FREE EDITION · NOTES + 3 SAMPLE MCQS

CUET · COMPUTER SCIENCE · CLASS XI · CODE 308

Introduction to Problem Solving

CUET unit: Introduction to Problem Solving

By **UniDrill** · NCERT-grounded study material

WWW.UNIDRILL.IN

📌 Snapshot

- Problem solving with computers follows a systematic process: analyse the problem, develop an algorithm, code it, then test and debug — the core computational thinking pipeline.
- An algorithm has defined characteristics and two standard representations (flowchart and pseudocode), both directly testable in CUET as definition/application questions.
- Flow of control comes in three forms — sequence, selection, and repetition/iteration — and NTA sets statement-based and case-based questions on each.
- Verification of algorithms (dry run) and comparison on the basis of time complexity and space complexity are conceptual distinctions NTA frequently tests.
- Decomposition is a strategy for complex problem solving, linking back to real-world software design (e.g., railway reservation system).

📖 Detailed Notes

2.1 Core concepts

- **Problem solving defined:** Problem solving is the process of identifying a problem, developing an algorithm for the identified problem, and finally implementing the algorithm to develop a computer program. The success of a computer in solving a problem depends on how correctly and precisely the problem is defined, the solution (algorithm) is designed, and the solution is implemented using a programming language. (NCERT §4.1, p. 61)
- **GIGO:** The correctness of the output a computer gives depends upon the correctness of the input provided — popularly stated as "Garbage In Garbage Out." (NCERT §4.2 sidebar, p. 62)
- **Four steps for problem solving:** The key steps required for solving a problem using a computer are: (1) Analysing the problem, (2) Developing an Algorithm, (3) Coding, and (4) Testing and Debugging. These are depicted as a cyclic process in Figure 4.1. (NCERT §4.2, p. 62)
- **Analysing the problem (§4.2.1):** Requires reading and analysing the problem statement carefully to list the principal components of the problem and decide the core functionalities that the solution should have. By analysing a problem, the

programmer can figure out the inputs the program should accept and the outputs it should produce. (NCERT §4.2.1, p. 62)

- **Developing an algorithm (§4.2.2):** It is essential to devise a solution before writing program code. The solution is represented in natural language and is called an algorithm. A more than one algorithm is possible for a given problem; the most suitable one should be selected. The algorithm is a tentative plan that is refined until it captures all aspects of the desired solution. (NCERT §4.2.2, p. 62–63)
- **Coding (§4.2.3):** After finalising the algorithm, it is converted into the format understandable by the computer using a high-level programming language. Documenting the coding procedures followed is equally important for future reference. (NCERT §4.2.3, p. 63)
- **Testing and Debugging (§4.2.4):** The program must meet user requirements, respond within expected time, and generate correct output for all possible inputs. Syntactical errors produce no output; incorrect output signals logical errors. Software industry follows standardised testing methods: unit/component testing, integration testing, system testing, and acceptance testing. Errors found are debugged and the program retested until all errors are removed. Maintenance of the solution involves fixing problems faced by the user and adding/modifying features after delivery. (NCERT §4.2.4, p. 63–64)
- **Algorithm defined:** An algorithm is a finite sequence of steps required to get the desired output. It has a definite beginning, a definite end, and consists of a finite number of steps. The term "Algorithm" is traced to Persian astronomer and mathematician Abu Abdullah Muhammad ibn Musa Al-Khwarizmi (c. 850 AD). (NCERT §4.3, p. 64)
- **Why algorithms are needed:** An algorithm acts as the roadmap (building block) of a computer program. Writing an algorithm before coding increases reliability, accuracy, and efficiency of the solution. Once an algorithm is correct, the computer program written from it will run correctly every time. (NCERT §4.3.1, p. 64–65)
- **Characteristics of a good algorithm:** (a) Precision — steps are precisely stated; (b) Uniqueness — results of each step are uniquely defined and depend only on the input and results of preceding steps; (c) Finiteness — the algorithm always stops after a finite number of steps; (d) Input — the algorithm receives some input; (e) Output — the algorithm produces some output. (NCERT §4.3.1 (A), p. 65)
- **Representation of algorithms (§4.4):** Two common methods are flowchart and pseudocode. Either method should: (i) showcase the logic of the problem solution, excluding implementational details; (ii) clearly reveal the flow of control during execution. (NCERT §4.4, p. 65–66)
- **Flowchart (§4.4.1):** A flowchart is a visual/diagrammatic representation of an algorithm using boxes, diamonds, and other shapes connected by arrows. Each shape represents a step in the solution process; the arrow represents the order or link among the steps. Standardised symbols are used: oval/rounded rectangle for

Start/End (Terminator), rectangle for Process (Action Symbol), parallelogram for Input/Output (Data symbol), diamond for Decision, and arrow for flow of control. (NCERT §4.4.1, p. 66)

- **Pseudocode (§4.4.2):** A pseudocode is a non-formal language (intended for human reading, not computer execution) that helps programmers write an algorithm. It is a detailed description of instructions a computer must follow in a particular order. No specific standard exists. The word "pseudo" means "not real," so pseudocode means "not real code." Commonly used keywords include: INPUT, COMPUTE, PRINT, INCREMENT, DECREMENT, IF/ELSE, WHILE, TRUE/FALSE. Benefits: helps safeguard against leaving out important steps and allows non-programmers to review the logic. (NCERT §4.4.2, p. 68–69)
- **Flow of control (§4.5):** Depicts how events flow as represented in a flowchart. Events can flow in a sequence, branch on a decision, or repeat for a finite number of times. (NCERT §4.5, p. 70)
- **Sequence (§4.5.1):** All steps are executed one after another; every statement executes in the order written. (NCERT §4.5.1, p. 70)
- **Selection (§4.5.2):** One of the alternatives is selected based on the outcome of a condition. Conditionals use IF/ELSE structure. True or false values from conditions are called binary values. In programming languages, 'otherwise' is represented using the Else keyword, giving rise to the if-else block. (NCERT §4.5.2, p. 70–72)
- **Repetition/Iteration (§4.5.3):** A loop means execution of some program statements repeatedly till a specified condition is satisfied. Also known as iteration. When the number of repetitions is unknown beforehand, the WHILE construct is used. (NCERT §4.5.3, p. 74–76)
- **Verifying algorithms (§4.6):** The method of taking an input and manually running through all the steps of the algorithm is called a dry run. A dry run helps to: (1) identify any incorrect steps in the algorithm; (2) figure out missing details or specifics. All possible input values must be tested to ensure correctness. (NCERT §4.6, p. 77–79)
- **Comparison of algorithms (§4.7):** There can be more than one algorithm for a given problem. Algorithms are compared and analysed on the basis of the amount of processing time they need to run (time complexity) and the amount of memory needed to execute the algorithm (space complexity). The choice between algorithms is made depending on their efficiency in terms of time complexity and space complexity. (NCERT §4.7, p. 79–80)
- **Coding (§4.8):** Once an algorithm is finalised, it is coded in a high-level programming language. Syntax is the set of rules or grammar governing statement formulation (spellings, order of words, punctuation, etc.). Machine language (0s and 1s) is directly understood by hardware but difficult for humans. High-level languages (FORTRAN, C, C++, Java, Python, etc.) are close to natural languages, portable (can run on different computers with little or no modification), but require translation to

machine language via a compiler or interpreter. A program written in a high-level language is called source code. (NCERT §4.8, p. 80–81)

- **Decomposition (§4.9):** When a problem is complex (solution not directly derivable), it is decomposed into simpler sub-problems. Each sub-problem can be solved independently and by different persons or teams. The sub-problems are then combined logically to obtain the solution to the main problem. The spirit of decomposition is "divide and conquer." Examples: weather forecasting, events management, delivery management systems, railway reservation system. (NCERT §4.9, p. 81–82)

2.2 Definitions to memorise

Term	Definition	Page
Algorithm	A finite, step-by-step sequence of well-defined instructions that, when followed correctly, leads to the desired result; has a definite beginning, definite end, and finite number of steps.	64
Flowchart	A visual/diagrammatic representation of an algorithm made up of boxes, diamonds, and other shapes connected by arrows, where each shape represents a step and each arrow represents the order among steps.	66
Pseudocode	A non-formal, human-readable description of instructions a computer must follow in a particular order; uses keywords like INPUT, COMPUTE, PRINT, IF/ELSE, WHILE; cannot be executed directly by a computer.	68
Dry Run	The method of taking an input and manually running through all steps of the algorithm to verify its correctness and identify incorrect or missing steps.	77
Time Complexity	The amount of processing time an algorithm needs to run; used to compare and choose between algorithms.	80
Space Complexity	The amount of memory needed to execute an algorithm; used alongside time complexity to compare algorithms.	80
Syntax	The set of rules or grammar that governs the formulation of statements in a programming language (spellings, order of words, punctuation, etc.).	80
Source Code	A program written in a high-level language that must be translated to machine language using a compiler or interpreter before the computer can execute it.	81
Decomposition	The strategy of breaking down a complex problem into smaller, simpler sub-problems that are solved independently and then combined to solve the original problem.	81
GIGO	"Garbage In Garbage Out" — the correctness of the output depends upon the correctness of the input provided.	62

Term	Definition	Page
Iteration/Loop	Execution of some program statements repeatedly till a specified condition is satisfied; also called repetition.	75
Selection	A flow-of-control construct where one of the alternatives is selected based on the outcome (true/false) of a condition, implemented using if-else.	71
Sequence	Flow-of-control where all statements execute one after another in the order written	70
Precision (algorithm)	Characteristic that every step of the algorithm is precisely stated	65
Uniqueness (algorithm)	Characteristic that the result of each step depends only on the input and results of preceding steps	65
Finiteness (algorithm)	Characteristic that the algorithm terminates after a finite number of steps	65
Al-Khwarizmi	Persian astronomer and mathematician (c. 850 AD) from whose name the term "algorithm" is derived	64
WHILE construct	Repetition construct used when the number of iterations is not known in advance	76
Counter-controlled loop	Loop driven by a counter variable when the number of iterations is known in advance	75
Conditional / IF-ELSE	Selection construct that executes one of two branches based on a Boolean condition	71–72
Machine language	Language of 0s and 1s directly understood and executed by computer hardware	80
High-level language	Language close to natural language (e.g., Python, C++) needing translation before execution	80
Portability	Property of high-level language code that lets it run on different computers with little or no modification	80
Debugging	Process of locating and fixing errors found during testing	64
Maintenance	Post-delivery activity of fixing problems and adding features to the deployed program	64

2.3 Diagrams / processes to remember

- **Figure 4.1 (p. 62) — Problem Solving Steps (cyclic diagram):** Four steps shown as a cycle: 1. Analysing the Problem → 2. Developing an Algorithm → 3. Coding → 4. Testing and Debugging → back to start. Students must know the correct order of these steps.
- **Table 4.1 (p. 66) — Flowchart symbols:** Five symbols to memorise: (1) Oval/Rounded rectangle = Start/End (Terminator); (2) Rectangle = Process (Action

Symbol); (3) Diamond = Decision (yes/no or true/false branching); (4) Parallelogram = Input/Output (Data symbol); (5) Arrow = Flow of control connector.

- **Figure 4.2 (p. 67) — Flowchart to calculate square of a number:** Shows simple sequence: Start → Input num → square = num*num → Print square → Stop. Illustrates a pure sequence flow with no branching.
- **Figure 4.8 (p. 72) — Flowchart to check even or odd:** Illustrates selection using a diamond (Is num1 mod 2 == 0?) branching to "Print Even" (Yes) or "Print Odd" (No). Key example of if-else in a flowchart.
- **Figure 4.10 (p. 75) — Flowchart to calculate average of 5 numbers:** Illustrates repetition/iteration with a counter (count < 5 as loop condition). Counter increments inside the loop; average computed after loop exits.
- **Figure 4.12 (p. 82) — Railway reservation system decomposition:** Shows six sub-problems of the system: Trains' information, Reservation information, Information about staff/security/infrastructure, Food service, Billing service, Other details about railways. Canonical example of decomposition.

2.4 Common confusions / NTA trap points

- **Flowchart symbol mix-up:** Students frequently swap the parallelogram (Input/Output) with the rectangle (Process). NTA often provides a "match the symbol to function" question — remember: parallelogram = data/I/O; rectangle = action/process; oval = terminator.
- **Pseudocode vs. source code:** Pseudocode is not executable by the computer; source code (high-level language) is. NTA distractors often call pseudocode "informal source code" or claim it can be run directly — both are wrong.
- **Algorithm characteristics:** "Effectiveness" is not listed as a characteristic in this NCERT chapter; the five listed are Precision, Uniqueness, Finiteness, Input, and Output. Do not import external definitions.
- **Time complexity vs. space complexity:** Students mix these up. Time complexity = processing time; space complexity = memory. Algorithm (iv) for primality testing (using a list of primes up to 100) takes less time but more memory — a direct illustration of the trade-off.
- **Iteration terminology (NCERT §4.5.3, pp. 75-76).** In this NCERT chapter, repetition, iteration, and loop are used interchangeably. NTA may test whether a student knows that a WHILE construct is used when the number of repetitions is not known beforehand (Example 4.9, p. 76), whereas a counter-controlled loop is used when the count is known (Example 4.8, p. 75).
- **Algorithm input characteristic (NCERT §4.3.1(A), p. 65).** "Input" is one of the five characteristics, but NCERT does not require an algorithm to have non-zero input. An algorithm that prints the constant "Hello" still has input characteristic by convention but no input data — NTA distractor may claim "every algorithm must read input from user."

- **Compiler vs interpreter is outside this topic (NCERT §4.8, p. 81).** Although §4.8 mentions translation, the detailed compiler/interpreter distinction belongs to §1.7.3. NTA may try to test compiler details here — keep answers grounded in problem-solving scope.
- **Decomposition is NOT a loop construct (NCERT §4.9, p. 81).** Decomposition is a problem-solving strategy (divide and conquer), not a flow-of-control construct. NTA might list it alongside sequence/selection/iteration to mislead.
- **Dry run requires testing ALL possible input values (NCERT §4.6, p. 79).** Not just one. NTA may suggest a single test case is sufficient — false.
- **Pseudocode keywords (NCERT §4.4.2, p. 69).** INPUT, COMPUTE, PRINT, INCREMENT, DECREMENT, IF/ELSE, WHILE, TRUE/FALSE — these are conventions, not standardised keywords. There is no universal pseudocode standard.
- **Source code = high-level program (NCERT §4.8, p. 81).** Source code must be translated; it is NOT the machine code itself. NTA distractor: "source code is directly executed by hardware" — false.
- **Time and space complexity may trade off (NCERT §4.7, p. 80).** Saving time often costs more memory and vice versa — the primality example shows this trade-off directly.

🎯 Practice MCQs

Q1. Which of the following correctly lists the four steps of problem solving using a computer, in the right sequence?

A. Coding → Analysing the Problem → Developing an Algorithm → Testing and Debugging

B. Analysing the Problem → Developing an Algorithm → Coding → Testing and Debugging

C. Developing an Algorithm → Analysing the Problem → Testing and Debugging → Coding

D. Analysing the Problem → Coding → Developing an Algorithm → Testing and Debugging

Q2. Consider the following statements about a good algorithm: 1. The results of each step are uniquely defined and depend only on the input and the results of preceding steps. 2. The algorithm must always stop after a finite number of steps. 3. The algorithm may or may not produce an output. Which of the statements given above are correct as per the NCERT chapter?

A. 1 and 2 only

B. 2 and 3 only

C. 1 and 3 only

D. 1, 2 and 3

Q3. Which flowchart symbol is used to represent a decision or branching point, where a yes/no or true/false question is asked and the path splits into two branches?

A. Rectangle (Process box)

B. Parallelogram (Data symbol)

C. Oval/Rounded rectangle (Terminator)

D. Diamond

📊 PYQ Alignment

Problem solving fundamentals appear consistently across CUET Computer Science papers, with questions drawn predominantly from algorithm characteristics, flowchart symbol identification (Table 4.1), pseudocode vs. source code distinctions, and flow-of-control constructs (sequence, selection, iteration). NTA particularly favours match-the-following questions on flowchart symbols and statement-based questions testing whether students can identify the correct property (precision, uniqueness, finiteness) violated by a faulty algorithm. Use the PYQ archive for Computer Science for further drilling.