# UNIT I ALGORITHMIC PROBLEM SOLVING

Algorithms, building blocks of algorithms (statements, state, control flow, functions), notation (pseudo code, flow chart, programming language), algorithmic problem solving, simple strategies for developing algorithms (iteration, recursion). Illustrative problems: find minimum in a list, insert a card in a list of sorted cards, guess an integer number in a range, Towers of Hanoi.

Part A (2 marks)

**1. List out any five programming languages**

C, C++, Java, python and Javascript

**2. Define an algorithm**

Algorithm is the set of explicit and clear steps of instructions to solve a problem. An algorithm must always terminate after a finite number of steps, after obtaining a required output for any legitimate input.

**3. Differentiate the algorithm and the program**

|  |  |
| --- | --- |
| **Algorithm** | **Program** |
| Algorithm is the set of explicit and clear steps to solve a problem, yet not implemented for the computer | When an algorithm is implemented in a specific programming language, it is known as program |
| Algorithm is developed in the design phase (before programming) | Program is implemented in the development phase |
| Algorithm is independent of any programming language | Program is developed in a specific programming language |
| Algorithm is represented as pseudo code or flowchart | Program is coded in a specific programming language (such as python) |

**4. List out the properties of the (good) algorithm**

1. **Finiteness:** terminates after a finite number of steps

2. **Definiteness:** rigorously and unambiguously specified

3. **Input:** valid inputs are clearly specified

4. **Output:** can be proved to produce the correct output given a valid input

5. **Effectiveness:** steps are sufficiently simple and basic

5. What are the ways to represent the algorithm?

|  |  |
| --- | --- |
| **Flowchart** | **Pseudocode** |
| Flowchart is the pictorial representation of the algorithm. | Pseudocode is written using generic syntax (keywords) and English.  Pseudocode ‘looks’ like the code. |
| Algorithm to find the maximum of three numbers | |
| **Flowchart** | **Pseudocode**  procedure **max**(a, b, c)  **if** a > b **and** a > c **then**  return a  **else if** b > c **then**  return b  **else**  return c  end procedure |

Apart from the above, algorithm is also represented in,

* Step by step instructions in plain english
* Decision tables

**6. Differentiate code and pseudo code**

Pseudo code is written using generic syntax (keywords) and English. Pseudo code ‘looks’ alike the code, but not the code itself.

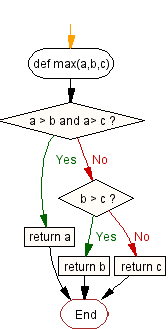
|  |  |
| --- | --- |
| **procedure** **max**(a, b, c)  **if** a > b **and** a> c **then**  **return** a  **else if** b > c **then**  **return** b  **else**  **return** c  **end max** | **def max(a, b, c):**  if a > b and a> c:  return a  elif b > c:  return b  else  return c |

**7. Write an algorithm to print the even numbers upto 100**

Step 1: Start  
Step 2: number = 2  
Step 3: While number is less than or equal to 100,  
 Step 3.1: print number  
 Step 3.2: Increment number by 2  
Step 4: End

**8. What is a flowchart. Give an example**

Flowchart is the pictorial representation of the algorithm.  
Example: Algorithm to find the maximum of three numbers,



**9. List out any three rules for drawing a flowchart**

* Input/ Output statements are represented in parallelogram symbol

print result

* Processes/ tasks performed are represented in rectangle symbol

x = 20  
y = 30  
result = x + y

* Decision/ selection statement is represented in diamond symbol

False

a > b ?

True

* The decision symbol must have two paths (for True and False conditions)
* All the boxes are connected with arrows
* Flowchart flows from top to bottom in general. (except for loops which has the upward flow).
* Connectors can be used to connect the flowchart from one page to another.
* Each procedure shall be represented in a separate flowchart.
* The start and end point of the algorithm is represented with terminal symbol

start

**10. What are the three building structures used in any algorithm**

* Sequence structure: Sequence of steps executed one after other.
* Selection structure: A path is selected based on the given condition.
* Loop structure: Set of steps are repeated till a condition is met or specific number of times.

**11. What are the benefits of using flowchart**

* A picture is worth 1000 words.
* Clearly represents the algorithm in the diagram
* Used for documentation and communication
* Easier to analyze
* Used for software maintenance

**12. Mention any two techniques used to design an algorithm**

* Divide and conquer
* Brute-force
* Successive approximation
* Backtracking
* Greedy algorithms
* Branch and bound

**13. How do you measure the performance of an algorithm**

* Time complexity: measures the CPU time consumed by the algorithm
* Space complexity: indicates the memory required for the algorithm

**14. What is Big-O notation**

Big-O notation is the order of magnitude which represents the number of steps (n) required to complete the algorithm, for bigger n.

For example, consider the program

a=5

b=6

c=10

**for** i **in** range(n):

**for** j **in** range(n):

x = i \* i

y = j \* j

z = i \* j

**for** k **in** range(n):

w = a\*k + 45

v = b\*b

d = 33

It requires 3+ 3n2 + 2n + 1= 3n2 + 2n + 4 steps to complete. But, when we consider for bigger n, we can ignore other terms and simply represent the time complexity of the algorithm in Big-O notation as **O(n2)**.

**15. What is worst case analysis**

Best, worst and average case complexities of an algorithm represent what the CPU utilization is atleast, atmost and on average respectively.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm** | **Time complexity: Best** | **Time complexity: Average** | **Time complexity: Worst** | **Space complexity: Worst** |
| Merge sort | O(*n* log(*n*)) | O(*n* log(*n*)) | O(*n* log(*n*)) | O(n) |
| Bubble sort | O(*n*) | O(*n*2) | O(*n*2) | O(1) |
| Insertion sort | O(*n*) | O(*n*2) | O(*n*2) | O(1) |
| Selection sort | O(*n*2) | O(*n*2) | O(*n*2) | O(1) |

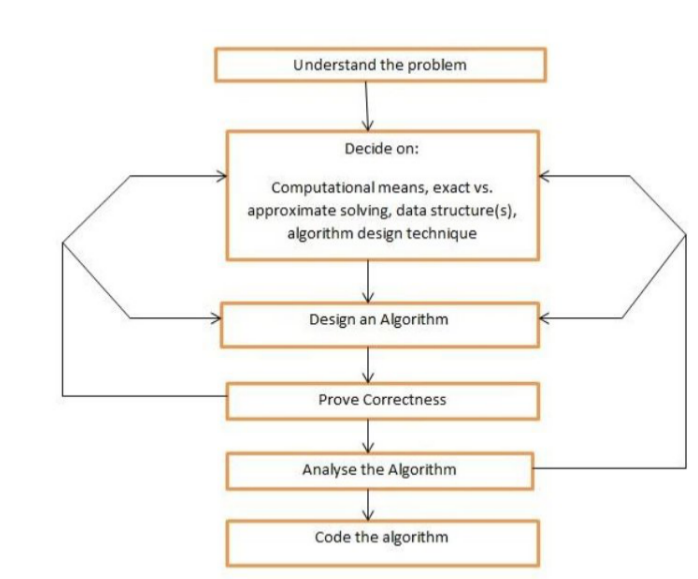
**16. What is the first step in solving the problem using algorithm?**

**Understanding the problem**

Clearly define the problem

* + Illustrated with sample inputs and expected outputs
  + No ambiguity
  + No guess – Assumptions to be explicitly defined and clarified
  + Ask all possible questions
  + Focus on ‘what to do’ (problem) rather than ‘how to do’ (algorithm)

**17. What are the steps in algorithmic problem solving?**



**18. What is space complexity and time complexity**

* Time complexity: measures the CPU time consumed by the algorithm
* Space complexity: indicates the memory required for the algorithm

**19. Differentiate high level and low level language**

|  |  |
| --- | --- |
| **Low level language** | **High level language** |
| The set of instructions in binary format which the processors (computer) can understand and execute | The program written in English-like programming language such as python |
| Also known as machine level language | The compiler is used to translate the high level language into assembly language and then into machine level language (in combination with assembler). |

**20. Draw the symbols to represent the terminator, process, input/ output and decision in a flowchart.**

* Input/ Output statements are represented in parallelogram symbol

print result

* Processes/ tasks performed are represented in rectangle symbol

x = 20  
y = 30  
result = x + y

* Decision/ selection statement is represented in diamond symbol

False

a > b ?

True

* The start and end point of the algorithm is represented with terminal symbol

start

Essay Questions (Part B & Part C)

**1. Describe the general rules for drawing flowchart with examples.**

Flowchart is the pictorial representation of the algorithm.

* Input/ Output statements are represented in parallelogram symbol

print result

* Processes/ tasks performed are represented in rectangle symbol

x = 20  
y = 30  
result = x + y

* Decision/ selection statement is represented in diamond symbol

False

a > b ?

True

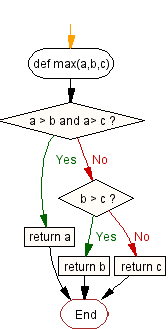
* The decision symbol must have two paths (for True and False conditions)
* All the boxes are connected with arrows
* Flowchart flows from top to bottom in general. (except for loops which has the upward flow).
* Connectors can be used to connect the flowchart from one page to another.
* Each procedure shall be represented in a separate flowchart.
* The start and end point of the algorithm is represented with terminal symbol

start

Benefits of using Flowchart:

* A picture is worth 1000 words.
* Clearly represents the algorithm in the diagram
* Used for documentation and communication
* Explains the computing process to the operator (operator manual)
* Easier to analyze
* Used for software maintenance

Example: Algorithm to find the maximum of three numbers,



**2. Mention the general procedure for writing pseudo code with examples.**

Pseudocode is written using generic syntax (keywords) and English. Pseudocode ‘looks’ like the code.

General procedure to write a pseudocode:

* Only one statement per line
* Capitalize keywords such as IF, ELSE, WHILE, etc.
* Selection structure can be represented as follows:  
  IF: (test condition)
  + Statement(s) to be executed if test condition is TRUE

ELSE:

* + Statement(s) to be executed if test condition is FALSE
* Represent the while loop with

WHILE condition DO  
 process  
END WHILE

Example:

number = 2  
WHILE number < = 100 DO  
 print number  
 number = number + 2

END WHILE

* Write what do you mean, not how to program it.
* Example:

IF number is even THEN  
 print number

instead of,

IF number % 2 == 0 THEN  
 print number

* Do not include data declarations in your pseudocode.

Examples:

**Algorithm: Finding the Maximum of Three Numbers**.

Input: Three numbers *a*, *b*, and *c*

Output: The largest of *a*, *b*, and *c*

**procedure** max(a, b, c):

**if** a > b **and** a> c **then** **return** a  
 **else if** b > c **then**  
 **return** b  
 **else**  
 **return** c  
**end** max

**Algorithm: Testing Whether a Positive Integer is Prime.**

Input: m, a positive integer

Output: true, if m is prime; false, if m is not prime

**procedure** isPrime(m)

**for** i = 2 **to** m − 1 **do**

**if** m is divisible by i **then**

**return** false

**return** true

**end** isPrime

**Algorithm: Finding a Prime Larger Than a Given Integer.**

Input: n, a positive integer

Output: m, the smallest prime greater than n

**procedure** largePrime(n)

m = n + 1

**while** not isPrime(m) **do**  
 m = m + 1

**end while**

**return** m

**end** largePrime

**Algorithm: Find GCD using Euclidean Algorithm.**

Input: a and b (nonnegative integers, not both zero)  
Output: Greatest common divisor of a and b

**procedure** gcd(a, b)  
**if** a < b **then**

swap(a, b)

**while** b is not 0 **do**

**begin**

r := a mod b

a := b

b := r

**end while**

**return** a

**end gcd**

**3. Describe the common traits(properties) of a good algorithm**

Algorithm is the set of explicit and clear steps to solve a problem. An algorithm must always terminate after a finite number of steps.

There may many algorithms to solve the same problem. But, the good algorithm uses less CPU time and memory.

A good algorithm is

* Simple to understand
* Reliable and works for all the valid test cases
* Economical in the use of CPU time and memory (less time and space complexity)
* Not dependent on any hardware, OS or programming language
* Modular to be re-used by other algorithms

**Properties of the good algorithm**

1. **Finiteness**: An algorithm must always terminate after a finite number of steps
2. **Definiteness**: Each step of an algorithm must be explicit and clear
3. **Input**: Initial values given to the algorithm in the beginning
4. **Output**: The result produced by the algorithm
5. **Effectiveness**: A good algorithm is simple to understand, reliably works for all the test cases and economical in the use of memory and CPU time (space and time complexity).

**4. Describe the various building blocks of an algorithm with examples**

**Constants:** The numeric and text values (data) used in the program.  
Example:  
100  
20.54  
“python”

**Variables:** A named identification of a memory location   
Example:   
a = 100  
b = 200

The above statements store the values 100 and 200 in the memory locations identified by a and b respectively. The values identified by the variable can vary in different states. For example,   
a = a + 10   
increments the value stored in a by 10.

**Operators**: Arithmetic operators such as addition (+), subtraction (-), multiplication (\*) and division (/) are used to develop the expressions.

**Expressions**: An expression can be built with the combination of variables, constants and operators  
 Example:

(a \* 40) + (b / 20) – 10  
  
**Statements:** Statement can be built from expressions and assignment operators.  
Example:

result = (a \* 40) + (b / 20) – 10

**Control flow:** Control flow block (if..else..) controls the execution of statements based on the condition.  
Example:  
if number is prime then:  
 sum = sum + number

**State:** State refers to the values of a variable at the particular instance. For example, the switch S1 can be in ON (True) state or OFF (False) state.

Example:

Heater = ON  
IF Temperature is above 24 THEN  
 Heater = OFF

The state consideration is important, when the variables are modified within a while/for loop.  
Example:  
N = [23, 45, 20]  
sum = 0  
for index in range(3):  
 N[index] += 10  
 sum += N[index]  
  
The state table represents how the values vary through each iteration in the loop.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Iteration (index) | numbers[0] | numbers[1] | numbers[2] | sum |
| Before loop | 23 | 45 | 20 | 0 |
| 0 | 33 | 45 | 20 | 33 |
| 1 | 33 | 55 | 20 | 88 |
| 2 | 33 | 55 | 30 | 118 |

**Functions:** A complex software system can be easily developed and maintained, when it is decomposed into modules and functions. A function performs a particular task.  
Example:  
c = sum(a , b)  
This statement calls the function ‘sum’, passing two arguments a and b. The function ‘sum’ adds the two input parameters and returns the result to c.

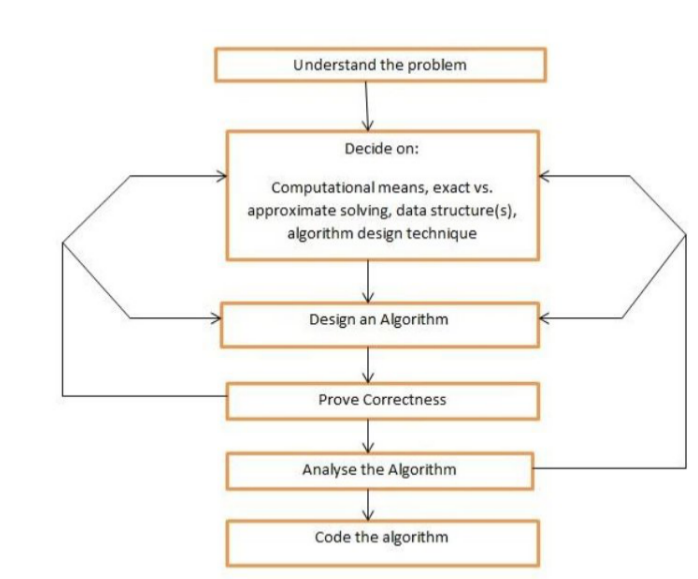
A set of related functions can be grouped into a module.

For example, ‘math’ module consists of functions such as sin, cos, tan, factorial, pow, sqrt

Benefits of using functions:

* It is easier to test the smaller size code, rather testing the entire system(program) all together.
* Functions, once thoroughly tested, can be used in many places.
* Updating function is lot easier than updating all sections of the entire program
* Functions provide modularity

**5. Explain various steps involved in algorithmic problem solving (Program Life cycle)**

****

**Understand the problem**:   
  
Clear problem definition

* + Illustrated with sample inputs and expected outputs
  + No ambiguity
  + No guess – Assumptions to be explicitly defined and clarified
  + Ask all possible questions
  + Focus on ‘what to do’ (problem) rather than ‘how to do’ (algorithm)

Example   
  
Problem: Prepare the list of employees who have been working for five years or more

Understand the problem

* + Full time employees only?
  + Can we include employees on contract?
  + What is the cut-off date for calculation?
  + Can we include employees who worked in break. ( 3 years + 1 year break + 2 years)
  + What attributes to be included in the report? (Name, phone number, etc)
  + Is all this data available?

Problem Analysis:

* + Input
  + Output
  + Tasks
  + Objects and their relations
  + Constraints on the problem

Example:

* + Problem: Find the GCD
  + Input: Two numbers a, b
  + Output: gcd of a and b
  + Task: Calculate the greatest common devisor (GCD)
  + Constraints: Only positive integer non-zero numbers ought to be accepted as inputs

**Decide on Computational means**:

* Memory storage limitations
* Main memory (RAM) limitations
* Platform capabilities
* Operating system capabilities
* Capabilities for handling larger data (parallel algorithms)
  + Parallel processing capabilities
  + Cluster configuration requirements

Case study: The rocket Ariane-5 exploded in 1996. This was due to the shutdown occurred 36.7 seconds after launch, when the guidance system's own computer tried to convert one piece of data — the sideways velocity of the rocket — from a **64-bit format** to a **16-bit format**. The number was too big, and an overflow error resulted.

[ Search for more catastrophes due to software/hardware bugs]

**Look for familiar things:**

* You should never reinvent the wheel
* In computing, you see certain problems again and again in different guises
* A good programmer sees a task, or perhaps part of a task (a subtask), that has been solved before and plugs in the solution.

**Decide on algorithm design techniques:**

* + Brute-force:   
    - Try all possible combinations to arrive at the solution  
    - Depends on computing speed  
    - Example: Find the password by brute-force method
  + Divide and conquer:
    - Divide the problem to smaller problems (by a factor of at least 2)
    - Solved recursively
    - Combine the solutions
    - Example: Merge sort
  + Successive approximation
  + Greedy algorithms
  + Branch and bound
  + Backtracking

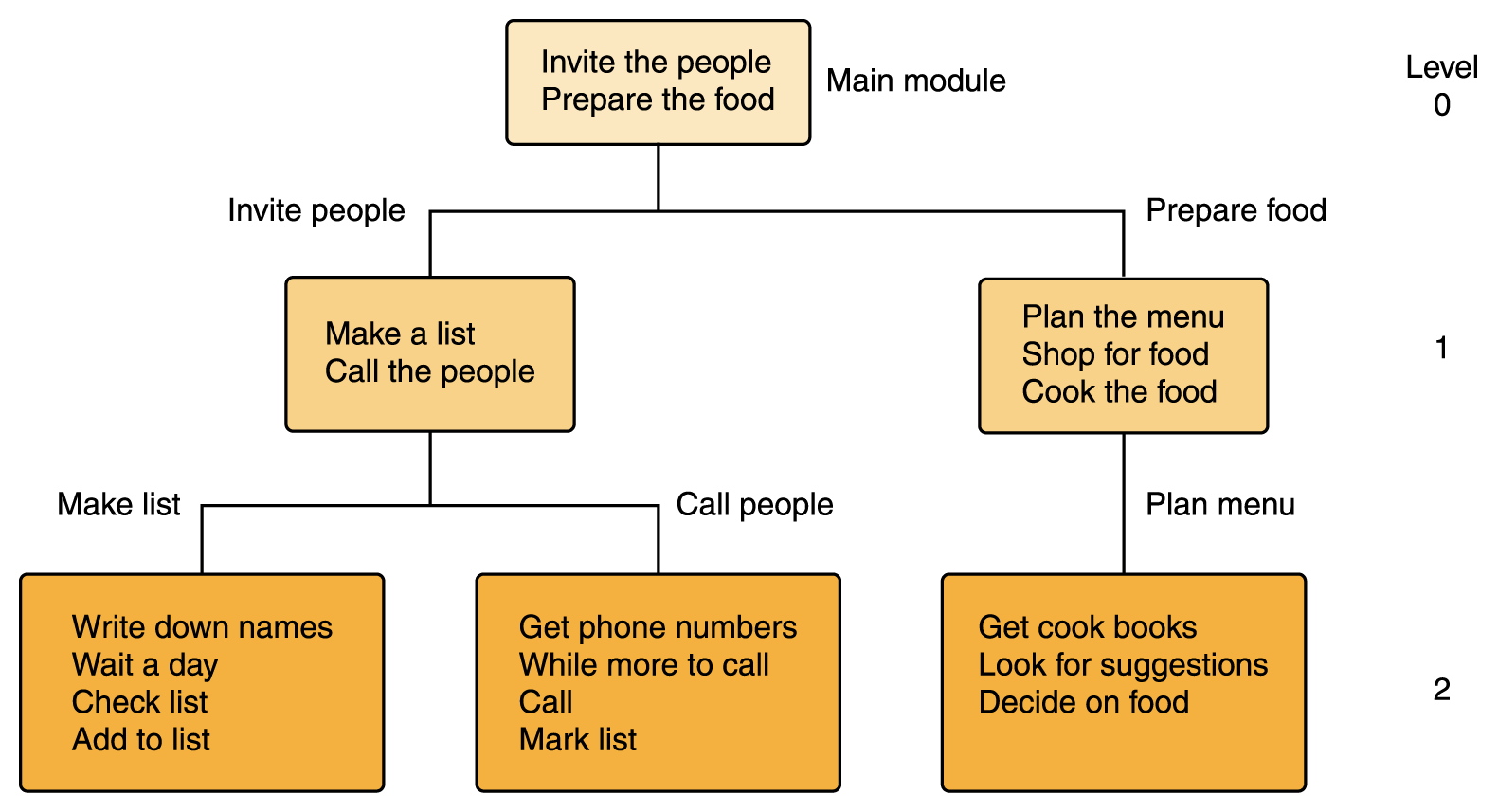
**Decide between approximate vs exact problem solving:**  
Decide on whether you need the approximate solution or the exact solution. For example, you may decide to choose the successive approximation method to find the square root.

**Decide on data structures:**Data structures play the vital role in the design of the algorithm.   
Program = Algorithm + data structures  
Examples: List, Heap, Queue, Stack

**Design an algorithm:**

* Breaking the problem into a set of sub-problems called **modules**
* Creating a hierarchical structure of problems and sub-problems

Example: Algorithm to plan for a party



You may represent the algorithms using

* Step by step instructions in plain english
* Flowcharts
* Pseudo code
* Decision tables

**Proving an algorithm’s correctness:**

Correctness has to be proved for every algorithm.

* The algorithm is complete/correct
  + the post-condition is respected on all possible inputs satisfying the pre-condition
  + Precondition: a predicate I on the input data
  + Postcondition: a predicate O on the output data
  + Correctness: proving I ⇒ O
* The algorithm terminates
* For all possible input, the algorithm exits
* Proving the algorithm terminates (ie, exits) is required at least for recursive algorithm

**Analyze the algorithm:**

* Correctness
* Time complexity: measures the CPU time consumed by the algorithm
* Space complexity: indicates the memory required for the algorithm

Best, worst and average case complexities of an algorithm represent what the CPU utilization is atleast, atmost and on average respectively.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm** | **Time complexity: Best** | **Time complexity: Average** | **Time complexity: Worst** | **Space complexity: Worst** |
| Merge sort | O(*n* log(*n*)) | O(*n* log(*n*)) | O(*n* log(*n*)) | O(n) |
| Bubble sort | O(*n*) | O(*n*2) | O(*n*2) | O(1) |
| Insertion sort | O(*n*) | O(*n*2) | O(*n*2) | O(1) |
| Selection sort | O(*n*2) | O(*n*2) | O(*n*2) | O(1) |

**Properties of the good algorithm:**  
1. Finiteness: terminates after a finite number of steps

2. Definiteness: rigorously and unambiguously specified

3. Input: valid inputs are clearly specified

4. Output: can be proved to produce the correct output given a valid input

5. Effectiveness: steps are sufficiently simple and basic

6. Summarize the difference between algorithm, flowchart and pseudo code with examples

Algorithm is the set of explicit and clear steps of instructions to solve a problem. An algorithm must always terminate after a finite number of steps, after obtaining a required output for any legitimate input.

Example: Algorithm to find the maximum of three numbers

**Step 1:** Get three numbers a, b, c  
 **Step 2: ‘**a’ is maximum if it is greater than b and c  
 **Step 3:** Else, ‘b’ is maximum, if ‘b’ is greater than c  
 **Step 4:** Else, ‘c’ is maximum.

There are two ways to represent the algorithm.

|  |  |
| --- | --- |
| **Flowchart** | **Pseudocode** |
| Flowchart is the pictorial representation of the algorithm. | Pseudocode is written using generic syntax (keywords) and English.  Pseudocode ‘looks’ like the code. |
| Algorithm to find the maximum of three numbers | |
| **Flowchart** | **Pseudocode**  procedure **max**(a, b, c)  **if** a > b **and** a > c **then**  return a  **else if** b > c **then**  return b  **else**  return c  end procedure |

Flowchart is the pictorial representation of the algorithm.

* Input/ Output statements are represented in parallelogram symbol

print result

* Processes/ tasks performed are represented in rectangle symbol

x = 20  
y = 30  
result = x + y

* Decision/ selection statement is represented in diamond symbol

False

a > b ?

True

* The decision symbol must have two paths (for True and False conditions)
* All the boxes are connected with arrows
* Flowchart flows from top to bottom in general. (except for loops which has the upward flow).

Pseudocode is written using generic syntax (keywords) and English. Pseudocode ‘looks’ like the code.

* Only one statement per line
* Capitalize keywords such as IF, ELSE, WHILE, etc.
* Selection structure can be represented as follows:  
  IF: (test condition)
  + Statement(s) to be executed if test condition is TRUE

ELSE:

* + Statement(s) to be executed if test condition is FALSE
* Represent the while loop with

WHILE condition DO  
 process  
END WHILE

7. Explain the following programming languages  
 i) Machine language  
 ii) Assembly language  
 iii) High level language

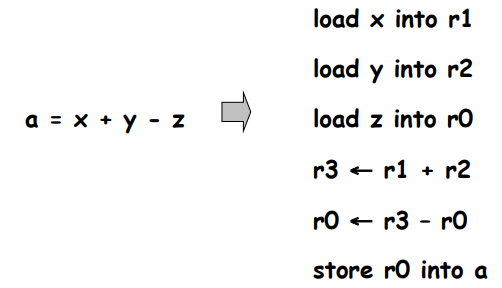
**High level language**

The program written in English-like programming language such as python and Java. It is machine/ processor independent. They are easy to learn, easy to use, maintain and migrate.  
The compiler is used to translate the high level language into assembly language and then into machine level language (in combination with assembler).

Example:

**a = x + y - z**

**Assembly Language**

The high level language is converted into assembly language using compiler. The assembler then converts the code in assembly language into machine language. The assembly language is processor specific and the instructions are represented by Mnemonics. Each processor has a different set of registers, and different assembly language instructions.  
  
Assembly instructions:

Load x, r1

Load y, r2

Load z, r0

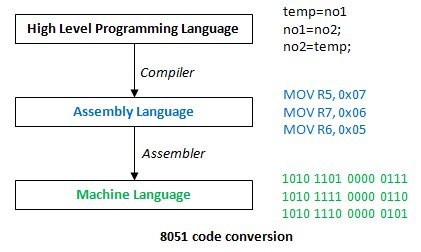
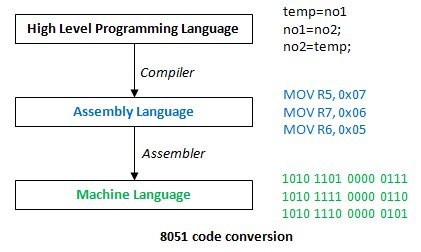
Add r3, r1, r2

Sub r0, r3, r0

Store r0, a

**Machine Language**

It is also known as low level language. Both program and data are represented using only 0’s and 1’s inside a computer. It is the set of instructions in binary format (**opcode**) which the processors (computer) can understand and execute.

8. Explain the use of iteration and recursion in algorithmic problem solving

**Iteration:** It is the ability to repeat the portion of the code over and again. The two available iteration/ loop structures in python are **while** loop and **for** loop.

For example, the iteration can be used to find the minimum in the list of given numbers.

* Find the minimum weight from the **list** of the weights of 10 persons.
* [ 67, 56, 89, 54, 58, 78, 71, 64, 62, 70]

def findMin(L):

minVal = L[0]

for number in L:

if number < minVal:

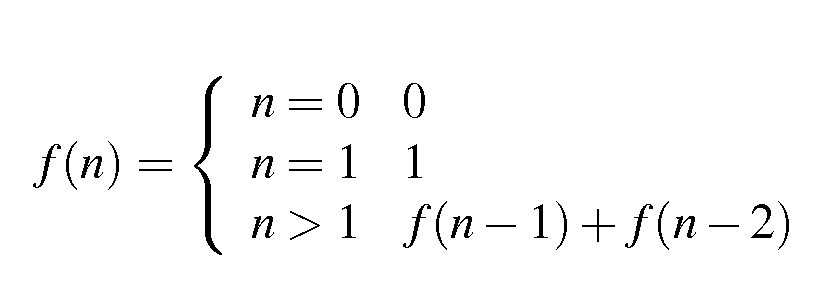
minVal = number

return minVal

**Recursion**: A function is called recursive if the body of the function calls the function itself.

For example, find the nth number in the Fibonacci series (Fn)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **n** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | **7** |
| Fn | 0 | 1 | 1 | 2 | 3 | 5 | 8 | **13** |



def fibo(n):

if n <= 1:

return n

else:

return fibo(n-1) + fibo(n-2)

10. Describe the algorithm to find the minimum in a list

* Example: Find the minimum weight from the **list** of the weights of 10 persons.
* [ 67, 56, 89, 54, 58, 78, 71, 64, 62, 70]
* Let’s solve this problem incrementally.
* The algorithm to find **the minimum of two numbers**:

|  |  |
| --- | --- |
|  |  |

* The algorithm to find **the minimum of three numbers**:

|  |  |
| --- | --- |
|  |  |

The algorithm to find **the minimum in the list of numbers (L)**:

def **findMin**(L):

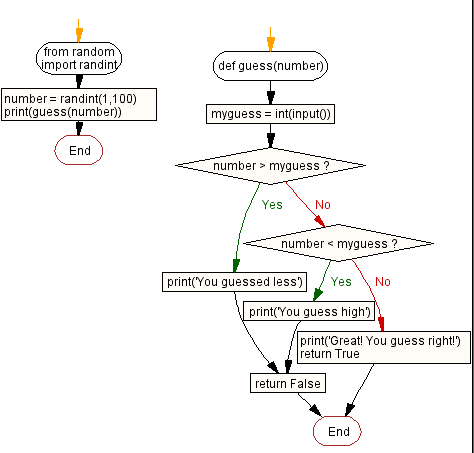
minVal = L[0]

for number in L:

if number < minVal:

minVal = number

return minVal

11. Describe the algorithm and draw the flowchart to guess the number randomly generated by the computer  


from random import randint

def guess(number):

myguess = int(input())

if number > myguess:

print('You guessed less')

elif number < myguess:

print('You guess high')

else:

print('Great! You guess right!')

return True

return False

attempt = 10

number = randint(1,100)

while attempt > 0:

if guess(number):

break

attempt -= 1

print('Good try! Play again!')

12. Illustrate the algorithm to insert a card in a list of sorted cards

The list of sorted cards (deck)



The new card to be inserted



**Iteration 1**

**Iteration 2**

**Iteration 3**

**Iteration 4**

**Program:**

13. How do you solve the tower of honai problem?  
Refer the course book by Dr. R. Ravichandran (page: 44)