Notion of an Algorithm

An algorithm is a finite sequence of instructions each of which has a clear (unambiguous) meaning and whatever the input to the algorithm, the algorithm always terminates after executing the instructions a finite number of times.

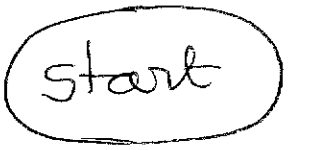
If there is no guarantee of termination, the word procedure is used instead of algorithm.

An algorithm is independent of implementation details such as programming language. An algorithm becomes a program when the instructions of a programming language are used to transform the algorithm into a form that can be executed on a computer. Data structures play an important role in this process.

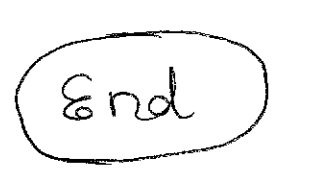
Algorithms + Data Structures = Programs

Algorithms are expressed usually in flow-chart form or using a suitable pseudocode. The idea of using a flow-chart or pseudocode is to express an algorithm in a programming language-independent notation.

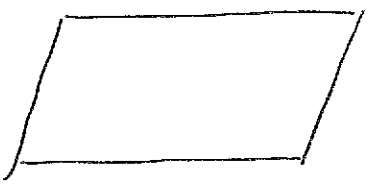
A flowchart primarily uses the following symbols:



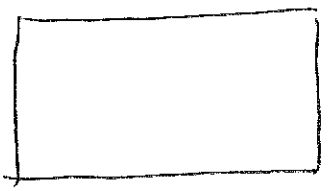
Beginning of the algorithm



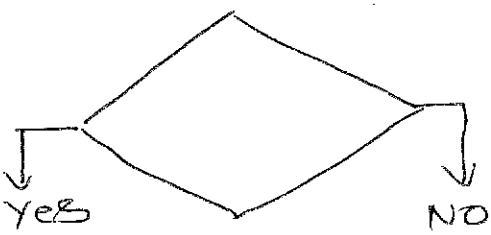
End of the algorithm



Data or input

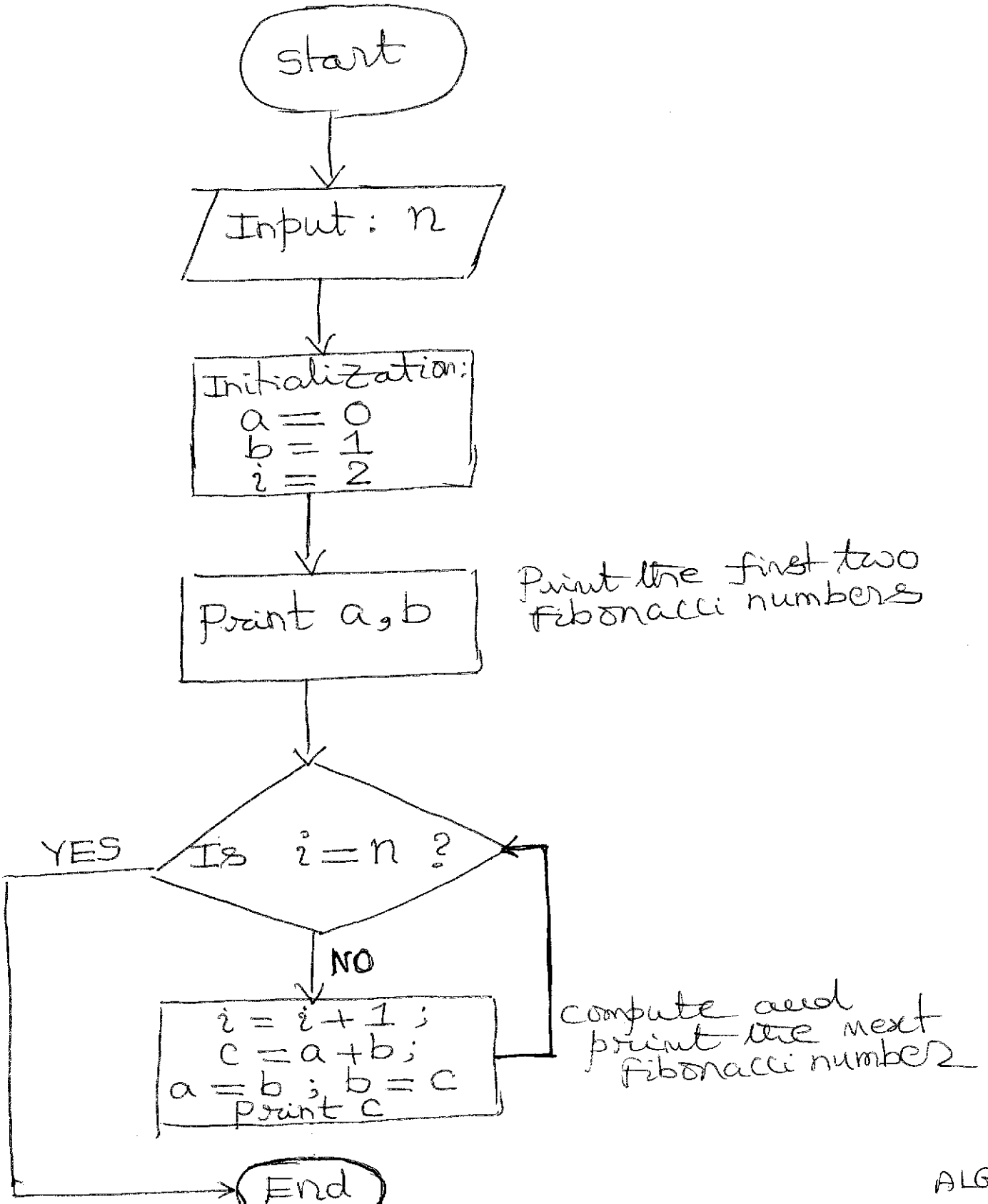


process



Decision

Algorithm for generating the first n numbers in the Fibonacci sequence 0, 1, 1, 2, 3, 5, 8, 13, 21, ...



A Pseudocode representation of the same would be:

1. Read the input n
2. Initialize: a = 0; b = 1; i = 2
3. Print a, b
4. If (i = n) goto step 6
5. i = i + 1; c = a + b;
   a = b; b = c;
   print c
6. End

The above algorithm uses a technique called Iteration. Iteration is the process of executing a set of instructions repetitively until a certain condition is satisfied. often, this is called "looping." All programming languages will have several constructs for implementing iterations.

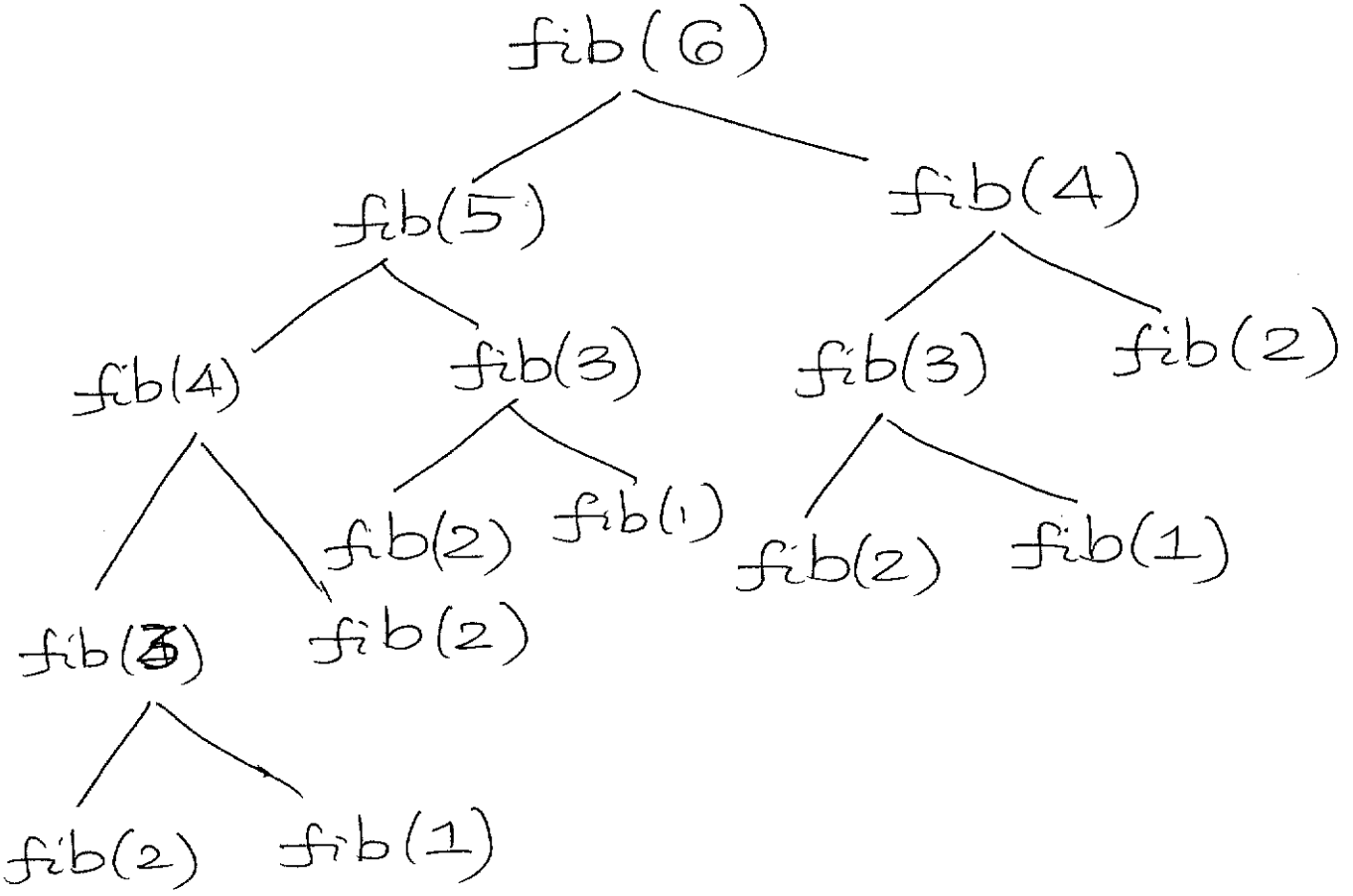
Fibonacci numbers can also be computed using "Recursion" which involves invoking the same function with a lower argument. A recursive function to compute the nth Fibonacci number will look like the following:

```
int fib(int n);
if (n=0) return 0;
else if (n=1) return 1;
else return (fib(n-1) + fib(n-2));
```

Note that the above is only pseudocode and not a C program.

It is very interesting to see the detailed trace of execution of the above recursive pseudocode to get an idea of how recursion works.

Iteration and Recursion are two fundamentally different paradigms. Often they are equally efficient. Sometimes recursion is more efficient; at other times iteration is more efficient.



The above shows the execution trace for computing fib(6) recursively. Note that many intermediate values are computed multiple times.

fib(6)	is	invoked	once
fib(5)	is	invoked	once
fib(4)	is	invoked	2 times
fib(3)	is	invoked	3 times
fib(2)	is	invoked	5 times
fib(1)	is	invoked	3 times