Mizpah Christian School - https://www.mizpahchristianschool.org

---

Topic : Algorithms – Theory and practice questions

Theory and practice questions

What is an Algorithm?

A **computer algorithm** is a detailed set of instructions that, when executed sequentially, solves a particular problem.

Steps to Develop an Algorithm

1. **Understand the problem**.
2. **Identify the output** required for the problem.
3. **Identify the inputs** necessary to achieve the desired output.
4. **Design a logical approach** to reach the desired output.
5. **Test the algorithm**.
6. **Repeat** steps 1 to 5 until the desired results are achieved.

Example Algorithm: Making a Mobile Call

1. Pick up the Mobile.
2. Open mobile MENU.
3. Select Contacts option.
4. Search for the contact you wish to call.
5. Select the contact.
6. Go to Option.

7. Press the call button.

Symbols for Operations

- **Addition**: `+`
- **Subtraction**: `-`
- **Multiplication**: `*`
- **Division**: `/`
- **Assignment**: `←` (e.g., `A ← X * 3` assigns `X * 3` to `A`)

Forms of Algorithm Representation

1. **Pseudo code**
2. **Flow chart**

Advantages of Pseudo Code

- **Reduced complexity**.
- **Increased flexibility**.
- **Ease of understanding**.

Example Problems and Algorithms

Problem 1: Find the Area of a Circle with Radius r

- **Inputs**: Radius `r`
- **Expected Output**: Area of the circle

Algorithm:

1. Read/input the radius `r`.
2. Calculate the area as `Area ← PI * r * r`.
3. Print the area.

Problem 2: Find the Sum of Two Numbers

- **Inputs**: `num1`, `num2`
- **Expected Output**: Sum of the two numbers

Algorithm:

1. Start.
2. Read/input the first number `num1`.
3. Read/input the second number `num2`.
4. Calculate the sum as `Sum ← num1 + num2`.
5. Print the sum.
6. End.

Problem 3: Convert Fahrenheit to Celsius

- **Input**: Temperature in Fahrenheit `F`
- **Expected Output**: Temperature in Celsius `C`

Algorithm:

1. Start.
2. Read the temperature in Fahrenheit `F`.
3. Calculate `C ← 5/9 * (F - 32)`.
4. Print temperature in Celsius: `C`.
5. End.

Types of Control Structures in Algorithms

Algorithms and flowcharts use three types of control structures:

1. **Sequence**
2. **Branching (Selection)** - Uses conditions for binary decisions, represented with `if-then` in pseudo-code.
3. **Loop (Repetition)** - Repeats statements based on conditions, often represented by `while` and `for` constructs.

Example Problems and Solutions with Control Structures

Problem 4: Find the Greater Number Between Two Numbers

- **Inputs**: `A`, `B`
- **Expected Output**: Greater number

Algorithm:

1. Start.
2. Read/input `A` and `B`.
3. If `A > B`, then `C = A`.
4. If `B > A`, then `C = B`.
5. Print `C`.
6. End.

Problem 5: Evaluate a Function Based on Input Condition

- **Input**: Number `x`
- **Expected Output**: `f(x) = -x` if `x < 0`, otherwise `f(x) = x`

Algorithm:

1. Start.
2. Read/input `x`.
3. If `x < 0`, then `F = -x`.
4. If `x >= 0`, then `F = x`.
5. Print `F`.
6. End.

Problem 6: Find the Largest Value Among Three Numbers

- **Inputs**: `A`, `B`, `C`
- **Expected Output**: Largest number

Algorithm:

1. Start.
2. Read/input `A`, `B`, and `C`.
3. If `(A >= B) and (A >= C)`, then `Max = A`.
4. If `(B >= A) and (B >= C)`, then `Max = B`.
5. If `(C >= A) and (C >= B)`, then `Max = C`.
6. Print `Max`.
7. End.

Problem 7: Calculate Even Numbers Between 0 and 99

Algorithm:

1. Start.
2. Set `I ← 0`.
3. Print `I`.

4. Update `I ← I + 2`.
5. If `(I <= 98)`, go to step 3.
6. End.

Problem 8: Calculate Odd Numbers Less Than or Equal to n

- **Input**: Natural number `n`
- **Expected Output**: Odd numbers up to `n`

Algorithm:

1. Start.
2. Read `n`.
3. Set `I ← 1`.
4. Print `I`.
5. Update `I ← I + 2`.
6. If `(I <= n)`, go to step 4.
7. End.

Problem 9: Generate Even Numbers from 1000 to 2000 and Calculate Their Sum

Algorithm:

1. Start.
2. Set `I ← 1000` and `S ← 0`.
3. Print `I`.
4. Update `S ← S + I`.
5. Update `I ← I + 2`.
6. If `(I <= 2000)`, go to step 3; otherwise, go to step 7.
7. Print `S`.
8. End.

Problem 10: Calculate a Series Sum

- **Input**: Natural number `n`
- **Expected Output**: `S = ½ + ¼ + ... + 1/n`

Algorithm:

1. Start.
2. Read `n`.

3. Set `I ← 2` and `S ← 0`.
4. Update `S = S + 1/I`.
5. Update `I ← I + 2`.
6. If `(I <= n)`, go to step 4; otherwise, print `S`.
7. End.

Problem 11: Product of First `n` Natural Numbers

Algorithm:

1. Start.
2. Set `Product` to 1.
3. For `i = 1` to `n`, do:
    - `Product = Product * i`.
4. Print `Product`.
5. Stop.

Problem 12: Find the Greatest of Three Numbers

Algorithm:

1. Start.
2. Read `A`, `B`, `C`.
3. If `A > B` is true, check if `A > C`; if true, `A` is greatest, else `C` is greatest.
4. If `A > B` is false, check if `B > C`; if true, `B` is greatest, else `C` is greatest.
5. Stop.

Problem 13: Determine if a Number is Odd or Even

Algorithm:

1. Start.
2. Input `n`.
3. Compute `num = n % 2`.
4. If `num = 0`, print “Even”; else, print “Odd”.
5. Stop.

Problem 14: Calculate Area and Circumference of a Circle

Algorithm:

1. Start.
2. Input radius `r`.

3. Calculate `Area = π * r * r`.
4. Calculate `Circumference = 2 * π * r`.
5. Print `Area`, `Circumference`.
6. Stop.

Flowchart

A flowchart can therefore be used to:

– Define and analyze processes
– Build a step-by-step picture of the process for analysis, discussion, or communication
– Define, standardize or find areas for improvement in a process

Summary / Important definitions
Algorithm is the sequence of steps to be performed in order to solve a problem by the computer.
Three reasons for using algorithms are efficiency, abstraction and re-usability.
Algorithms can be expressed in many different notations, including natural languages, pseudo code, flowcharts and programming languages.

Analysis of algorithms is the theoretical study of computer program performance and resource usage, and is often practiced abstractly without the use of specific programming language or implementation.
The practical goal of algorithm analysis is to predict the performance of different algorithms in order to guide program design decisions.
Most algorithms do not perform the same in all cases; normally an algorithm's performance varies with the data passed to it.
Typically, three cases are recognized: the **best case, average case and worst case.**

Worst case analysis of algorithms is considered to be crucial to applications such as games, finance and robotics.
O-notation, also known as Big O-notation, is the most common notation used to express an algorithm's performance in a formal manner.

Flowchart is a graphical or symbolic representation of an algorithm. It is the diagrammatic representation of the step-by-
step solution to a given problem.
Flowcharts are used in analyzing, designing, documenting or managing a process or program in various fields.
Benefits of using flowcharts include ease of communication, effective and efficient analysis and coding, proper documentation and maintenance.
Limitations of using flowcharts include **complex logic and multiple modifications.**

The types of flowcharts are High-Level Flowchart and Detailed flowchart.

Program Design consists of the steps a programmer should do before they start coding the program in a specific language.

END