Experiment 1 21th March 2022

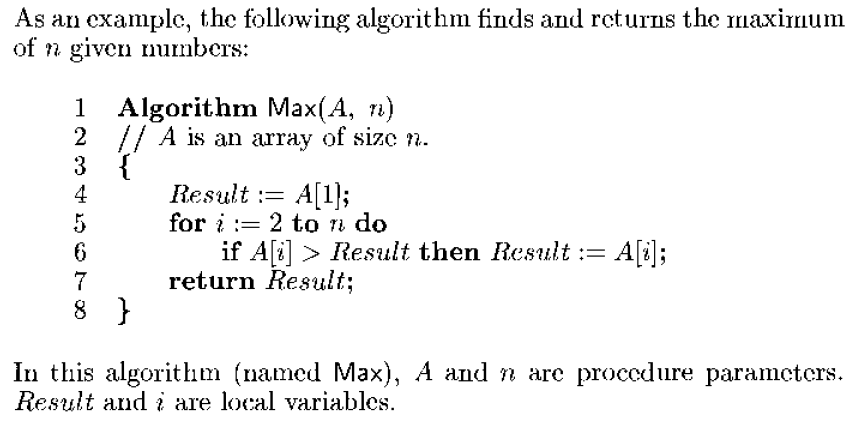
**Iteration and Recursion**

Aim: To study iteration and recursion.

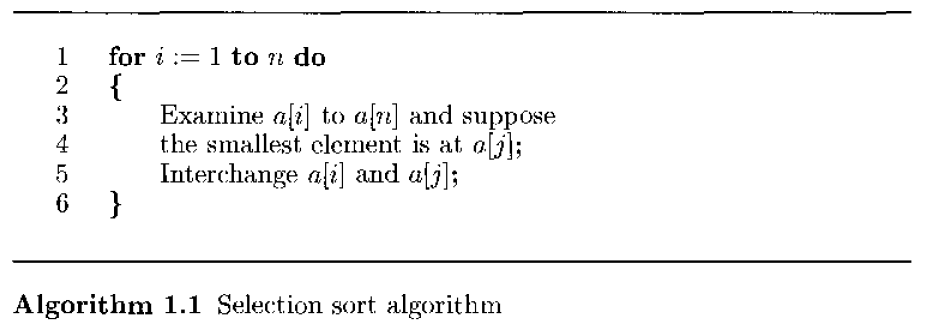
Theory:

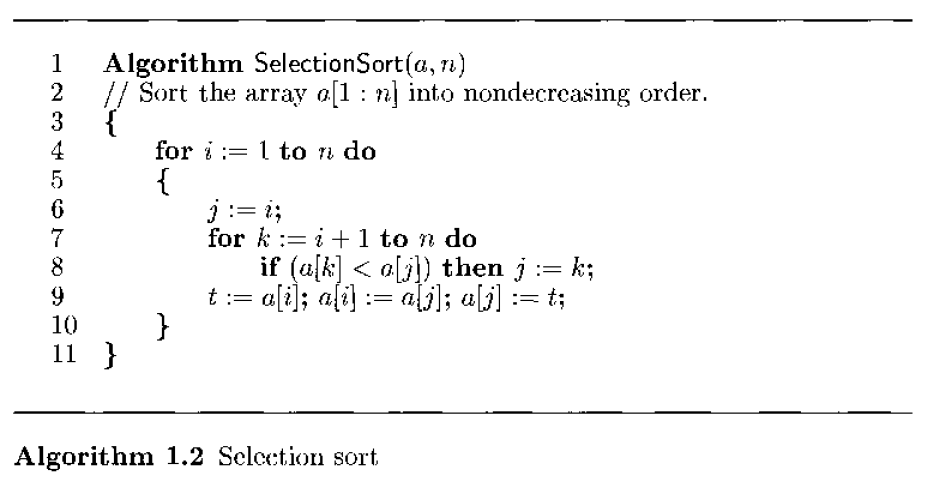
**Algorithms** are used to help design programs that perform particular tasks. Algorithms consist of steps that are carried out (performed) one after another. There are three basic building blocks (constructs) to use when designing algorithms: sequencing, selection, and iteration.

Sometimes an algorithm needs to repeat certain steps until told to stop or until a particular condition has been met. **Iteration** is the process of repeating steps. **Iteration allows us to simplify our algorithm by stating that we will repeat certain steps until told otherwise.** This makes designing algorithms quicker and simpler because they don’t have to include lots of unnecessary steps.

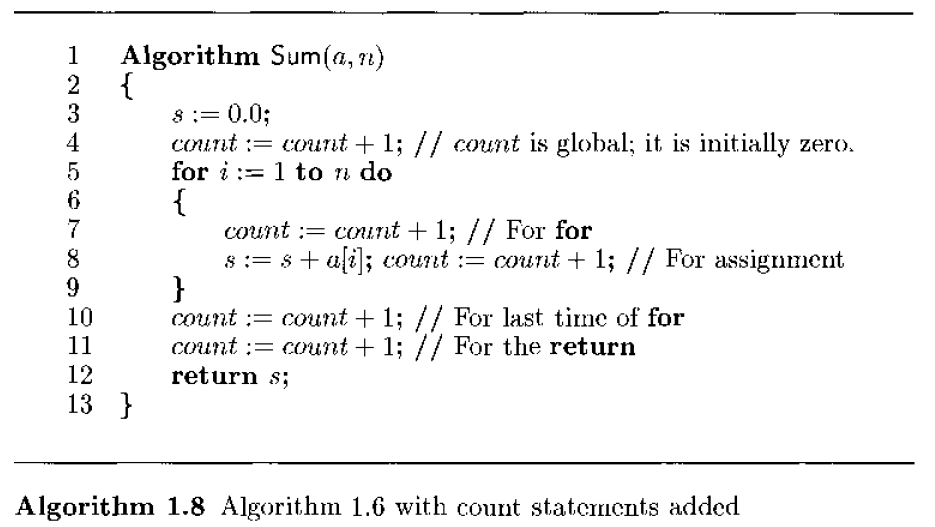


Once an algorithm has been designed and perfected, it must be translated – or programmed – into code that a computer can read. We create programs to implement algorithms. Algorithms consist of steps. Programs consist of statements. **A statement is a single instruction** - in other words, a single step. In programming, iteration is often referred to as ‘looping’, because when a program iterates it ‘loops’ to an earlier step.





We can *determine the number of steps needed by a program* to solve a particular problem instance in one of two ways. In the first method, we introduce a new variable, **count**, into the program. This a global variable with initial value 0. Statements to increment count by the appropriate amount are introduced into the program. This is done so that each time a statement in the original program is executed, count is incremented by the step count of that statement. The second method builds a table in which we list the total number of steps contributed by each statement. In this experiment we are studying the first method.



**Recursion** is a process in which a problem is defined in terms of itself. The problem is solved by repeatedly breaking it into smaller problems, which are similar in nature to the original problem. The smaller problems are solved, and their solutions are applied to get the final solution of the original problem. To implement recursion technique in programming, a function should be capable of calling itself. A recursive function is a function that calls itself.

E.g. void main ()

{

Rec ();

}

void Rec ()

{

Rec ();

}

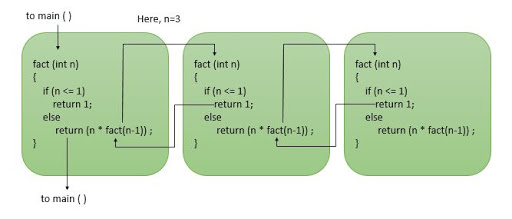
Here the function Rec () is calling itself inside its own function body, so Rec () is a recursive function. When main () calls Rec (), the code of Rec () will be executed and since there is a call to Rec () inside Rec (), again Rec () will be executed. A terminating condition is written inside the recursive function which ends this recursion. This **terminating condition** is also known as exit condition or the base case. This is the case when function will stop calling itself and will finally start returning.

Recursion proceeds by repeatedly breaking a problem into smaller versions of the same problem, till finally we get the smallest version of the problem which is simple enough to solve. The smallest version of the problem that can be solved without recursion and this is the base case.

The two main steps in writing a recursive function are:

1. Identification of the base case and its solution, i.e., the case where solution can be achieved without recursion. There may be more than one base case. The base case is important to terminate recursion.
2. Identification of the general case or the recursive case, i.e., the case in which recursive call will be made.

Flow of control in recursive function calls



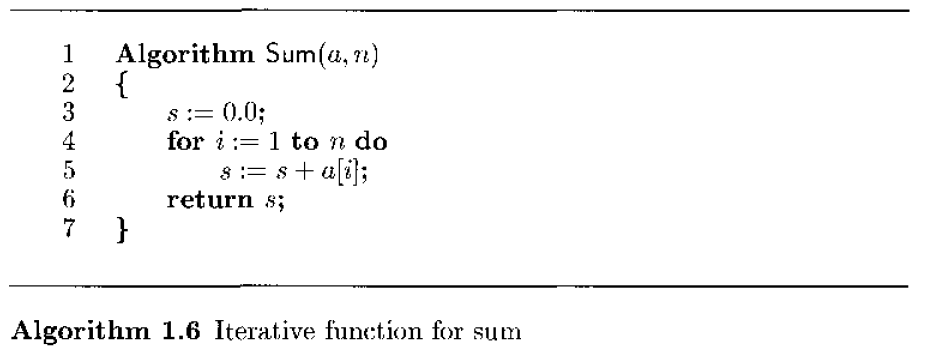
All recursive functions work in two phases – winding phase and unwinding phase. Winding phase begins when the recursive function is called for the first time and each recursive call continues the winding phase. The function keeps calling itself and no return statements are executed in this phase. This phase terminates when the terminating condition becomes true in a call. After this the unwinding phase begins and all the recursive function calls start returning in reverse order till the first instance of the function returns.

Task:

Write programs for the following algorithms:

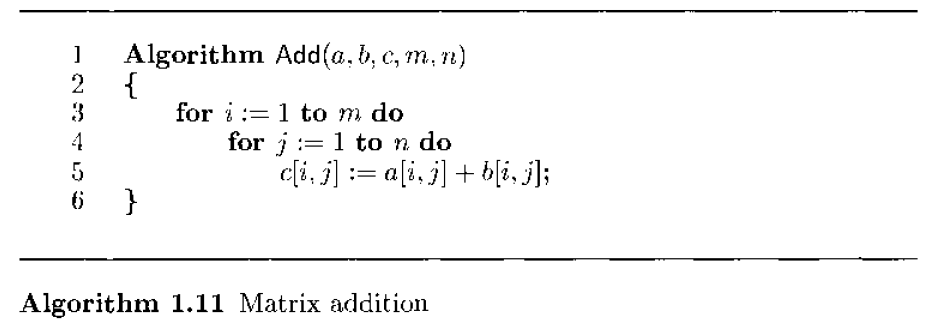
**Iteration:**

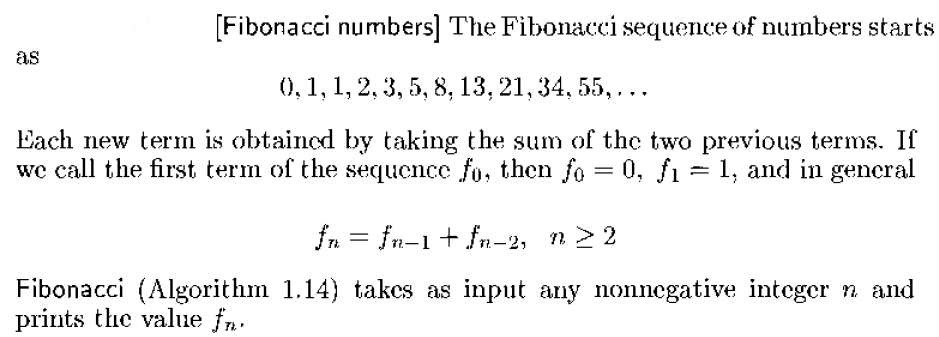
1. Find and return the maximum and minimum of n given numbers.
2. Selection sort
3. Iterative Sum without counting statements

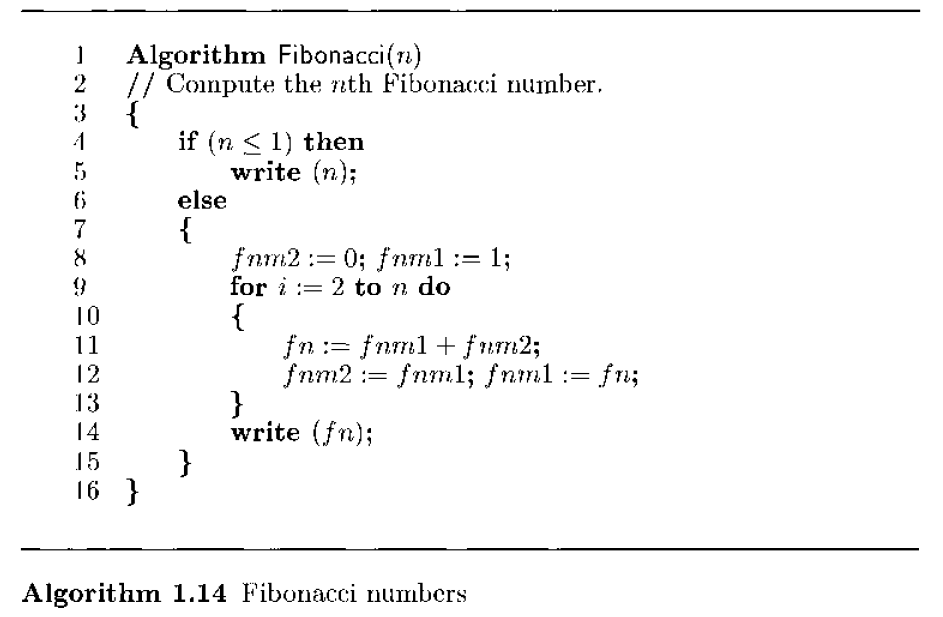


1. Iterative Sum with count statements.

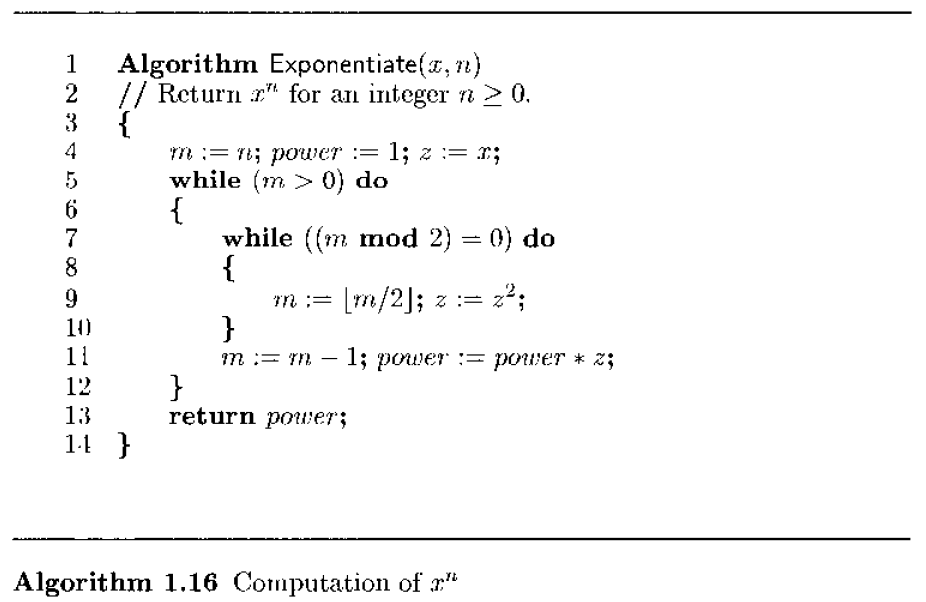
Add statements to increment count at each program step and display the total number of program steps at the end. Refer algorithm 1.8 above.

1. Add two m x n matrices.
2. Fibonacci numbers

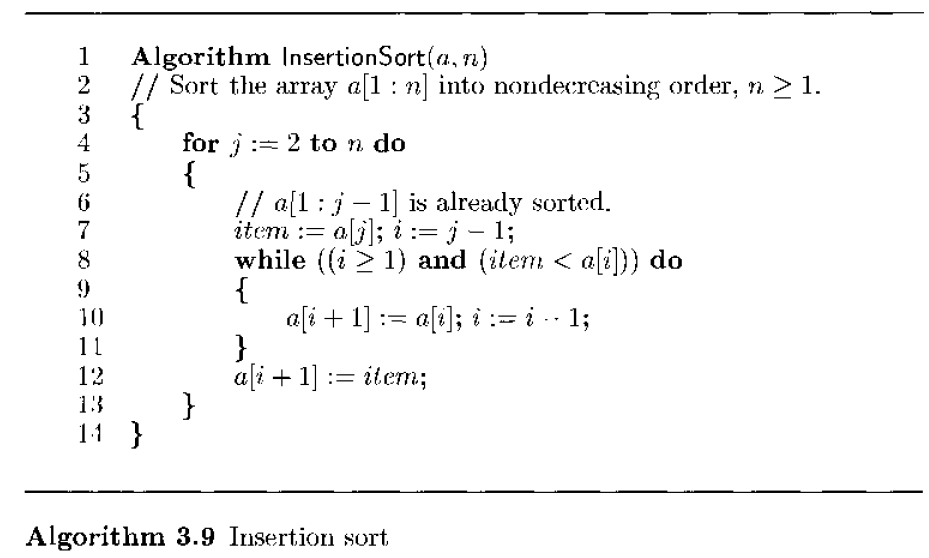




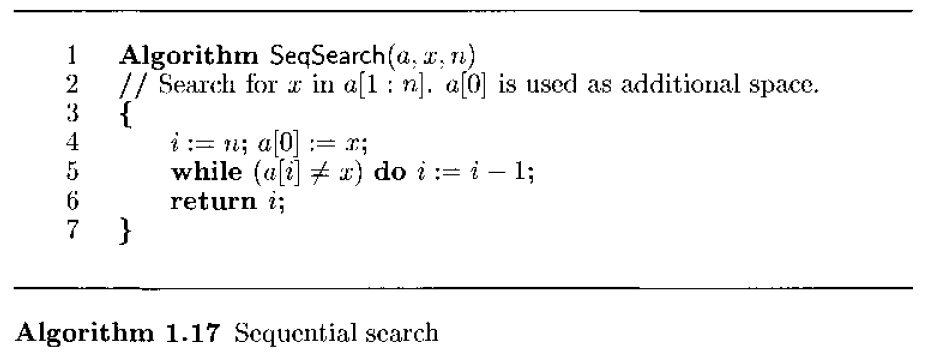
1. Exponentiation



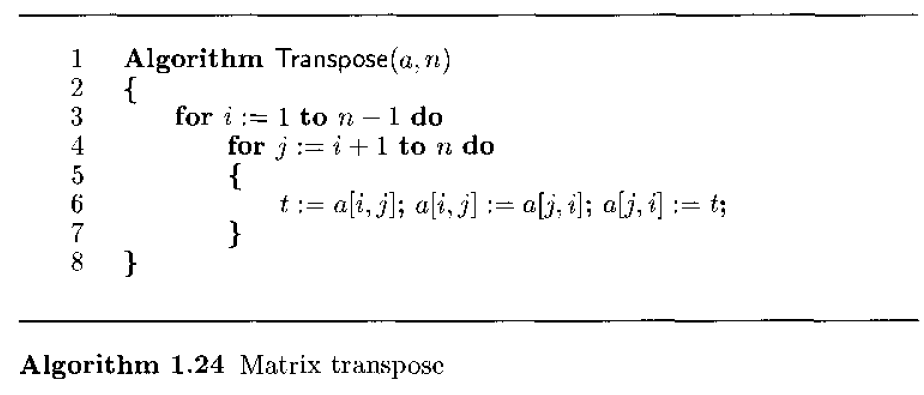
8. Insertion Sort



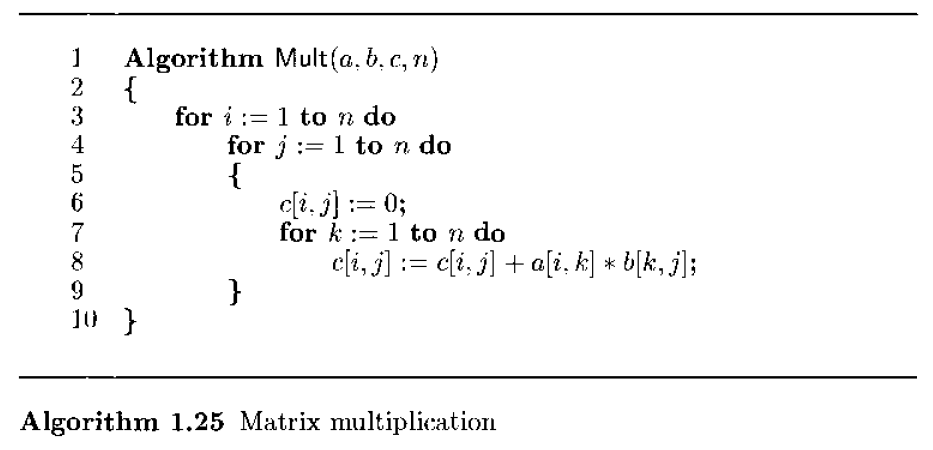
9. Sequential search



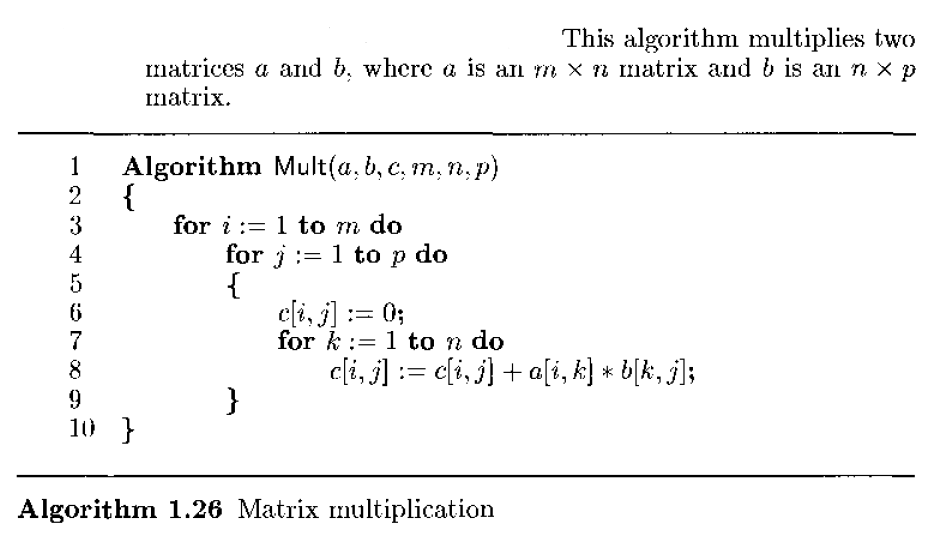
10. Matrix transpose



11. Multiplication of n x n matrices.

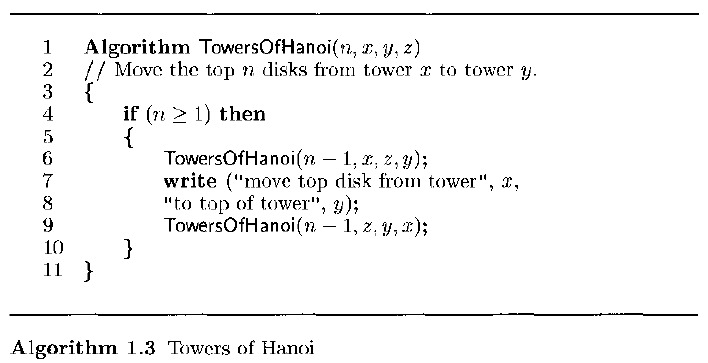


12. Multiplication of an “m x n” matrix and a “n x p” matrix.

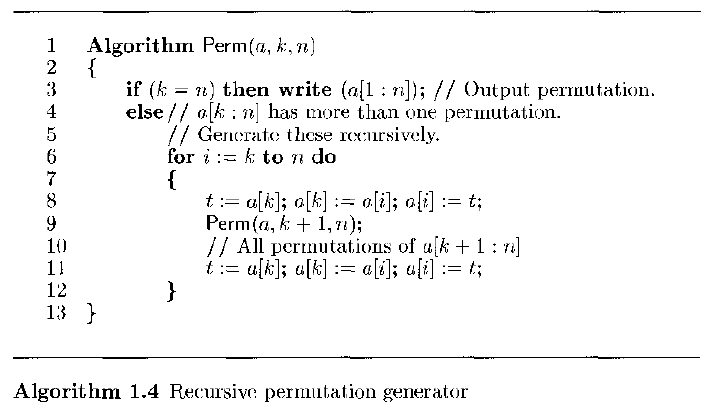


**Recursion:**

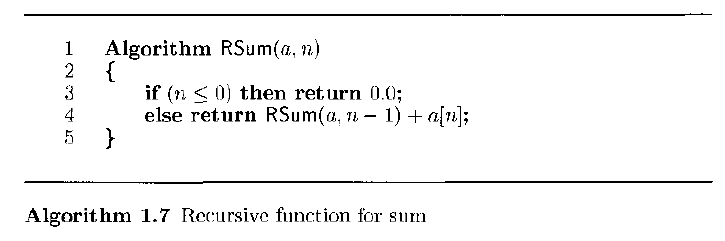
1. Towers of Hanoi,



1. Permutation Generator. E.g. input is {a, b}, output will be { (a, b), (b, a)}

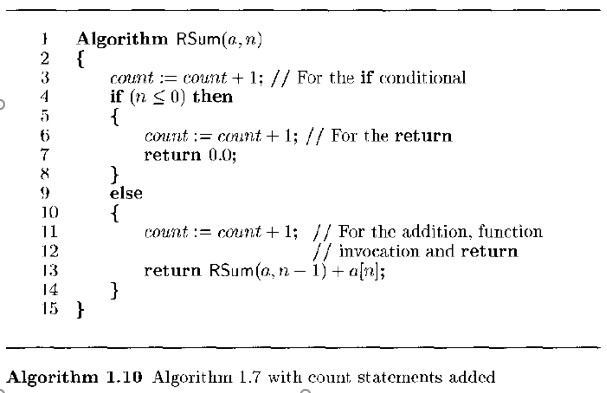


1. Recursive Sum without counting statements.



1. Recursive Sum with counting statements.

Add statements to increment count at each program step and display the total number of program steps at the end.



Programs and Output:

12 programs of iteration and 4 programs of recursion.

Conclusion: *Compare recursion and iteration.*