Algorithm Specification

Dr. Bibhudatta Sahoo

Communication & Computing Group

Department of CSE, NIT Rourkela

Email: bdsahu@nitrkl.ac.in, 9337938766, 2462358

Algorithm

- Algorithm: is a procedure that consists of a *finite set of instructions* which, given an *input* from some set of possible inputs, enables us to obtain an *output* if such an output exists or else obtain nothing at all if there is no output for that particular input through a *systematic execution* of the *instructions*.
- **Algorithm:** It's an organized logical sequence of the actions or the approach towards a particular problem.
- A programmer implements an algorithm to solve a problem.
- Algorithms are expressed using natural verbal but somewhat technical annotations.

Algorithm

An algorithm is a finite set of instructions that, is followed, accomplishes a particular task. In addition, all algorithms must satisfy the following criteria:

1. Input: Zero or more quantities
2. Output: At least one
3. Definiteness : Each instruction is clear and unambiguous
4. Finiteness: Algorithm terminates in finite number of steps
5. Effectiveness: Every instruction must be very basic and feasible.

Algorithm Description

How to describe algorithms independent of a programming language

- **Text (Pseudo-Code)** = a description of an algorithm that is
  - more structured than usual prose but
  - less formal than a programming language

- **Diagrams** e.g., flowchart, sequence diagram (good for complex interactions among objects) Communicate visually, however, time-consuming to create, hard to maintain, separate from source file.

Example: find the maximum element of an array.

Algorithm arrayMax(A, *n*):
Input: An array A storing n integers.
Output: The maximum element in A.
currentMax ← A[0]
for *i* ← **1 to *n* -1 do**
    if *currentMax* < A[i] **then** *currentMax* ← A[i]
return *currentMax*

What is Pseudocode?

- **Pseudocode** (pronounced SOO-doh-kohd) is a detailed yet readable description of what a computer program or algorithm must do, expressed in a formally-styled natural language rather than in a programming language.
- **Pseudocode** is sometimes **used** as a detailed step in the process of developing a program
- **Pseudocode** is a plain-text description of a piece of code or an algorithm. It's not actually coding; there is no script, no files, and no programming. As the name suggests, it's "fake code".
- **Pseudocode** is not written in any particular programming language. It's written in plain English that is clear and easy to understand.
- Writing a full program in pseudocode requires a lot of different statements and keywords much like regular programming. In fact, once you get far enough along in your pseudocode it will start to look very close to a real program.

Pseudo code

- **Pseudo code:** It's simply an implementation of an algorithm in the form of annotations and informative text written in plain English.
- It has no syntax like any of the programming language and thus can't be compiled or interpreted by the computer.

Advantages of Pseudocode

- Improves the readability of any approach. It's one of the best approaches to start implementation of an algorithm.
- Acts as a bridge between the program and the algorithm or flowchart. Also works as a rough documentation, so the program of one developer can be understood easily when a pseudo code is written out. In industries, the approach of documentation is essential. And that's where a pseudo-code proves vital.
- The main goal of a pseudo code is to explain what exactly each line of a program should do, hence making the code construction phase easier for the programmer.

Difference between Algorithm, Pseudocode and Program

- **Algorithm : Systematic logical approach** which is a well-defined, step-by-step procedure that allows a computer to solve a problem.

- **Pseudocode :** It is a **simpler version of a programming code** in plain English which uses short phrases to write code for a program before it is implemented in a specific programming language.

- **Program :** It is exact code written for problem **following all the rules of the programming language.**

Pseudocode Conventions

[1] Comments begin with / / and continue until the end of line.

[2] Blocks are indicated with ·matching braces: { and }. A compound statement (i.e., a collection of simple statements] can be represented as a block. The body of a procedure also forms a block. Statements are delimited by ; .

[3] An identifier begins with a letter. The data types of variables are not explicitly declared. The types will be clear from the context.

Pseudocode Conventions

[4] Assignment of values to variables is done using the assignment statement

(variable) := (expression);

[5] There are two boolean values **true** and **false**. In order to produce these values, the ***logical operators*** and, **or,** and **not** and the ***relational operators*** $<, \leq, =, \neq, \geq$, and $>$ are provided.

[6] Elements of multidimensional arrays are accessed using [and]. For example, if A is a two dimensional array, the $(i, j)^{th}$ element of the array is denoted as A[i, j]. Array indices start at zero.

Pseudocode Conventions

[7] The looping statements supported by the pseudo code are: for, while, and repeat - until.

<u>The while loop takes the following form</u>

```
While < condition> do
{
        <statement1>
}
```

Pseudocode Conventions

8. A conditional statement has the following forms:

   if $\langle condition \rangle$ **then** $\langle statement \rangle$
   if $\langle condition \rangle$ **then** $\langle statement\ 1 \rangle$ **else** $\langle statement\ 2 \rangle$

   Here $\langle condition \rangle$ is a boolean expression and $\langle statement \rangle$, $\langle statement\ 1 \rangle$, and $\langle statement\ 2 \rangle$ are arbitrary statements (simple or compound).

   We also employ the following **case** statement:

   case
   {
       :$\langle condition\ 1 \rangle$: $\langle statement\ 1 \rangle$
       $\vdots$
       :$\langle condition\ n \rangle$: $\langle statement\ n \rangle$
       :else: $\langle statement\ n+1 \rangle$
   }

Pseudocode Conventions

[9] Input and output are done using the instructions *read* and *write*. No format is used to specify the size of input or output quantities.

[10] There is only one type of procedure: *Algorithm*. An algorithm consists of a heading and a body.

The heading takes the form

Algorithm *Name ((parameter list))*

```
Algorithm Max(A, n)
// A is an array of size n.
{
    Result := A[1];
    for i := 2 to n do
        if A[i] > Result then Result := A[i];
    return Result;
}
```

Not an algorithm

- [Selection sort]

```
for i := 1 to n do
{
    Examine a[i] to a[n] and suppose
    the smallest element is at a[j];
    Interchange a[i] and a[j];
}
```

Algorithm 1.1 Selection sort algorithm

Algorithm finds and returns the maximum of *n* given numbers:

```
Algorithm Max(A, n)
// A is an array of size n.
{
    Result := A[1];
    for i := 2 to n do
        if A[i] > Result then Result := A[i];
    return Result;
}
```

In this algorithm(named Max), A and n are procedure parameters. ***Result*** and ***i*** are local variables

Selection Sort

```
Algorithm SelectionSort(a, n)
// Sort the array a[1 : n] into nondecreasing order.
{
    for i := 1 to n do
    {
        j := i;
        for k := i + 1 to n do
            if (a[k] < a[j]) then j := k;
        t := a[i]; a[i] := a[j]; a[j] := t;
    }
}
```

Theorem 1.1 Algorithm SelectionSort(a, n) correctly sorts a set of $n \geq 1$ elements; the result remains in $a[1:n]$ such that $a[1] \leq a[2] \leq \cdots \leq a[n]$.

Correctness of an algorithm

- In theoretical computer science, **correctness** of an **algorithm** is asserted when it is said that the **algorithm** is correct with respect to a specification.
- Functional **correctness** refers to the input-output behaviour of the **algorithm** (i.e., for each input it produces the expected output).
- An algorithm is correct if:
  - for any correct input data: it stops and it produces correct output.
  - Correct input data: satisfies pre-condition
  - Correct output data: satisfies post-condition

Total Correctness of Algorithm

- **Definition:** An algorithm which for any correct input data: (i) **stops** and (ii) **returns correct output** is called **totally correct** for the given specification.

These split into 2 sub-properties in the definition above.

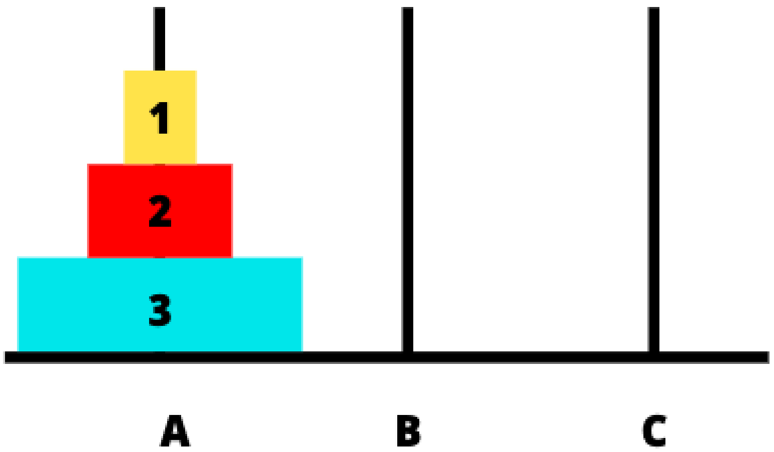
- **correct input data** is the data which satisfies the initial condition of the specification
- **correct output data** is the data which satisfies the final condition of the specification

Recursive Algorithms

- A recursive function is a function that is defined in terms of itself. Similarly, an algorithm is said to be recursive if the same algorithm is invoked in the body.
- An algorithm that calls itself is direct recursive. Algorithm A is said to be indirect recursive if it calls another algorithm B which in turn calls A.
- Directly recursive: a function that calls itself
- Indirectly recursive: a function that calls another function and eventually results in the original function call
- Tail recursive function: function in which the last statement executed is the recursive call.

Recursive algorithm : Towers of Hanoi

```
Algorithm TowersOfHanoi(n, x, y, z)
// Move the top n disks from tower x to tower y.
{
    if (n ≥ 1) then
    {
        TowersOfHanoi(n − 1, x, z, y);
        write ("move top disk from tower", x,
        "to top of tower", y);
        TowersOfHanoi(n − 1, z, y, x);
    }
}
```



Recursive binary search

```
Algorithm BinSrch(a, i, l, x)
// Given an array a[i : l] of elements in nondecreasing
// order, 1 ≤ i ≤ l, determine whether x is present, and
// if so, return j such that x = a[j]; else return 0.
{
    if (l = i) then  // If Small(P)
    {
        if (x = a[i]) then return i;
        else return 0;
    }
    else
    { // Reduce P into a smaller subproblem.
        mid := ⌊(i + l)/2⌋;
        if (x = a[mid]) then return mid;
        else if (x < a[mid]) then
                return BinSrch(a, i, mid − 1, x);
            else return BinSrch(a, mid + 1, l, x);
    }
}
```

Iterative binary search

```
Algorithm BinSearch(a, n, x)
// Given an array a[1 : n] of elements in nondecreasing
// order, n ≥ 0, determine whether x is present, and
// if so, return j such that x = a[j]; else return 0.
{
    low := 1; high := n;
    while (low ≤ high) do
    {
        mid := ⌊(low + high)/2⌋;
        if (x < a[mid]) then high := mid − 1;
        else if (x > a[mid]) then low := mid + 1;
             else return mid;
    }
    return 0;
}
```

Thanks for Your Attention!

Exercises

Exercise: Write algorithms in Pseudo code

1. Present an algorithm that searches an unsorted array a[l: n] for the element x. If x occurs, then return a position in the array; else return zero.
2. The factorial function n! has value 1, when n $\leq$ 1and value n * (n-1)!, when n > 1. Write both a recursive and an iterative algorithm to Compute n!.
3. The Fibonacci numbers are defined $f_0 = 0$, $f_1 = 1$, and $f_j = f_{j-1} + f_{j-2}$ for j > 1. Write both a recursive and an iterative algorithm to compute n^{th} Fibonacci number.
4. Give an algorithm to solve the following problem: Given n, a positive integer, determine whether n is the sum of all of its divisors, that is, whether n is the sum of all t such that $1 \leq t < n$, and t divides n.