudocode is inspected and verified by the designer’s team or programmers to match design specifications. Catching errors or wrong program flow at the pseudocode stage is beneficial for development as it is less costly than catching them later.

Once the pseudocode is accepted by the team, it is rewritten using the vocabulary and syntax of a programming language. The purpose of using pseudocode is an efficient key principle of an algorithm. It is used in planning an algorithm with sketching out the structure of the program before the actual coding takes place.

Because pseudocode is detailed yet readable, it can be inspected by the team of designers and programmers as a way to ensure that actual programming is likely to match design specifications. Catching errors at the pseudocode stage is less costly than catching them later in the development process. Once the pseudocode is accepted, it is rewritten using the vocabulary and syntax of a programming language. Pseudocode is sometimes used in conjunction with computer-aided software engineering-based methodologies.

ADVANTAGES OF PSEUDOCODE

- Pseudocode is understood by the programmers of all types.
- It enables the programmer to concentrate only on the algorithm part of the code development.
- It cannot be compiled into an executable program. Example, Java code : if (i < 10) { i++; } pseudocode :if i is less than 10, increment i by 1.

1. An artificial language designed to express actions that control the behavior of a machine is (a) Algorithm (b) Pseudocode (c) Natural language **(d) Programming language**

2. A person who thinks rationally and objectively to get an unbiased solution to a problem is a **(a) Problem solver** (b) Critical thinker (c) Programmer (d) Systems analyst

3. The preprocessor removes the _______ from the source code. **(a) comments** (b) header files (c) both comments and header files (d) none of the above

4. Quick sort algorithm is an example of (a) Greedy approach (b) Improved binary search (c) Dynamic Programming **(d) Divide and conquer**

5. Quick sort algorithm is an example of (a) Greedy approach (b) Improved binary search **(c) Dynamic Programming** (d) Divide and conquer

6. In C programming, when we remove an item from bottom of the stack, then **(a) The stack will fall down.**(b) Stack will rearranged items. (c) It will convert to LIFO (d) This operation is not allowed.

4. In the expression count = count + 1, the R value of count is (a) Count (b) Count + 1 (c) = count + 1 (d) + 1

5. The waterfall model differs from the systems development life cyle model in that (a) The stages are long and tedious (b) Iterations are difficult and must be done at the end (c) New problems cannot be incorporated in the solution (d) Users are not involved in its development

6. In programming the value of a constant (a) Remains the same once the variable is given a value (b) Changes regularly in the program Does not changes during program execution (c) Is only affected when new values are given to it

7. One phase of the systems development model (a) Testing of users (b) Training of users (c) Production of documentations (d) Evaluation of the new system

8. For a new system to be implemented it must be backed by: i. the users ii. the systems development team iii. the managers of the company (a) I and ii only are needed to support the new system (b) I and iv only are needed to support the new system (c) I and iii only are needed to support the new system (d) I, ii and iii must support the new system

9. Which is true about Algorithms (a) Algorithms can easily be converted into program codes (b) Algorithms are written in formal notation (c) Algorithms provide a sequence of steps that will solve a problem (d) Algorithms are only used for maths and computer related problems

10. Pseudocode differs from algorithm in that (a) Pseudocodes are precise while algorithms are not (b) Pseudocode must solve the problem whereas algorithm does not have to (c) Pseduocode are pre-cursor to computer program codes (d) Algorithm is written by the problem solver while pseudocode is written by the code designer

11. If an area is fully understood by the users and developers, then which model would be suitable (a) Problem solving (b) Spiral (c) Programming (d) HTML Development

12. Consider the following codes: Begin Line 1 sum = 0 Line 2 input num1, num2, num3 Line 3 sum = num1+num2 +num3 Line 4 Print sum End If num1 = 2, num2

= 3, num3 = 4; and the codes were executed as is what would be the result: (a) 0 (b) Error (c) 9 (d) Program failure

13. Consider the following codes:BeginLine 1 sum = 0 Line 2 input num1, num2, num3 Line 3 sum = num1+num2 +num3 Line 4 Print “Your total is”, sum End If num1 = 8, num2 = 7, num3 = 9; and the codes were executed as is what would be the result: (a) Your total is 24 (b) Your total is sum (c) "Your total is", 24 (d) "Your total is 24"

14. In a pseudocode the statements used must (a) Solve the problem (b) Be properly structured and indented to illustrate flow of logic and control (c) be Literally correct (d) Have good English meaning and be recognizable English words

15. One conversion method that can be used by the system development team while the users in the office are still in training is (a) Phased conversion (b) Direct conversion (c) Pilot conversion (d) Systematic conversion

16. At what stage of the SDLC is the complete system checked for compatibility. (a) The design phase (b) Evaluation phase (c) Implementation phase (d) Maintenance phase

17. Which of the following provides an accurate description of a problem (a) A difficulty that cannot be solved (b) A difficulty that prevents the accomplishment of some objective (c) A difficulty that arise from some existing dilemma (d) A challenge that hinders progress, that may have no solution

18. Which is most accurate about problem statements (a) The problem statement help the users to communicate the problem (b) The problem statement is just a summary of the issues faced (c) The problem statement is the backbone of the feasibility and the problem solution (d) It includes the symptoms and constraints, and so often mislead users

19. The word INPUT is used to (a) Set aside some memory to run the program (b) Set aside some memory area to store the value of the constant inputted (c) Set aside some memory area to store the value of the variable entered (d) Used to give the variable its value

20. Last step in process of problem solving is to (a) design a solution (b) define a problem **(c) practicing the solution** (d) organizing the data

21. Rectangle shaped symbol is used to show the **(a) carried operations** (b) errors (c) decisions (d) defined variables

22. Sequence of the statements are shown in the order which are from **(a) left to right** (b) right to left (c) top to down (d) down to top

23. Algorithm is made up of (a) sequence to print data (b) selection (c) repetition **(d) all of above**

24. Program statement in programming language 'PASCAL' usually ends with a (a) colon **(b) semicolon** (c) comma (d) apostrophe

25. Set of diagrams and notes that accompany program implementation are known as (a) program execution (b) program planning **(c) program documentation** (d) program existence

26. Ideal way to show a top down algorithm is to draw a (a) sequence diagram (b) selection diagram (c) reference diagram **(d) structure diagram**

27. Lines and arrows in the system flowchart are used to represent the **(a) flow of data** (b) order of operations (c) order of input (d) order of processing

28. Condition for the selection of statement written in the (a) circle above it **(b) box above it** (c) triangle above it (d) line above it

29. Input or output operations, working on main modules and execution of program is classified as **(a) outline program flowchart** (b) detailed program flowchart (c) canceled flowchart (d) dynamic flowchart

30. All variables used in any program must be declared first in (a) FIX statement **(b) VAR statement** (c) REF statement (d) REP statement

31. Writing a program into a suitable language is called **(a) program coding** (b) program instructions (c) program debugging (d) program testing

32. The word ____________comes from the name of a Persian mathematician Abu Ja'far Mohammed ibn-i Musa al Khowarizmi. (a) Flowchart (b) Pseudocode **(c) Algorithm** (d) Programming

33. This characteristic often draws the line between what is feasible and what is impossible. **(a) Performance** (b) System Evaluation (c) Modularity (d) Reliability

34. The time that depends on the input: an already sorted sequence that is easier to sort. (a) Process (b) Evaluation **(c) Running** (d) Input

35. Which of the following is incorrect? Algorithms can be represented: (a) as pseudo codes **(b) as syntax** (c) as programs (d) as flowcharts

36. A system wherein items are added from one and removed from the other end. (a) Stack **(b) Queue** (c) Linked List (d) Array

37. The process of drawing a flowchart for an algorithm is called (a) Performance (b) Evaluation (c) Algorithmic Representation **(d) Flowcharting**

38. Actual instructions in flowcharting are represented in (a) Circles **(b) Boxes** (c) Arrows (d) Lines

39. A box that can represent two different conditions in flow chart is a (a) Rectangle **(b) Diamond** (c) Circle (d) Parallelogram

40. A detailed flowchart is called ______ (a) Stack (b) Macro **(c) Micro** (d) Union

41. A flowchart that outlines the main segments of a program is (a) Queue **(b) Macro** (e) Micro (d) Union

42. The _______ provides pictorial representation of given problem. (a) Algorithm **(b) Flowchart** (c) Pseudocode (d) All of these

43. The linker (a) Is similar to interpreter (b) Uses source code as its input (**c) Is required to create a load module** (d) None of the mentioned

44. What shape represents the start and end of a flowchart? **(a) Oval** (b) Rectangle (c) Diamond (d) Square

45. Which of the following processes the source code before it goes to the compiler? (a) compiler (b) simulator **(c) pre-processor** (d) emulator

#D