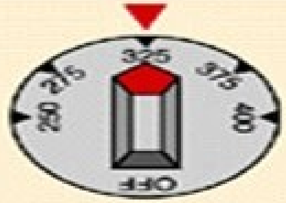
Algorithm

Algorithm

- **Step by step procedure** to solve a problem
- In Computer Science following notations are used to represent algorithm
- Flowchart: This is a graphical representation of computation
- **Pseudo code:** They usually look like English statements but have additional qualities

Heat oven to 325°F



Gather the ingredients

Mix ingredients thoroughly
in a bowl

Pour the mixture into a
baking pan

Bake in the oven
50 minutes

Repeat

Bake 5 minutes more

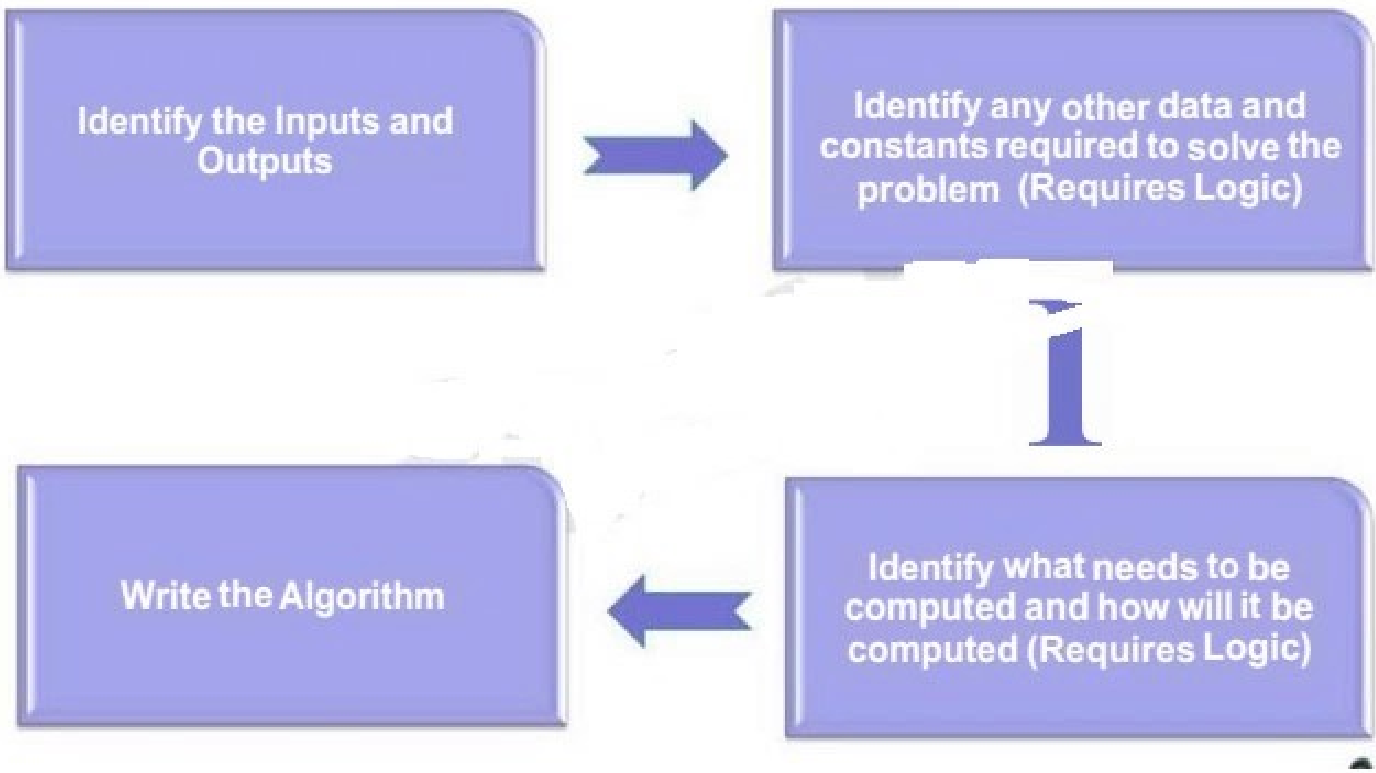
Until cake top springs back when touched in the center

Cool on a rack before cutting

Algorithm

- Algorithms are not specific to any programming language
- An algorithm can be implemented in any programming language
- Use of Algorithms
  – Facilitates easy development of programs
  – Iterative refinement
  – Easy to convert it to a program
  – Review is easier

Steps to Develop an Algorithm



Algorithm for Real life Problem

PROBLEM: Heat up a can of soup

ALGORITHM:

1 open can using can opener

2 pour contents of can into saucepan

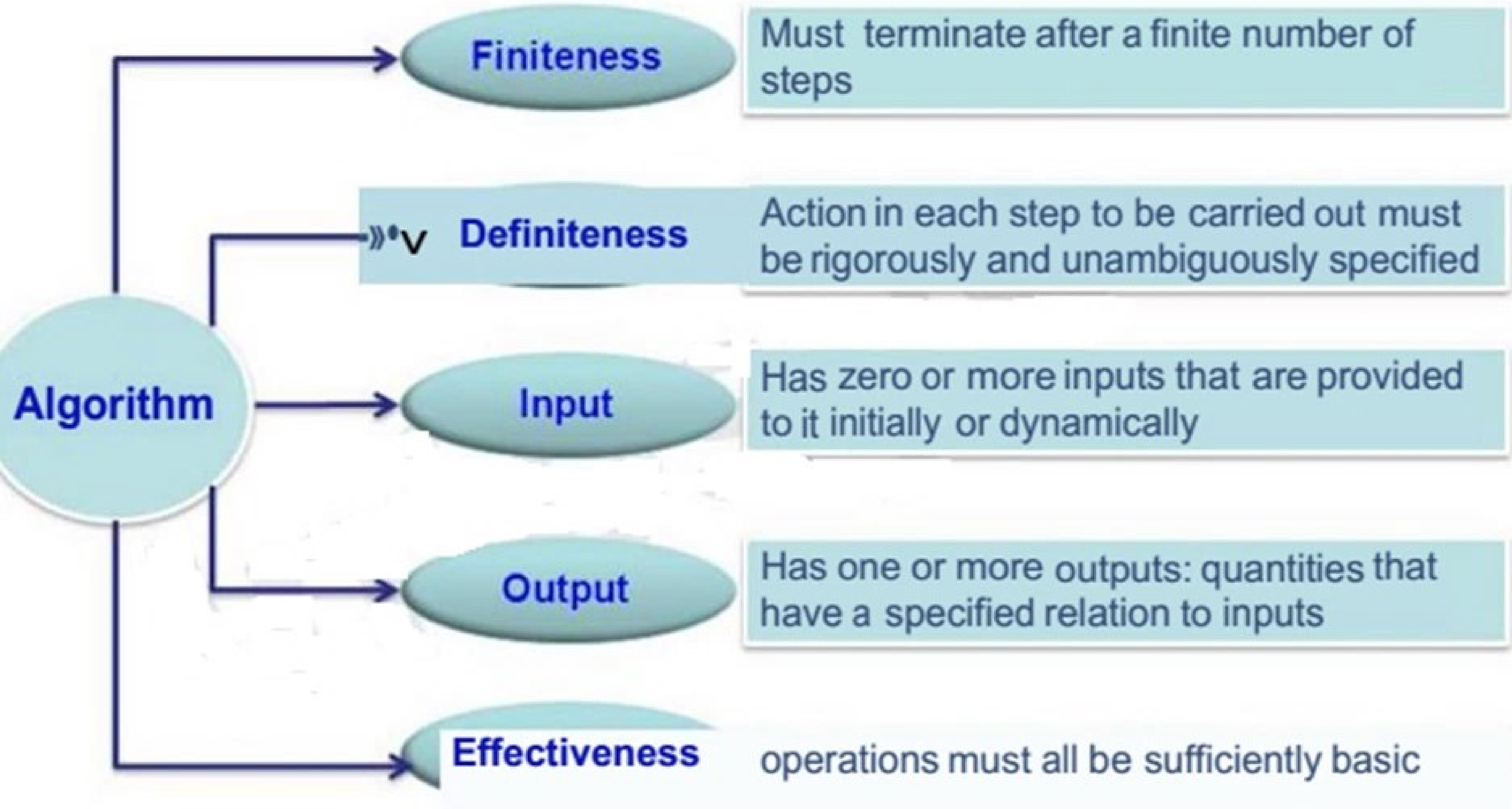
3 place saucepan on ring of cooker

4 turn on correct cooker ring

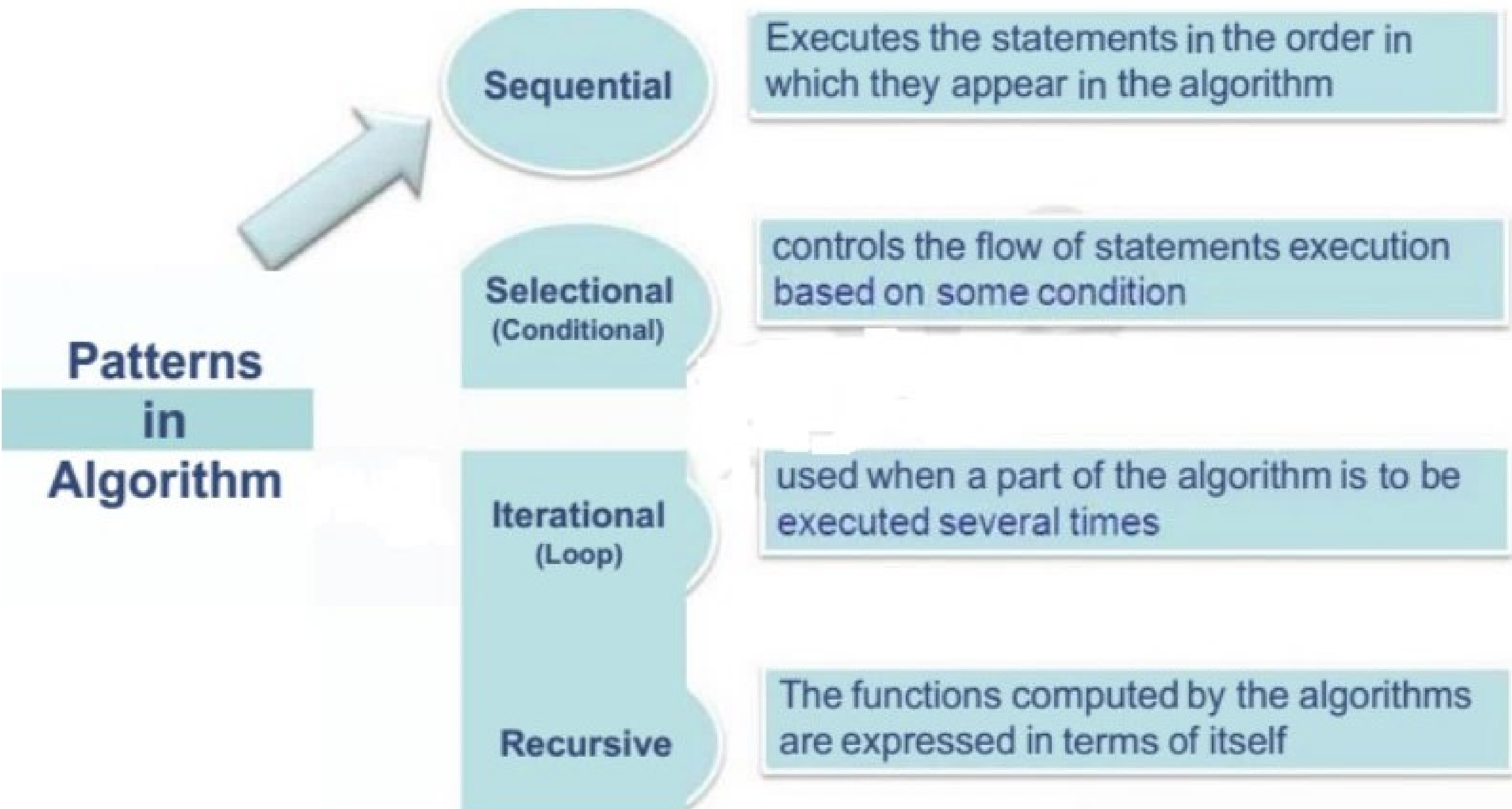
5 stir soup until warm

may seem a bit of a silly example but it does show us that the order of the events is important since we cannot pour the contents of the can into the saucepan before we open the can.

Properties of an Algorithm



Different patterns in Algorithm



Sequential Algorithms

Algorithm for adding two numbers

Step 1: Read two numbers A and B

Step 2: Let C = A + B

Step 3: Display C

Area of a Circle

Step 1 : Read the RADIUS of a circle

Step 2 : Find the square of RADIUS and store it in SQUARE

Step 3 : Multiply SQUARE with 3.14 and store the result in AREA

Step 4: Print AREA

Average Marks

- Find the average marks scored by a student in

  3 subjects:

  Step 1 : Read Marks1, Marks2, Marks3

  Step 2 : Sum = Marks1 + Marks2 + Marks3

  Step 3 : Average = Sum / 3

  Step 4 : Display Average

Selectional Algorithms

Algorithm for Conditional Problems

PROBLEM: To decide if a fire alarm should be sounded

ALGORITHM:

1 IF fire is detected **condition**

2 THEN sound fire alarm **action**

Another example is:-

PROBLEM: To decide whether or not to go to school

ALGORITHM:

1 IF it is a weekday AND it is not a holiday

2 THEN go to school

3 ELSE stay at home

Pass/ Fail and Average

- Write an algorithm to find the average marks of a student. Also check whether the student has passed or failed. For a student to be declared pass, average marks should not be less than 65.

  Step 1 : Read Marks1, Marks2, Marks3
  Step 2 : Total = Marks1 + Marks2 + Marks3
  Step 3 : Average = Total / 3
  Step 4 : Set Output = “Student Passed”
  Step 5 : if Average < 65 then Set Output = “Student Failed"
  Step 6 : Display Output

Leap Year or Not

Step 1 : Read YEAR

Step 2 : IF **({YEAR%4=0 AND YEAR%100!=0)OR (YEAR%400=0))**

Display "Year is a leap year"

ELSE

Display “Year is not a leap year"

ENDIF

Algorithm for Iterative Problems

This type of loop keeps on carrying out a command or commands UNTIL a given condition is satisfied, the condition is given with the UNTIL command, for example:-

PROBLEM: To wash a car

ALGORITHM:

1 REPEAT

2 wash with warm soapy water

3 UNTIL the whole car is clean

Iterational Algorithms – Repetitive Structures

- Find the average marks scored by ‘N’ number of students

Step 1 : Read Number Of Students

Step 2 : Let Counter = 1

Step 3 : Read Marks1, Marks2, Marks3

Step 4 : Total = Marks1 + Marks2 + Marks3

Step 5 : Average = Total / 3

Step 6 : Set Output = “Student Passed”

Step 7 : If (Average < 65) then Set Output = “Student Failed"

Step 8 : Display Output

Step 9 : Set Counter = Counter + 1

Step 10 : If (Counter <= NumberOfStudents) then goto step 3

Bigger Problems

- If you are asked to find a solution to a major problem, it can sometimes be very difficult to deal with the complete problem all at the same time.
- For example building a car is a major problem and no-one knows how to make every single part of a car.
- A number of different people are involved in building a car, each responsible for their own bit of the car's manufacture.
- The problem of making the car is thus broken down into smaller manageable tasks.
- Each task can then be further broken down until we are left with a number of step-by-step sets of instructions in a limited number of steps.
- The instructions for each step are exact and precise.

Top Down Design

- Top Down Design uses the same method to break a programming problem down into manageable steps.
- First of all we break the problem down into smaller steps and then produce a Top Down Design for each step.
- In this way sub-problems are produced which can be refined into manageable steps.

Top Down Design for Real Life Problem

PROBLEM: To repair a puncture on a bike wheel.

ALGORITHM:

1. remove the tyre
2. repair the puncture
3. replace the tyre

Step 1: Refinement:

1. Remove the tyre

1.1 turn bike upside down

1.2 lever off one side of the tyre

1.3 remove the tube from inside the tyre

Step 2: Refinement:

2. Repair the puncture Refinement:

2.1 find the position of the hole in the tube

2.2 clean the area around the hole

2.3 apply glue and patch

Step 3: Refinement:

3. Replace the tyre Refinement:

3.1 push tube back inside tyre

3.2 replace tyre back onto wheel

3.3 blow up tyre

3.4 turn bike correct way up

Still more Refinement:

Sometimes refinements may be required to some of the sub-problems, for example if we cannot find the hole in the tube, the following refinement can be made to 2.1:-

Still more Refinement:

Step 2.1: Refinement

2.1 Find the position of the hole in the tube

2.1.1 WHILE hole cannot be found

2.1.2 Dip tube in water

2.1.3 END WHILE

Algorithm
(Examples)

Write an algorithm to add two numbers entered by user.

[Sequential Structure]

Step 1: Start

Step 2: Declare variables num1, num2 and sum.

Step 3: Read values num1 and num2.

Step 4: Add num1 and num2 and assign the result to sum.

$$sum \leftarrow num1 + num2$$

Step 5: Display sum

Step 6: Stop

Write an algorithm to find the largest among three different numbers entered by user.

[Conditional Structure]

Step 1: Start

Step 2: Declare variables a,b and c.

Step 3: Read variables a,b and c.

Step 4: If a>b

```
        If a>c
          Display a is the largest number.
        Else
          Display c is the largest number.
        Else
          If b>c
            Display b is the largest number.
              Else
            Display c is the greatest number.
```

Step 5: Stop

Write an algorithm to find all roots of a quadratic equation ax2+bx+c=0.

[Conditional Structure]

Step 1: Start

Step 2: Declare variables a, b, c, D, x1, x2, rp and ip;

Step 3: Calculate discriminant

$D \leftarrow b2-4ac$

Step 4: If $D \geq 0$

$r1 \leftarrow (-b+\sqrt{D})/2a$

$r2 \leftarrow (-b-\sqrt{D})/2a$

Display r1 and r2 as roots.

Else

Calculate real part and imaginary part

$rp \leftarrow b/2a$

$ip \leftarrow \sqrt{(-D)}/2a$

Display rp+j(ip) and rp-j(ip) as roots

Step 5: Stop

Write an algorithm to find the factorial of a number entered by user.

[Iterational Structure]

Step 1: Start

Step 2: Declare variables n,factorial and i.

Step 3: Initialize variables

factorial←1

i←1

Step 4: Read value of n

Step 5: Repeat the steps until i=n

5.1: factorial←factorial*i

5.2: i←i+1

Step 6: Display factorial

Step 7: Stop

Write an algorithm to check whether a number entered by user is prime or not.

[Conditional and Iterational Structure]

Step 1: Start

Step 2: Declare variables n,i,flag.

Step 3: Initialize variables

flag←1

i←2

Step 4: Read n from user.

Step 5: Repeat the steps until i<(n/2)

5.1: If remainder of n÷i equals 0

flag←0

Go to **step 6**

5.2: i←i+1

Step 6: If flag=0

Display n is not prime

else

Display n is prime

Step 7: Stop

Write an algorithm to find the Fibonacci series till term≤1000.

[Iterational Structure]

Step 1: Start

Step 2: Declare variables first_term, second_term , temp.

Step 3: Initialize variables first_term←0 second_term←1

Step 4: Display first_term and second_term

Step 5: Repeat the steps until second_term≤1000

  5.1: temp←second_term

  5.2: second_term←second_term+first term

  5.3: first_term←temp

  5.4: Display second_term

Step 6: Stop