Chapter 1

Introduction

1.1 Introduction

Everyone must have solved at least one problem or the other in his or her lifetime preferably without much difficulty or even with some difficulties. Problem solving is an art and there are no universal approaches one can take to solving problems. Basically, one must explore possible options available to a solution one by one until one comes across a right and possibly the best solution to the problem. Generally speaking, there is guessing and hence an element of luck involved in problem solving. However, as one gains experience in solving problems, one develops one's own techniques and strategies for solving a problem. The ability to solve problems is a basic life skill and is essential to our day-to-day lives, may it be at home, at school, and at work. We solve problems every day without really thinking about how we solve them. For example, it is raining and you need to go to school. What do you do? There are lots of possible solutions. Take your umbrella and walk. If you do not want to get wet, you can drive, or take a taxi cab. You might decide to call a friend for a ride, or you might decide to go to school another day. There is no right or universal way to solve this problem and different people will solve it differently.

Problem solving is the process of identifying a problem, developing possible solution paths, and taking the appropriate course of action. Good problem-solving skills empower you not only in your personal life but are critical in your professional life. In the current fast-changing global economy, employers often identify everyday problem solving as crucial to the success of their organizations. For employees, problem solving can be used to develop practical and creative solutions, and to show independence and initiative to employers. The ability to solve problems is a skill, and just like any other skill, the more you practice, the better you get. So how exactly do you practice problem solving? Learning about different problem-solving strategies and when to use them will give you a good start. Problem solving is a process. Most strategies provide ***steps*** that help you identify the problem and choose the best solution. There are two basic types of strategies: algorithmic and heuristic.

Algorithmic strategies are traditional step-by-step guides to solving problems. As an example, suppose a student wants to prepare for school as he woke up in the morning. He must follow certain definite steps before getting to school. One way of going about the steps are:

Algorithm: Preparing for school
Input: Get all required materials e.g., books, bag, sandal, uniform, food
Output: Arrive at school
Step 1: Wake up in the morning
Step 2: Say your daily prayer (if necessary)
Step 3: Bruch your teeth
Step 4: Take your bath
Step 5: Rob cream and other make-ups
Step 6: Put on your dress and shoes
Step 7: Take your breakfast (if necessary)
Step 8: Carry your bag and start going to school
Step 9: Arrive at your school.

This is a simple algorithm that illustrates the necessary steps to take when you want to go to school in the morning as a student. One striking point in this algorithm is that it is sequential and finite. It sequential in the sense that the events that led to your going to school is in sequential or chronological order. Suppose the algorithm is designed this way:

Algorithm: Preparing for school
Input: Get all required materials e.g., books, bag, sandal, uniform, food
Output: Arrive at school
Step 1: Wake up in the morning
Step 2: Put on your dress and shoes
Step 3: Say your daily prayer (if necessary)
Step 4: Bruch your teeth
Step 5: Take your bath
Step 6: Rob cream and other make-ups
Step 7: Take your breakfast (if necessary)
Step 8: Carry your bag and start going to school
Step 9: Arrive at your school.

Notice that in this second case, the student put on his dress and shoes before going to the bathroom to shower with his clothes and shoes on. That is, **step 6** is now carried out in **step 2**. Then one would say that the something must be wrong with the student for having his bath with his school uniform put on. In this case, the order of preparing for school has been altered since the student cannot wear wet clothes to school. This is an example to illustrate our earlier point that in designing algorithm, the steps must be in sequential order for the algorithm to work properly except for situations where loops repetitions are used.

Thus an algorithm is a step-by-step representation of instructions in human readable language. It is a step-by-step procedure for solving a problem. It is a set of detailed instructions used for calculations, data processing, and automated reasoning. It helps in reducing the errors or rectifies them if any in a program. It is a description of a procedure which terminates with a result. Is a step-by-step procedure, especially an established recursive computational procedure for solving a problem in a finite number of steps. An algorithm is a sequence of unambiguous instructions for solving a problem by obtaining a required output for any legitimate input in a finite amount of time. It is usually applied in areas such as sorting, searching, optimization problems, randomness problems, graph problems, data mining and clustering problems, computational geometry problems, image processing problems, numerical problems, etc. Algorithmic strategies are great for solving math problems (in algebra: multiply and divide, then add or subtract) or for helping us remember the correct order of things. Algorithms are best used when there is a single path to the correct solution.

But what do you do when there is no single solution for your problem? ***Heuristic methods*** are general guides used to identify possible solutions. A popular one that is easy to remember is ***IDEAL*****(Bransford and Stem, 1993).**

- Identify the problem
- Define the context of the problem
- Explore possible strategies
- Act on best solution
- Look back and learn

i) **Identify the Problem:** The first step in problem solving is to identify a problem to be solved. There are several real-life problems militating against mankind. Most of these problems have defiled solutions. However, some of the problems also have been solved and some still require better solutions. Therefore, ability to identify these problems is a crucial step to solving them. A problem identified, they say, is half solved. You need to know:

- What input data/information is available?
- What does it represent?
- What format is it in?
- Is anything missing?
- Do I have everything that I need?
- What output information am I trying to produce?
- What do I want the result to look like … text, a picture, a graph …?
- What am I going to have to compute?

ii) **Formulate a Model**

Having identify the problem and think of a solution among possible alternative solutions, the next step is to formulate a model for solving the problem. These problems can be broken down into smaller problems that require some kind of simple mathematical computations in order to process the data. In our example, we are going to compute the average of the incoming grades. So, we need to know the model (or formula) for computing the average of a bunch of numbers. If there is no such "formula", we need to develop one. In order to come up with a model, we need to fully understand the information available to us.

iii) **Develop an Algorithm**

Now that we understand the problem and have formulated a model, it is time to come up with a precise plan of what we want the computer to do.An algorithm is a step-by-step process for solving a problem. Some algorithms are simple while others are complex. Some of the more complex algorithms may be considered "**randomized algorithms**" or **non-deterministic algorithms**" where the instructions are not necessarily in sequence and may not even have a finite number of instructions tends to be more complex. However, the above definition applies to all algorithms that are discuss in this book. To develop an algorithm, we need to represent the instructions in some way that is understandable to a person who is trying to figure out the steps involved. Two commonly used representations for an algorithm is by using (1) **pseudo code**, or (2)**flow charts**. Consider the following example (from Wikipedia) of solving the problem of a broken lamp. To the right is an example of a flow chart, while to the left is an example of pseudocode for solving the same problem:

iv) **Write the Program (Code)**

Once the algorithms and pseudocodes are developed, the next step is to transform the algorithm into a set of instructions that can be understood by the computer. Therefore, a program is a sequence of instructions given to the computer to help it solve a problem. Writing a program is often called "**writing code**" or "**implementing an algorithm**". So the **code** (or **source code**) is actually the program itself. However, the processing code writing is less readable and seems somewhat more mathematical than pseudocodes or algorithms. The simplicity or complexity of the program may vary depending on the programming language that was used. Learning a programming language may seem difficult at first, but it will become easier with practice.The computer requires precise instructions in order to understand what you are asking it to do.Leaving one of them off will cause your program to generate what is known as a **compile error**.Compiling is the process of converting a program into instructions that canbe understood by the computer.The longer your program becomes, the more likely you will have multiple compile errors. You need to debug the program and fix all such compile errors before continuing on to the next step.

v) ***Test the Program:***

Once you have a program written that compiles, you need to make sure that it solves the problem that it was intended to solve and that the solutions are correct. Running a program is the process of telling the computer to evaluate the compiled instructions. When you run your program, if all is well, you should see the correct output. It is possible however, that your program works correctly for some set of data input but not for all. If the output of your program is incorrect, it is possible that you did not convert your algorithm properly into a proper program. It is also possible that you did not produce a proper algorithm back in step 3 that handles all situations that could arise. Maybe you performed some instructions out of sequence. Whatever happened, such problems with your program are known as **bugs**. Bugs are problems/errors with a program that cause it to stop working or produce incorrect or undesirable results. You should fix as many bugs in your program as you can find. To find bugs effectively, you should test your program with many test cases (called a **test suite**). It is also a good idea to have others test your program because they may think up situations or input data that you may never have thought of. The process of finding and fixing errors in your code is called **debugging** and it is often a very time-consuming "chore" when it comes to being a programmer. If you take your time to carefully follow problem solving steps 1 through 3, this should greatly reduce the amount of bugs in your programs and it should make debugging much easier.

vi) **Evaluate the Solution**

Once your program produces a result that seems correct, you need to re-consider the original problem and make sure that the answer is formatted into a proper solution to the problem. It is often the case that you realize that your program solution does not solve the problem the way that you wanted it to. You may realize that more steps are involved.

For example, if the result of your program is a long list of numbers, but your intent was to determine a pattern in the numbers or to identify some feature from the data, then simply producing a list of numbers may not suffice. There may be a need to display the information in a way that helps you visualize or interpret the results with respect to the problem. Perhaps a chart or graph is needed.

It is also possible that when you examine your results, you realize that you need additional data to fully solve the problem. Or, perhaps you need to adjust the results to solve the problem more efficiently (e.g., your game is too slow).

It is important to remember that the computer will only do what you told it to do. It is up to you to interpret the results in a meaningful way and determine whether or not it solves the original problem. It may be necessary to re-do some of the steps again, perhaps going as far back as step 1 again, if data was missing.

So there you have it. Those are the 6 steps that you should follow in order to solve problems using computers. Throughout the course, you should try to use this approach for all of your assignments. It is a good idea to practice problem solving to make sure that you understand the process. Below are some practice exercises that will help you practice the first 3 steps of the problem solving process. Later, you will gain experience with steps 4 through 6.

Steps to Designing an Algorithm

The following steps should be followed when designing an algorithm

1. Read the description.
2. Identify specified inputs and required outputs
3. Figure out how to process the input(s) to get the desired output

Step 3 in Design and Step 2 in Writing is the Hard Part

1. Decide if you need other variables, in addition to the inputs

- If you are calculating a sum or product, you need a variable to hold the sum or product.
- If you are accumulating a sum in a loop, initialize the sum to zero (e.g. sumN)
- If you are accumulating a product in a loop, initialize the product to 1. (e.g. FactN)

2. When coding an iterative statement (a loop): Always make sure there is a way for the loop to stop

   For example, if you write:

   ```
   While the value of I < N Do
       ......
       ......
       ......
   Endwhile
   ```

   Make sure you increase the value of I somewhere in the loop by an appropriate value, so that I eventually becomes equal to or greater than N. This will allow the algorithm to get out of the loop.

3. When designing the algorithm determine whether the input(s) are related to the output(s) by a standard formula.

   For example:

   If the problem deals with circular objects at all, start thinking of formulas related to circles

   If the problem deals with squares or rectangles, think of the formulas related to these shapes

4. If there is no obvious standard formula, determine your own procedure for determining the required outputs

1.2 Developing Problem Solving Processes

Solving problems is the core of computer science. Programmers must first understand how a human solves a problem, then understand how to translate the "algorithm" into something a computer can so, and finally how to "write" the specific syntax (required by a computer) to get the problem solved. However most often, a computer will solve a problem in a completely different way. Problem solving is a process that uses steps to solve problems. The first step in solving a problem is to recognize that there is a problem and identify the right cause of the problem. This is because similar problems may arise from different events, and the real issue may not always be apparent. To really solve the problem, it is important to find out what started it all. This is called identifying the ***root cause***. The best way to identify the root cause of a problem is to ask questions and gather information. If there is a problem, investigating facts is more productive than guessing a solution. Ask yourself questions about the problem. What do you know about the problem? What do you not know? Was there a similar problem in the past? When was the last time it worked correctly? What has changed since then? Can you sketch the process into separate steps? Where in the process is the problem occurring? Be curious, ask questions, gather facts, and make logical deductions rather than assumptions.

1.2.1 Making Decisions

At this stage, a decision is taken on the particular method or technique to use to ensure results are quickly achieved and the most optimized method is used to avoid wastage of resources including money and time.

1.2.2 Problem Solving Strategies in Computer Science

A problem solving strategy is a plan of action used to find a solution. They are actually the steps that one should follow and use to find solution to a problem. It is sometimes referred to as problem-solving cycle. In this cycle, the problem solver will acknowledge and recognize a problem to be solved, define the problem, organize the knowledge of the problem, find out the resources at the user's disposal, monitor owns progress, and evaluate the solution for accuracy. The process continues to find better and better solution. This is only it is called a cycle.

Usually, problems are solved by insight or through step-by-step analysis. Insight is the sudden solution to a long-term problem a sudden recognition of a new data, or a sudden recognition of a new idea, or a sudden understanding of a complex situation. However, it must be noted that solutions found through insightful thought are often more accurate than those found through step-by-step analysis. To solve more problems at a faster rate, insight is necessary for selecting productive moves at different stages of the problem – solving cycle. This problem – solving strategy is used for insight problem.

Problem solving involves the use of generic methods or techniques in an orderly manner to find solutions to problems. Some of the problem – solving techniques developed and used in philosophy, computer science, artificial intelligence, engineering, mathematics, medicine, management sciences and social sciences in general are related to mental problem-solving techniques in psychology and cognitive sciences.

1.2.3 Steps for Problem Solving

Suppose we have a car and while driving it we notice a strange noise. Although, we might not be in the best position to solve the problem, however, we may need to identify where the noise is coming from. Then we need to call a mechanic to come and help fix the car. The mechanic will identify the problem by analyzing the source of the problem, make plan(s) to fix the problem and finally repair the car. As seen from above, there could be many steps involved in problem solving. When problems are straightforward and easy we can easily find the solution. However, for complex problems there is need to have a methodical approach to finding the right solution.

That is, we need to apply the right technique(s) for solving the problem. Problem solving starts with the precise identification of the problem and ends with a complete working solutions in terms of a program or software. The steps required to solving a problem are discussed below.

1. Understanding the Problem
2. Analyzing the Problem
3. Developing an Algorithm
4. Coding
5. Testing and Debugging – Tested using different parameters, it should meet the requirements of the user, respond within the expected time. It should produce correct resulted or outputs. However, if there are errors in the program (i.e., syntax or logical) errors. If the program will not run and as a result no output will be generated. However, if the program runs and output(s) are generated are incorrect, it means the error is a logical error. Thus should be checked and the errors corrected.

In the software industry standardized testing methods are used in software testing. These method include unit or component testing, integration testing, system testing, and acceptance testing for complex applications. This is to ensure that the software meets all the business and technical requirements and works as expected. The errors or defects found during the testing phase are debugged (i.e., removed) or corrected and the program is tested again and again until all the errors are removed from the software. Once the software has been developed, tested, and delivered to the end user feedbacks are expected from them on the functionality of the software. This is because problems may arise in the course of using the software with different parameters. When this occurs, the software is then maintained from time to time by fixing the problems faced while using the software and the software are will continue to evolve.

Some of the techniques used in problem solving are:

- Critical thinking
- Abstraction
- Brainstorming
- Analogy
- Divide and conquer, heuristics, algorithms
- Proof, insight, trial and error
- Reduction root cause analysis
- Research etc

In computer science, however, some of the problem – solving strategies include:

- Divide – and conquer
- Proof
- Reduction
- Research
- Algorithms
- Heuristics

In cognitive psychology, the term problem – solving refers to the mental process that people go through to discover, analyze, and solve problems. In computer science, problem solving involve four main steps:

i) Understanding the problem – solving the right problem is the most important part problem solving.

ii) Design a solution by formulating an algorithm to solve the problem

iii) Implement your solution by writing the code to solve the problem. First you need to understand the problem and design the solution on paper as seen in step (i) and (ii). Then you need to translate your design into actual code. This can best be done by implementing small chunks of problem at a time instead of trying to solve the whole problem at once. Break you code into subroutines, and ensure that each subroutine works before proceeding to the next. Compile your code and save often. If there are syntax errors in your code, determine lines causing errors by systematically commenting out blocks of code until you find the block(s) causing the errors are logical errors, (in which case the program compiles but does not do what it is expected of it), find some examples on which your problem consistently fails. Trace through the programme by line, with one of these examples to figure out exactly which line is not doing what you intended. Also, if the output does not match what you expect, use print statements to trace through what you program is doing and compare it to what it is expected. Better still, you can use a debugger to debug the program. Several debuggers exist. These are discussed in chapter 10.

iv) Check your solution. This step is often overlooked but is very crucial in problem solving. Your have to think critically about your code. Ensure that you check your program to ensure that it is doing what is expected of it. Think of how the program will break or crash, how the software can fail and how one can carioca the users that it will not fact or crash.

1.3 What is Computer Science?

Computer Science is difficult to define. This is probably due to the use of the word "computer" in the name. As you are aware, computer science is not simply the study of computers. Although computers play an important role as a tool in the discipline.

Computer science is the study of problems, problem – solving, and the solutions thereof. Given a problem, a computer scientist's goal is to develop a step – by – step procedure or list of instructions, called algorithm to solve the problem. Algorithms are finite problem. Computer science is the study of algorithms. However, there are problems that computers can solve, yet there are some other problems that computers cannot solve are said to be computable while these problems that computers cannot solve are referred to as uncomputable or unsolvable problems and stating that computer science is the study of solutions to problems as well as the study of alternative definition to computer science is to say that computer science is the study of problems that are and that are not computable, the study of the existence and the non-existence of algorithms. Computer science as it pertains to the problem – solving process itself, is also the study of abstraction. Abstraction allows us to view a problem and solution in such a way as to separate the so – called logical and physical perspectives.

1.3.1 Problem-Solving Strategies

People face problems every day, usually multiple problems. Some of these problems are easy while others are difficult to solve. When you are faced with a problem, whether simple or complex mathematical problem or simple problem, we are faced with the question "How do I or we solve this problem? Before finding a solution to a problem, the problem must clearly be identified. Having identified the problem, one of many problem solving strategies is then applied with the hope of finding a solution.

Therefore, a problem – solving strategy is a plan of action used to find a solution to a problem. Different strategies have different action plans associated with them. A well – known is trial and error. It involves trying different techniques solutions until the techniques that works is determined or until the problem is solved. However, this is not a time efficient method of solving problem strategy. The next technique is the algorithm

Algorithm: This is another type of strategy used in solving problems. An algorithm is a problem – solving technique that provides a step-by-step instructions and to achieve a desired result. An algorithm can be thought of as a recipe with high detailed instructions that produce the same result every time they are performed. Algorithms are used often in our everyday lives, especially in computer science. When you run a search on the Internet search engines like Google Facebook algorithms are used knowingly or unknowingly by the user to decide which entries will appear first in your list of results.

Heuristic Problem Solving Strategy: A heuristic is a general problem – solving technique. It is simply defined as any approach to problem solving or self – discovery that uses a practical method that is not guaranteed to be optimal, perfect, or rational, but is nevertheless sufficient for reaching an in mediate, short – term goal. In computer science and mathematics, heuristic is a technique designed for solving a problem more quickly when classic methods are too slow or for finding an approximate solution when classic methods fail to find any exact solution. This is achieved by trading optimality, completeness accuracy or precision for speed. It often referred to as a "shortcut" technique to problem solving. The objective of a heuristic is to produce a solution in a reasonable time frame that is good enough for solving the problem at hand. However the solution may not be the best of all the solutions to the problem, or it may simply approximate the exact solution. But it is valuable because finding it does not require a prohibitively long time. It should be noted that heuristics may produce results by themselves, or they may be used in conjunction with optimization algorithms to improve efficiency. Problems like travelling salesman problem, search algorithms, NP-hard problems may require the use of heuristic solution.

Definition:*Problem Solving is the act of defining a problem, determining the cause of the problem; identifying, prioritizing, and selecting alternatives for a solution; and implementing a solution.*

In order to effectively solve a problem, the following are the four – step problem – solving process and methodology to follow.

1. **Defining and understand the problem**
   - Differentiate fact from opinion
   - Specify underlying causes
   - Consult each faction involved for information
   - State the problem specifically
   - Identify what standard or expectation is violated
   - Determine in which process the problem lies.
   - Avoid trying to solve the problem without data

2. **Analyze the Problem and generate alternative solutions:**
   - Postpone evaluating alternatives initially
   - Include all involved individuals in the generating of alternatives
   - Specify alternatives consistent with organizational goals
   - Specify short and long – term alternatives
   - Brainstorm on others ideas
   - Seek alternatives that may solve the problem.

3. **Evaluate and Select an alternative**
   - Evaluate alternatives relative to a target standard
   - Evaluate all alternatives without bias
   - Evaluate alternatives relative to established goals
   - Evaluate both proven and possible outcomes
   - State the selected alternative explicitly

4. **Coding and implementation**
   - Plan and implement a pilot test of the chosen alternative
   - Gather feedback from all affected parties
   - Seek acceptance or consensus by all those affected
   - Establish ongoing measures and monitoring.
   - Evaluate long-term results based on final solution

1. Defining the Problem

Diagnose the situation so that you focus your attention on the problem. Helpful problem – solving techniques include using flowcharts to identify the expected steps of a process and cause – and – effect diagrams to define and analyze the root cause(s). This involve reviewing and documenting how processes currently work. It also involve evaluating the possible impact of new tools and revised policies in the development of your model.

2. **Generate Alternative Solutions**
Postpone the selection of one solution until several problem – solving alternatives have been proposed. Considering multiple alternatives can significantly enhance the value of an ideal solution. Once you have decided on the "what should be" model, this target standard becomes the basis for developing a road map for investigating alternatives. Brainstorm and team problem – solving techniques are both useful tools in this stage of problem solving.

Many alternative solutions to the problem should be generated before final evaluation. A common mistake in problem solving is that alternatives are evaluated as they are proposed, so the first acceptable solution is chosen, even if it is not the best fit. If we focus our attention on trying to get the result(s) we want, we may miss the potential for learning something new that will allow for real improvement in the problem – solving process.

3. **Evaluate and Select an Alternative**
Postpone the selection of one solution until several problem–solving alternatives have been developed and tested. Also, consider the cost implications and time of software development for the various alternatives and choose the best alternative.

4. Coding and Implementation
This is where computer comes in. it involves the programmer developing the program (or code) to implement the solution. Plan and implement a pilot test of the chosen alternative.

5. Error Correction and Debugging
Debugging is the process of removing errors from the program. There are two main types of errors that can affect your program. These are:

- **Syntax errors:** These are errors which occur as a result of not following the grammatical rules of a particular programming language. These errors include not using the appropriate punctuation. They are easy to detect because the compiler will tell you where such errors are in the program.
- **Logic errors:**These type of errors occur when a program has ran but is generating incorrect results. Examples include division by zero(0), etc. These types of errors are much difficult and harder to detect. The ability to detect logic errors comes with the experience of the programmer.

6. Testing and Program Correctness
Test the software or program with different data and parameters to ensure that the software is very robust and that it can withstand any data set and can also stand the test of time. Gather feedback from software users. Seek acceptance or consensus by all those affected. Establish ongoing measures and monitoring and evaluate long-term results.

Example 1
Write a Python program to find the sum of a series 1/1! + 2/2! + 3/3! + 4/4! + . . . + n/n!

Solution

You have been given a series 1/1! + 2/2! + 3/3! + 4/4! + . . . + n/n!Find out the sum of the series till nth term.

```
#Python program to find sum of series
defSumOfSeries(n):
        res = 0
        fact = 1
        fori in range(1, n + 1):
                fact * = i
                res = res + (i / fact)
        return res
n = 7
print ("Sum =", SumOfSeries(n))
```

Example 2

Given n and x, where n is the number of terms in the series and x is the value of the angle in degree. Write a Python program to find the sum of since series using the formula:

$Sin\, x = x - (x^3/3!) + (x^5/5!) - (x^7/7!) + . . .$

```
#Import Module
Import Math
#Create since function
def sin(x, n)
        since = 0
        fori in range(n):
                sign = (- 1) * x i
                pi = 22/7
                y = x * (pi / 180)
                since + = ((y **(2.0 * i + 1))/math factorial (2 * i + 1) * sign
return since
# Main ( )
# Enter Value in degree in x
x = 10
# Enter number of terms
```

```
n = 60
# call since function
print(round (sin (x, n), 2))
```

Example 3

Write a Python Program to print matrix in Z form

Solution

```
# Python program to print a square matrix in # Z form
array  =     [[1, 2, 3, 4,]
              [5, 6, 7, 8],
              [9, 10, 11, 12]
              [13, 14, 15, 16]]
n = len(array [0])
i = 0
for j in range(0, n – 1):
      print(array[i][1], end = " ")
cnt = 1
fori in range (0, n):
for j in range(n, 0, - 1):
if(j = = n – cnt):
print (arr[i][j], end = " "
break j
cnt + = 1
# Main ( )
i = n – 1
for j in range (0, n):
print (arry[i] [j], end = " ")
```

Exercise 4

Write a Python program to find the sum of series with n-th term as n^2 – (n – 1)^2

Solution

We are given an integer n and n-th term in a series as:
We want to find Sn mod $(10^9 + 7)$, where Sn is the sum of all the terms of the given series and
$Sn = T_1 + T_2 + T_3 + T_4 + \ldots + T_n$

Let's do some calculations before writing the program. First we need to reduce Tn to 2n – 1.

Given, $Tn = n^2 - (n-1)^2$

This can be written as

$Tn = n^2 - (n^2 - 2n + 1)$ by expanding (n – 1)2

Thus $Tn = n^2 - n^2 + 2n - 1$ (expansion)

$\therefore Tn = 2n - 1$

Now, let's find ΣTn

Recall that Tn = 2n – 1

$\therefore \Sigma Tn = \Sigma(2n - 1)$

Let's simplify the formula as:

$\Sigma(2n - 1) = 2 * \Sigma n - \Sigma 1$

i.e., $\Sigma(2n - 1) = 2 * \Sigma n - n$

whereΣn is the sum of first n natural numbers However, the sum of n natural numbers = n(n + 1)/2

Therefore, substituting this in the above equation, we have

$\Sigma Tn = (2 * (n) * (n + 1)/2 - n = n^2$

Suppose the value of n is very large, we will use the property of modular multiplication for calculating squares:

(a * b)%k = (a%k) * (b%k) % k

We then write the Python Program to find the sum.

# Python program to find sum of a given series.

```
mod = 344936985
```

```
deffinSum(n):

return((n % mod) * (n % mod)) % mod

# main ( )

n = 344936985

print(findSum(n))
```

Example 5
Write a Python program to find nth occurrence of substring in a string.

Solution
Given a string and a substring we write a Python program to find the n^th^ occurrence of the string.

```
#Python program to find the n^th occurrence of

#Substring

#Initializing values

ini-str = "abababababab"

sub-str = "ab"

occurrence = 4

# Finding nth occurrence of substring

Val = - 1

fori in range(0, occurrence):

val = ini-str find (Sub-str, val + 1)

print("nth occurrence is at", val)
```

#Python Program to find position of nth multiple of a number k in Fibonacci series

```
deffindPosition(k, n):

    f1 = 0

    f2 = 1

    i = 2;
    while i ! = 0:
        f3 = f1 + f2;
        f1 = f2;
        f2 = f3;
        if f2 %k = = 0:
            return n * i
        i + = 1
    return
# Multiple no.
n = 5,
#Number of whose multiple we are finding k = 4;
Print("Position of n\th multiple of k in "Fibonacci Series is", find Position (k, n)).
```

Factorial

A classic example of a recursive procedure is the function used to calculate the factorial of a natural number.

$$fact(n) = \begin{cases} 1 & if\ n = 0 \\ n \,.\, fact(n-1) & if\ n > 0 \end{cases}$$

Algorithm (Recursion):

```
Function factorial is:
input: integer n such that n > = 0
output: [n * (n – 1) * (n – 2) * - * 1]
    1. if n is 0, return 1
    2. otherwise, return [n * factorial (n – 1)]
end factorial
```

The function can also be written as a recurrence relation:

$b_n = nb_{n-1}$

$b_0 = 1$

This can be evaluated using the algorithm above

$$
\begin{aligned}
b_n &= nb_{n-1} \\
b4 &= 4 * b_3 \\
&= 4 * (3 * b_2) \\
&= 4 * (3 * (2 * b_1)) \\
&= 4 * (3 * 2 * (1 * b_0))) \\
&= 4 * (3 * (2 * (1 * 1))) \\
&= 4 * (3 * (2 * 1)) \\
&= 4 * (3 * 2) \\
&= 4 * 6 \\
&= 24
\end{aligned}
$$

The factorial function can also be described without using recursion by making use of the typical looping constructs found in imperative programming languages:

Algorithm (Iteration):

```
function factorial is:
input: integer n such that n >= 0
output: [n * (n – 1) * (n – 2) * . . . * 2 * 1]

  1. create new variable called running-total with a value of 1
  2. begin loop
     1. if n is 0, exit loop
     2. set running total to (running-total * n)
     3. decrement n
     4. repeat loop
  3. return running-total

end factorial
```

Greatest Common Divisor

The Euclidean algorithm which computes the greatest common divisor of two integers can b
written recursively as follows:

ience

typical

n be

Function definition:

$$\gcd(x, y) = \begin{cases} x & if\ y = 0 \\ \gcd(y, remainder(x, y) & if\ y > 0 \end{cases}$$

Algorithm (recursion)

```
functiongcd:
input: integer x, integer y such that x > 0 and y > = 0
output: gcd
    1. if y is 0, return x
    2. otherwise, return [gcd(y, {remainder of x/y)}]
endgcd
```

However, recursive relation for greatest common divisor, where x%y expresses the remainder of x/y

gcd (x, y) = gcd(y, x%y) if y $\neq$ 0

gcd(x, 0) = x

Computing the recurrence relation for x 36 and y = 6:

gcd (36, 6) = gcd(6, 36%6)
= gcd (6, 0)
= 6

Computing the recurrence relation for x = 111 and y = 259:

gcd (111, 259) = gcd(259, 111%259)
= gcd (259, 111)
= gcd (111, 259%111)
= gcd (111, 37)
= gcd (37, 111%37)
= gcd (37, 0)
= 37.

The recursive above is called tail recursive. It is equivalent to iterative algorithm as shown below.

Algorithm (Iteration):

functionged is:

input: integer x, integer y such that x > = y and y > = 0

The iterative algorithm as shown above requires a temporary variable, and with knowledge of the Euclidean algorithm it is move difficult to understand the process by simple inspection, although the two algorithm are very similar in their steps.

Exercises

1. What are the steps involved in problem-solving. What are the strategies of problem solving?
2. What do you understand by debugging? State and explain the types of errors involved in computer program or software.
3. Write a Python program to solve the quadratic equation $ax^2 + bx + c = 0$.