



Gurukul Technicals Institute

Computer education · Robotics · Science projects · Delhi

FREE EBOOK

PROGRAMMING AND WEB

Flowcharts, Algorithms and Logic Building

Think like a programmer before you write any code

INSIDE THIS BOOK

- What is an algorithm?
- Properties of a good algorithm
- Pseudocode
- Flowcharts
- ...and 7 more chapters, practice questions and a glossary

Beginner

Students starting any programming language

engineergurukul.com

+91 97116 99511

Flowcharts, Algorithms and Logic Building

Learn to plan a program before you type it. You will write clear steps, pseudocode and flowcharts, check them with dry runs and trace tables, and work through 25 solved problems and a set of logic puzzles that build the thinking every programming language needs.

WHO IT IS FOR	Students starting any programming language
LEVEL	Beginner
SUBJECT AREA	Programming and web
WHAT IS INSIDE	11 chapters, practice questions with answers, and a glossary

CONTENTS

1	What is an algorithm?	4
2	Properties of a good algorithm	5
3	Pseudocode	6
4	Flowcharts	7
5	Sequence, selection and iteration	9
6	Dry runs and trace tables	10
7	Solved problems: sequence and decisions	12
8	Solved problems: loops	15
9	Solved problems: lists, searching and sorting	18
10	How fast is an algorithm?	21
11	Logic puzzles	22
	Practice questions	23

Answers**24****Glossary****25**

Free ebook: you may read, print and share it for learning; it may not be sold or changed. © 2026 Gurukul Technicals Institute. First edition, October 2026. Contact details are on the back cover.

What is an algorithm?

Think about making a cup of chai. You boil water, add tea leaves, add milk and sugar, let it boil, strain it into a cup and serve. If you add the milk after straining, or forget the tea leaves, you do not get chai. The steps, and their order, matter.

An **algorithm** is a set of clear, ordered steps that solves a problem. The word comes from the name of al-Khwarizmi, a Persian mathematician who wrote about methods of calculation more than a thousand years ago. You already follow many algorithms:

- Recharging a mobile phone: open the app, choose the plan, pay, wait for the message.
- Long division: divide, multiply, subtract, bring down the next digit, repeat.
- Finding a word in a dictionary: open near the right letter, then move forward or back.

Input, process, output

Almost every algorithm has three parts.

Part	Meaning	Chai example	Shop bill example
Input	what you start with	water, leaves, milk, sugar	price and quantity
Process	what you do with it	boil, mix, strain	multiply them
Output	the result	a cup of chai	the bill amount

Why plan before you code?

A computer does exactly what it is told, nothing more and nothing less. It cannot guess what you meant. If your steps are wrong, the program will be wrong, however good your typing is. Programmers therefore solve the problem first, on paper, in plain language. Only then do they write it in C, Java or Python. A good algorithm can be turned into any programming language. A bad one cannot be saved by any language.

DID YOU KNOW

Algorithms run all around you. A map app finding the shortest road from Karol Bagh to Connaught Place, a bank checking your UPI PIN and a shopping site sorting by price all follow algorithms.

TRY THIS

Write the steps for crossing a road safely. Then read them as if you were a robot that knows nothing. Did you say which way to look first? Did you say what to do if a vehicle is coming?

CHAPTER 2

Properties of a good algorithm

Not every list of steps is a good algorithm. "Cook something tasty" is a wish, not an algorithm. A proper algorithm has five properties.

Property	Meaning	Bad example
Input	It takes zero or more values to work on	None: some algorithms need no input
Output	It gives at least one result	Steps that never report an answer
Definiteness	Every step is clear, with only one meaning	"Add some sugar"
Finiteness	It stops after a limited number of steps	"Keep adding 1 forever"
Effectiveness	Every step can actually be done	"Write down the last digit of pi"

Pi never ends, so its "last digit" does not exist. A step that cannot be carried out makes the whole algorithm useless.

REMEMBER

If a step could be understood in two ways, rewrite it until it can be understood in only one.

Writing steps in plain words

The simplest way to write an algorithm is as numbered steps in plain English. Many Indian textbooks write it like this, for finding the average of three marks:

1. Start.
2. Read the three marks a, b and c.
3. Add them: $\text{total} = a + b + c$.
4. Divide: $\text{average} = \text{total} / 3$.
5. Print total and average.
6. Stop.

Some rules make plain-word steps easy to follow:

1. Number every step.

2. Begin with Start and end with Stop.
3. Use one action per step.
4. Give each value a short, meaningful name, such as `total` or `marks`, and use the same name every time.
5. When a step depends on a condition, say both what happens if it is true and what happens if it is false.

TIP

Test your steps with real numbers. If Ritu's marks are 72, 85 and 90, the steps above should give a total of 247 and an average of 82.33. If they do not, a step is wrong.

CHAPTER 3

Pseudocode

Plain words are friendly but can get long. Program code is exact but full of brackets and semicolons. **Pseudocode** sits in between. It is "pretend code": structured like a program, written in simple English, and not tied to any language. There is no single official pseudocode, so the most important rule is to be consistent. This book uses these conventions.

Pseudocode	Meaning
START, STOP	where the algorithm begins and ends
INPUT <code>x</code>	read a value from the user into <code>x</code>
PRINT <code>x</code>	show a value or a message
SET <code>x = 5</code>	store 5 in <code>x</code>
IF ... THEN ... ELSE ... END IF	choose between two paths
WHILE ... END WHILE	repeat while a condition is true
FOR <code>i = 1 TO n</code> ... END FOR	repeat a fixed number of times
<code>a MOD b</code>	remainder after <code>a</code> is divided by <code>b</code>
<code>a DIV b</code>	whole-number division, with the remainder dropped
<code>A[i]</code>	item number <code>i</code> of a list <code>A</code> , starting from 1

So `17 MOD 5` is 2 and `17 DIV 5` is 3. After the word SET, the sign `=` means "store". After IF or WHILE, it means "is equal to". Lines inside an IF or a loop are pushed to the right, which is called **indentation**, so you can see where each block starts and ends.

Here is the average algorithm again, as pseudocode:

```
START
INPUT a, b, c
SET total = a + b + c
SET average = total / 3
PRINT total, average
STOP
```

REMEMBER

Lists in this book’s pseudocode start at 1, because that is how people count. C, Java and Python start counting at 0, so the first item is `A[0]`. Adjust when you turn pseudocode into code.

Words to use with conditions

Conditions compare values with `=`, `<>` (not equal), `<`, `>`, `<=` and `>=`. You can join them with `AND` (both must be true), `OR` (at least one must be true) and `NOT` (turns true into false).

TRY THIS

Write pseudocode that reads the price of one notebook and the number bought, and prints the bill. Test it with ! 35 and 6 notebooks. You should get ! 210.

CHAPTER 4

Flowcharts

A **flowchart** is a picture of an algorithm. Boxes of different shapes show the steps, and arrows show the order. Many people find a picture easier to follow than text, especially when the path splits. In this book we draw flowcharts with keyboard characters. On paper you would draw the real shapes listed below.

Real shape	Name	Used for	In this book
Oval (rounded ends)	Terminal	Start and Stop	(START)
Rectangle	Process	a calculation or action	[sum = 0]
Parallelogram (slanted box)	Input/ Output	reading or printing	/ INPUT n /
Diamond	Decision	a yes/no question	< n = 0 ? >
Small circle	Connector	joins parts of a chart	(A)

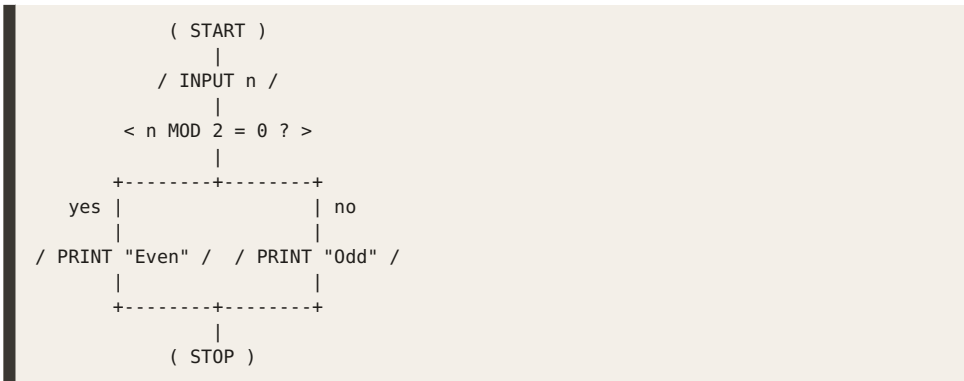
Real shape	Name	Used for	In this book
Arrow	Flow line	shows the next step	lines, with v or > as arrowheads

Rules for neat flowcharts

1. Every flowchart has exactly one Start and at least one Stop.
2. Flow goes from top to bottom, or left to right, wherever possible.
3. A process box has one arrow in and one arrow out.
4. A decision diamond has one arrow in and two arrows out, labelled yes and no.
5. Arrows should not cross. If a chart gets too big for the page, break it with connectors: a circle marked A at the bottom of one page continues at the circle marked A on the next.

An example: even or odd

A number is even if it leaves no remainder when divided by 2.



Follow it with $n = 14$. The remainder is 0, so take the yes path and print "Even". With $n = 7$ the remainder is 1, so take the no path and print "Odd". Both paths meet again before Stop.

TIP

Draw flowcharts in pencil first. Leave space between boxes, because you will almost always need to add a step you forgot.

DID YOU KNOW

Flowcharts are used far outside programming. Banks draw them for loan approvals, hospitals for emergency steps and factories for quality checks.

Sequence, selection and iteration

Here is a surprising fact: every algorithm, however large, is built from just three patterns. Learn these three and you can read any flowchart.

Sequence: one step after another

In a **sequence**, steps run in order, each exactly once. The average-of-three-marks algorithm is pure sequence. In a flowchart it is a straight line of boxes.

EXAMPLE

Converting temperature. INPUT c , then SET $f = c * 9 / 5 + 32$, then PRINT f . For a body temperature of 37 degrees Celsius, $f = 98.6$ degrees Fahrenheit.

Selection: choosing a path

In **selection**, a condition decides which steps run. In a flowchart it is a decision diamond. There are three common forms.

- **IF ... THEN:** do something only when the condition is true.
- **IF ... THEN ... ELSE:** choose between two paths, as in even or odd.
- **IF ... ELSE IF ... ELSE:** choose one of many paths, as in giving a grade.

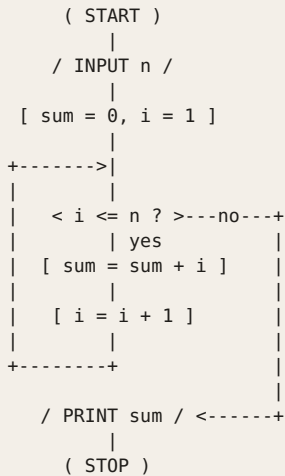
```
IF marks >= 40 THEN
  PRINT "Pass"
ELSE
  PRINT "Fail"
END IF
```

Here 40 is just an example of a pass mark. Your own exam may use a different one.

Iteration: repeating steps

In **iteration**, also called a **loop**, steps repeat. In a flowchart, an arrow goes back up to an earlier box. Every loop needs three things: a starting value, a condition that is checked each time, and a step that changes something so the loop can finally end.

This flowchart adds the numbers from 1 to n :



The arrow on the left carries the flow back up to the question. When i grows bigger than n , the answer is no and the flow leaves the loop.

NOTE

If you forget $i = i + 1$, then i never changes, the question always says yes, and the loop runs forever. This is called an **infinite loop**, and it is one of the most common loop mistakes. On a real computer it can freeze the program.

FOR loops are used when you know how many times to repeat. **WHILE** loops are used when you repeat until something happens, such as running out of digits.

CHAPTER 6

Dry runs and trace tables

How do you know an algorithm is right before you write code? You test it by hand. A **dry run** means following the algorithm step by step with real values, exactly as a computer would, without guessing. A **trace table** is a table where you record the value of every variable after each step. A **variable** is simply a named box that holds a value, such as `sum` or `n`.

A worked dry run

This algorithm adds the digits of a number.

```

START
INPUT n
SET sum = 0
WHILE n > 0
    SET digit = n MOD 10
    SET sum = sum + digit
    SET n = n DIV 10
END WHILE
PRINT sum
STOP

```

Dry run with $n = 472$:

Round	digit	sum	n after the round
Start	none	0	472
1	2	2	47
2	7	9	4
3	4	13	0

After round 3, n is 0, so the condition $n > 0$ is false and the loop stops. The output is 13, and $4 + 7 + 2$ is indeed 13. The algorithm works for this input.

How to make a trace table

1. Make one column for each variable, and one for output if needed.
2. Write the starting values in the first row.
3. Go through the algorithm one line at a time. Each time a variable changes, write its new value.
4. Each time you reach a condition, work out whether it is true or false before moving on.
5. Stop when the algorithm reaches STOP, then compare the output with the answer you expected.

TIP

Test special cases, not just ordinary ones. For the digit-sum algorithm, try $n = 0$ (the loop never runs and the answer is 0, which is right) and a number such as 1000 (the zeros must not upset it).

REMEMBER

A dry run finds mistakes when they are cheapest to fix: on paper, before any code exists. Examiners often ask for trace tables, so practise drawing them neatly.

Solved problems: sequence and decisions

Problems 1 to 11 use sequence and selection. Read each problem, try it yourself, and then compare with the solution.

Problem 1: Total and average of three marks

See the pseudocode in the Pseudocode chapter. Ritu's 72, 85 and 90 give a total of 247 and an average of 82.33.

Problem 2: Cost of tiling a floor

```
INPUT length, width, rate
SET area = length * width
SET cost = area * rate
PRINT area, cost
```

A room in Jaipur is 12 feet by 10 feet, and tiles cost ₹ 55 per square foot. Area = 120 square feet, cost = ₹ 6,600.

Problem 3: Simple interest

```
INPUT p, r, t
SET si = p * r * t / 100
SET amount = p + si
PRINT si, amount
```

For ₹ 10,000 at an example rate of 8% a year for 2 years, interest = ₹ 1,600 and amount = ₹ 11,600.

Problem 4: Swap two numbers

```
INPUT a, b
SET temp = a
SET a = b
SET b = temp
PRINT a, b
```

Why the third box, `temp`? If you write `SET a = b` first, the old value of `a` is lost. It is like swapping the contents of two full glasses: you need an empty third glass.

Problem 5: Celsius to Fahrenheit

`INPUT c, SET f = c * 9 / 5 + 32, PRINT f`. 100 degrees Celsius gives 212, and 0 gives 32.

Problem 6: Even or odd

```
INPUT n
IF n MOD 2 = 0 THEN
    PRINT "Even"
ELSE
    PRINT "Odd"
END IF
```

The flowchart is in the Flowcharts chapter.

Problem 7: Largest of three numbers

```
INPUT a, b, c
IF a >= b AND a >= c THEN
    PRINT a
ELSE IF b >= c THEN
    PRINT b
ELSE
    PRINT c
END IF
```

If a is not the largest, the answer must be b or c, so one more comparison settles it. Try it with two equal numbers, such as 7, 7 and 3: it still prints 7.

Problem 8: Leap year

A year is a leap year if it divides exactly by 400, or if it divides exactly by 4 but not by 100.

```
INPUT year
IF year MOD 400 = 0 THEN
    PRINT "Leap year"
ELSE IF year MOD 100 = 0 THEN
    PRINT "Not a leap year"
ELSE IF year MOD 4 = 0 THEN
    PRINT "Leap year"
ELSE
    PRINT "Not a leap year"
END IF
```

2024 and 2000 are leap years. 1900 is not. The order of the tests matters: check 400 before 100, and 100 before 4.

Problem 9: Grade from marks

Using a made-up scheme: 90 and above A, 75 to 89 B, 60 to 74 C, 40 to 59 D, below 40 E.

```

INPUT marks
IF marks >= 90 THEN
    PRINT "A"
ELSE IF marks >= 75 THEN
    PRINT "B"
ELSE IF marks >= 60 THEN
    PRINT "C"
ELSE IF marks >= 40 THEN
    PRINT "D"
ELSE
    PRINT "E"
END IF

```

Because the tests run from the top, a mark of 82 fails the first test and passes the second, so we never need to write "marks < 90".

Problem 10: Positive, negative or zero

```

INPUT n
IF n > 0 THEN
    PRINT "Positive"
ELSE IF n < 0 THEN
    PRINT "Negative"
ELSE
    PRINT "Zero"
END IF

```

Problem 11: Electricity bill with slabs

These rates are invented for practice and are not any real board's tariff: the first 100 units at ! 3 each, the next 100 at ! 5 each, and every unit above 200 at ! 7.

```

INPUT units
IF units <= 100 THEN
    SET bill = units * 3
ELSE IF units <= 200 THEN
    SET bill = 100 * 3 + (units - 100) * 5
ELSE
    SET bill = 100 * 3 + 100 * 5 + (units - 200) * 7
END IF
PRINT bill

```

For 250 units: $300 + 500 + 350 = \text{! } 1,150$.

TRY THIS

Dry run Problem 11 with 80 units and with 150 units. You should get ! 240 and ! 550.

Solved problems: loops

Problems 12 to 20 repeat steps. Most of them use the digit tricks `n MOD 10` (the last digit) and `n DIV 10` (the number without its last digit).

Problem 12: Sum of the first n numbers

```
INPUT n
SET sum = 0
FOR i = 1 TO n
    SET sum = sum + i
END FOR
PRINT sum
```

For $n = 10$ the answer is 55.

Problem 13: Multiplication table

```
INPUT n
FOR i = 1 TO 10
    PRINT n, " x ", i, " = ", n * i
END FOR
```

Problem 14: Factorial

The factorial of n , written $n!$, is $1 \times 2 \times 3 \times \dots \times n$. So $5! = 120$.

```
INPUT n
SET fact = 1
FOR i = 1 TO n
    SET fact = fact * i
END FOR
PRINT fact
```

Notice that `fact` starts at 1, not 0. Starting at 0 would make every answer 0. For $n = 0$ the loop does not run at all and the answer is 1, which matches the rule that $0! = 1$.

Problem 15: Sum of digits

This is the algorithm traced in the Dry runs chapter: 472 gives 13.

Problem 16: Reverse a number

```
INPUT n
SET rev = 0
WHILE n > 0
    SET rev = rev * 10 + n MOD 10
    SET n = n DIV 10
END WHILE
PRINT rev
```

For 1234, rev grows 4, 43, 432, 4321. Note that 1200 reverses to 21, because zeros at the front of a number are not written.

Problem 17: Palindrome number

A **palindrome** reads the same both ways, like 121 or the word "malayalam".

```
INPUT n
SET original = n
SET rev = 0
WHILE n > 0
    SET rev = rev * 10 + n MOD 10
    SET n = n DIV 10
END WHILE
IF rev = original THEN
    PRINT "Palindrome"
ELSE
    PRINT "Not a palindrome"
END IF
```

We save the number in `original` because the loop destroys `n`.

Problem 18: Prime number

A **prime** is a whole number greater than 1 whose only factors are 1 and itself.

```

INPUT n
SET isPrime = TRUE
IF n < 2 THEN
    SET isPrime = FALSE
END IF
SET i = 2
WHILE i * i <= n AND isPrime = TRUE
    IF n MOD i = 0 THEN
        SET isPrime = FALSE
    END IF
    SET i = i + 1
END WHILE
IF isPrime = TRUE THEN
    PRINT "Prime"
ELSE
    PRINT "Not prime"
END IF

```

Why stop when $i * i$ passes n ? If n had a factor bigger than its square root, it would also have a partner factor smaller than the square root, and we would already have found that. For $n = 29$ we test only 2, 3, 4 and 5, then print "Prime". For $n = 21$ we stop at 3, because $21 \text{ MOD } 3$ is 0.

Problem 19: Fibonacci series

Each term is the sum of the two before it: 0, 1, 1, 2, 3, 5, 8, 13 and so on.

```

INPUT n
SET a = 0
SET b = 1
FOR i = 1 TO n
    PRINT a
    SET next = a + b
    SET a = b
    SET b = next
END FOR

```

Problem 20: Greatest common divisor (GCD)

Euclid's method, more than two thousand years old, replaces the pair (a, b) with $(b, \text{remainder})$ until the remainder is 0.

```

INPUT a, b
WHILE b > 0
    SET r = a MOD b
    SET a = b
    SET b = r
END WHILE
PRINT a

```

a	b	r
48	18	12
18	12	6
12	6	0
6	0	loop ends

The GCD of 48 and 18 is 6. The LCM (lowest common multiple) is then $48 \times 18 / 6 = 144$.

REMEMBER

Before a loop, set every total to 0 and every product to 1. Wrong starting values are another very common loop mistake.

CHAPTER 9

Solved problems: lists, searching and sorting

A **list**, also called an **array**, holds many values under one name. `A[1]` is the first item and `A[n]` is the last. Problems 21 to 25 work on lists.

A tea stall in Pune records its sales for one week, Monday to Sunday: ! 1,450, ! 1,720, ! 980, ! 2,100, ! 1,650, ! 2,400 and ! 1,900.

Problem 21: Largest value in a list

```

INPUT n, A[1..n]
SET max = A[1]
FOR i = 2 TO n
    IF A[i] > max THEN
        SET max = A[i]
    END IF
END FOR
PRINT max

```

Start by assuming the first value is the largest, then let any bigger value take its place. For the tea stall, max = ! 2,400 (Saturday).

Problem 22: Total, average and days above average

```
INPUT n, A[1..n]
SET total = 0
FOR i = 1 TO n
    SET total = total + A[i]
END FOR
SET avg = total / n
SET count = 0
FOR i = 1 TO n
    IF A[i] > avg THEN
        SET count = count + 1
    END IF
END FOR
PRINT total, avg, count
```

Total = ! 12,200, average = about ! 1,742.86, and 3 days (Thursday, Saturday and Sunday) are above average. Two loops are needed, because we cannot know the average until we have the total.

Problem 23: Linear search

Check each item in turn until you find the one you want.

```
INPUT n, A[1..n], key
SET pos = 0
SET i = 1
WHILE i <= n AND pos = 0
    IF A[i] = key THEN
        SET pos = i
    END IF
    SET i = i + 1
END WHILE
IF pos = 0 THEN
    PRINT "Not found"
ELSE
    PRINT "Found at position ", pos
END IF
```

Problem 24: Binary search

If the list is already sorted, there is a much faster way. Look at the middle item. If it is too small, throw away the left half. If it is too big, throw away the right half. Repeat.

```

INPUT n, A[1..n], key
SET low = 1
SET high = n
SET pos = 0
WHILE low <= high AND pos = 0
    SET mid = (low + high) DIV 2
    IF A[mid] = key THEN
        SET pos = mid
    ELSE IF A[mid] < key THEN
        SET low = mid + 1
    ELSE
        SET high = mid - 1
    END IF
END WHILE
PRINT pos

```

If pos is still 0 at the end, the key is not in the list.

Searching for 52 in the sorted list 3, 8, 15, 21, 27, 34, 40, 52, 66:

low	high	mid	A[mid]	action
1	9	5	27	too small, low = 6
6	9	7	40	too small, low = 8
8	9	8	52	found at position 8

Three looks instead of eight.

Problem 25: Bubble sort

Compare each pair of neighbours and swap them if they are in the wrong order. After each pass, the largest remaining value has "bubbled" to the end.

```

INPUT n, A[1..n]
FOR pass = 1 TO n - 1
    FOR i = 1 TO n - pass
        IF A[i] > A[i + 1] THEN
            SWAP A[i] and A[i + 1]
        END IF
    END FOR
END FOR

```

Sorting 5, 1, 4, 2, 8:

After	List
Pass 1	1, 4, 2, 5, 8
Pass 2	1, 2, 4, 5, 8

After	List
Pass 3	1, 2, 4, 5, 8 (no swaps)
Pass 4	1, 2, 4, 5, 8 (no swaps)

TIP

Passes 3 and 4 did nothing. A better version keeps a flag that notes whether any swap happened in a pass, and stops as soon as a whole pass makes no swap.

CHAPTER 10

How fast is an algorithm?

Two algorithms can both give the right answer, yet one can be thousands of times faster. **Complexity** is the study of how much work an algorithm does as its input grows. We do not measure it in seconds, because seconds depend on the computer. We count steps.

Counting steps

Think about linear search in a list of n items. In the worst case, when the item is last or missing, we look at all n items. Double the list and the work doubles. We call this **linear time**.

Binary search halves the list at every look. For 1,000 sorted items it needs at most 10 looks. For 10 lakh items it needs at most 20, because halving 10 lakh twenty times leaves less than one item. Doubling the list adds just one look. We call this **logarithmic time**.

Bubble sort compares neighbours again and again. For n items it can make $n \times (n - 1) / 2$ comparisons: 45 for 10 items, but about 5 lakh for 1,000 items. Ten times the data means about a hundred times the work. We call this **quadratic time**.

Big-O notation in plain words

Programmers write these growth rates with **Big-O notation**. The O stands for "order of", and it describes how the work grows in the worst case.

Big-O	Name	Example	Work for 1,000 items
$O(1)$	constant	reading <code>A[5]</code>	1 step
$O(\log n)$	logarithmic	binary search	about 10 steps
$O(n)$	linear	linear search, largest in a list	about 1,000 steps
$O(n^2)$	quadratic	bubble sort	about 5 lakh steps

$O(n^2)$ is read "order n squared".

EXAMPLE

Problem 12 adds 1 to n with a loop, which is $O(n)$. The formula $n \times (n + 1) / 2$ gives the same answer in one step, which is $O(1)$. For $n = 10$ both give 55, but for $n =$ one crore the formula wins easily. Better thinking beat a faster computer.

DID YOU KNOW

The square-root trick in Problem 18 is also about complexity. To test whether 10,007 is prime, checking every number up to 10,006 takes about ten thousand steps, while checking up to its square root takes about a hundred.

CHAPTER 11

Logic puzzles

Programming is really organised thinking. These classic puzzles train the same habits: track every possibility, work in small steps, and check your answer. Try each for at least five minutes before reading the solution.

Puzzle 1: Three switches

There are three switches outside a closed room. One of them controls an old-style filament bulb inside. You may flip the switches as you like, but you may open the door only once. How do you find the right switch?

Solution: Turn switch 1 on for ten minutes, then off. Turn switch 2 on and walk in. If the bulb is lit, it is switch 2. If it is off but warm, it is switch 1. If it is off and cold, it is switch 3. The trick is to use a second kind of information, heat, besides light.

SAFETY

Do not touch a bulb that has been on, at home or anywhere else. Filament bulbs get hot enough to burn. This puzzle is for thinking, not for trying.

Puzzle 2: River crossing

A farmer must take a fox, a hen and a sack of grain across a river. His boat carries him and one item. Left alone, the fox eats the hen, and the hen eats the grain.

Solution: Take the hen across. Come back. Take the fox across and bring the hen back. Take the grain across. Come back and take the hen. Seven crossings. The key move is bringing something back, which feels like going backwards but is not.

Puzzle 3: Two jugs

With a 5-litre jug, a 3-litre jug and a tap, measure exactly 4 litres.

Solution: Fill the 5-litre jug. Pour from it into the 3-litre jug, leaving 2 litres. Empty the 3-litre jug and pour the 2 litres into it. Fill the 5-litre jug again and top up the 3-litre jug, which needs only 1 litre. Exactly 4 litres remain in the big jug. A trace table of the two jugs makes this easy to check.

Puzzle 4: The heavy coin

Nine coins look the same, but one is heavier. With a balance scale, find it in two weighings.

Solution: Split them into three groups of three. Weigh group A against group B. If one side goes down, the heavy coin is in it. If they balance, it is in group C. Now take the three suspects and weigh one against another. The heavier one is the answer. If they balance, it is the third coin. This "divide into parts and throw away the parts you do not need" idea is the same one behind binary search.

TRY THIS

Find the next number: 2, 6, 12, 20, 30, ? Look at the gaps between the numbers first.

CHECK YOURSELF

Practice questions

1. Write the five properties of an algorithm, with one line on each.
2. Why is "add a little salt" not a definite step? Rewrite it so that it is definite.
3. Name the flowchart symbol used for (a) reading input, (b) a yes/no question, (c) Start and Stop.
4. What are the values of $29 \text{ MOD } 4$ and $29 \text{ DIV } 4$?
5. Name the three basic building blocks of every algorithm.
6. Write pseudocode to read the length of a side of a square and print its area and perimeter.
7. Write pseudocode to read a number and print whether it is divisible by both 3 and 5.
8. Write pseudocode to read the age of a person and print "Can vote" or "Cannot vote", using 18 as the age limit.
9. Write pseudocode to read two numbers and print the smaller one.
10. Write pseudocode to print the even numbers from 2 to 20.
11. Write pseudocode to count the digits in a positive whole number.
12. Make a trace table for the reverse-a-number algorithm (Problem 16) with $n = 507$.

13. Dry run the factorial algorithm (Problem 14) for $n = 4$. Show fact after each round.
14. Using the GCD algorithm, find the GCD of 84 and 36 and show each step in a table.
15. What is an infinite loop? Give one way to cause one by mistake.
16. Write pseudocode to find the smallest value in a list of n numbers.
17. Why does binary search need a sorted list?
18. In the sorted list 4, 9, 13, 18, 25, 31, 47, how many looks does binary search need to find 31? Show the low, high and mid values.
19. Show the list 6, 3, 9, 1 after each pass of bubble sort.
20. A list has 1,000 items. About how many comparisons does linear search need in the worst case? About how many does binary search need on a sorted list?
21. What does $O(n^2)$ tell you about an algorithm?
22. Write pseudocode to read marks of n students and print how many scored 40 or more.
23. Answer the Try this puzzle from the last chapter: 2, 6, 12, 20, 30, ?
24. Ten friends at a party each shake hands once with every other friend. How many handshakes take place?

ANSWER KEY

Answers

1. Input (zero or more values), output (at least one result), definiteness (every step has one clear meaning), finiteness (it stops) and effectiveness (every step can be done).
2. "A little" means different amounts to different people. A definite step is "add half a teaspoon of salt".
3. (a) Parallelogram, (b) diamond, (c) oval, also called the terminal.
4. $29 \text{ MOD } 4 = 1$ and $29 \text{ DIV } 4 = 7$, because $4 \times 7 = 28$ and 1 is left over.
5. Sequence, selection and iteration.
6. `INPUT s, SET area = s * s, SET perimeter = 4 * s, PRINT area, perimeter.`
7. `INPUT n, then IF n MOD 3 = 0 AND n MOD 5 = 0 THEN PRINT "Divisible by both" ELSE PRINT "Not divisible by both" END IF.` For example, 30 is divisible by both and 20 is not.
8. `INPUT age, then IF age >= 18 THEN PRINT "Can vote" ELSE PRINT "Cannot vote" END IF.`
9. `INPUT a, b, then IF a < b THEN PRINT a ELSE PRINT b END IF.` If they are equal, it prints b, which is also the smaller value.
10. `FOR i = 1 TO 10, PRINT 2 * i, END FOR.` Another way is a WHILE loop that starts at 2 and adds 2 each time until it passes 20.

11. `INPUT n, SET count = 0, WHILE n > 0, SET count = count + 1, SET n = n DIV 10, END WHILE, PRINT count.` For 4096 it prints 4. A finished version should treat $n = 0$ as having one digit.
12. Start: rev 0, n 507. Round 1: rev 7, n 50. Round 2: rev 70, n 5. Round 3: rev 705, n 0. Output 705.
13. fact starts at 1. After $i = 1$: 1. After $i = 2$: 2. After $i = 3$: 6. After $i = 4$: 24. Output 24.
14. (84, 36) gives $r = 12$. (36, 12) gives $r = 0$. (12, 0) ends the loop. The GCD is 12.
15. A loop whose condition never becomes false, so it never stops. Forgetting to change the loop variable, such as leaving out `SET i = i + 1`, causes one.
16. `SET min = A[1], FOR i = 2 TO n, IF A[i] < min THEN SET min = A[i] END IF, END FOR, PRINT min.`
17. Binary search throws away half the list after comparing with the middle item. That is only safe if all the smaller items are on one side and all the larger items are on the other.
18. Two looks. low 1, high 7, mid 4: $A[4] = 18$ is too small, so low = 5. low 5, high 7, mid 6: $A[6] = 31$, found.
19. Pass 1: 3, 6, 1, 9. Pass 2: 3, 1, 6, 9. Pass 3: 1, 3, 6, 9.
20. Linear search: up to 1,000 comparisons. Binary search: at most about 10.
21. The work grows with the square of the input size, so ten times the data means about a hundred times the work.
22. `INPUT n, SET count = 0, FOR i = 1 TO n, INPUT m, IF m >= 40 THEN SET count = count + 1 END IF, END FOR, PRINT count.` Here 40 is only an example pass mark.
23. 42. The gaps are 4, 6, 8, 10, so the next gap is 12. Each term is also $n \times (n + 1)$: 1×2 , 2×3 and so on, up to $6 \times 7 = 42$.
24. 45. Each of the 10 friends shakes 9 hands, which gives 90, but that counts every handshake twice, so divide by 2.

WORD LIST

Glossary

Algorithm

a finite set of clear, ordered steps that solves a problem.

Array

a list of values stored under one name and reached by position, such as `A[3]`.

Big-O notation

a way of describing how the work of an algorithm grows as the input grows.

Binary search

a search on a sorted list that halves the part still to be searched at every step.

Bubble sort

a sort that repeatedly swaps neighbouring items that are in the wrong order.

Complexity

the amount of work or memory an algorithm needs, compared with the size of its input.

Condition

a test that is either true or false, such as $n > 0$.

Connector

a small circle that joins two parts of a flowchart.

Decision

a diamond-shaped flowchart box that asks a yes/no question.

Dry run

following an algorithm by hand with real values to check that it works.

Flowchart

a diagram of an algorithm made of shaped boxes joined by arrows.

Indentation

pushing lines to the right to show that they belong inside an IF or a loop.

Infinite loop

a loop that never stops because its condition never becomes false.

Iteration

repeating a group of steps; another word for a loop.

Linear search

checking items one by one until the wanted item is found or the list ends.

MOD

the remainder after one whole number is divided by another.

Palindrome

a number or word that reads the same forwards and backwards.

Prime number

a whole number greater than 1 whose only factors are 1 and itself.

Pseudocode

algorithm steps written in structured, simple English instead of a real programming language.

Selection

choosing which steps to run by testing a condition.

Sequence

steps that run one after another, each exactly once.

Trace table

a table recording the value of every variable at each step of a dry run.

Variable

a named box in memory that holds a value that can change.



Gurukul Technicals Institute

Computer education institute in Delhi, with authorised member centres, robotics classes and science projects, working models and DIY kits.

COURSES

Computer & Office Skills · Accounting & Business Software · Programming & Web Development · Design & Digital Skills · Hardware & Networking · Robotics & Coding · Other Programmes

PROJECTS AND KITS

Ready-made school projects · DIY and STEM kits · Robotics and coding · Components and spares · Engineering projects

CONTACT

Head office: A-10, Chief Commissioner Colony, opposite Rana Pratap Bagh, Delhi 110007

Phone: 011 2721 1512 · **Mobile and WhatsApp:** +91 97116 99511

Email: engineergurukul@yahoo.co.in

Website: engineergurukul.com · **Instagram:** [@science_projects_club](https://www.instagram.com/science_projects_club)

More free ebooks for computer courses, robotics and science projects: engineergurukul.com/resources/ebooks/

Free to read, print and share. Not for sale.