# PSEUDO-CODE

Komate AMPHAWAN

## PRELIMINARY

- Pseudocode is an informal tool that you can use to plan out your algorithms.
- As you begin to write more complex code, it can be hard to keep an entire program in your head before coding it.
- Think of pseudocode as a step-by-step verbal outline of your code that you can later transcribe into a programming language.

## PRELIMINARY

- It is a combination of human language and programming language: it mimics the syntax of actual computer code, but it is more concerned with readability than with technical specificity.

## KNOW WHAT PSEUDOCODE IS.

- Pseudocode is a step-by-step verbal outline of your code that you can gradually transcribe into programming language.
- Many programmers use it to plan out the function of an algorithm before setting themselves to the more technical task of coding.
- Pseudocode serves as an informal guide, a tool for thinking through program problems, and a communication device that can help you explain your ideas to other people.

## UNDERSTAND WHY PSEUDOCODE IS USEFUL.

- Pseudocode is used to show how a computing algorithm should and could work.
- Coders often use pseudocode as an intermediate step in programming, in between the initial planning stage and the stage of writing actual, executable code.
- Good pseudocode can become comments in the final program, guiding the programmer in the future when debugging the code, or revising it in the future.

- Pseudocode can also be useful for:
    - Describing how an algorithm should work. Pseudocode can illustrate where a particular construct, mechanism, or technique could or must appear in a program. Senior programmers often use the pseudocode to quickly explain the steps their junior programmers need to follow in order to accomplish a required task.
    - Explaining a computing process to less technical people. Computers need a very strict input syntax to run a program, but humans (especially non-programmers) may find it easier to understand a more fluid, subjective language that clearly states the purpose of each line of code.

- Designing code in a collaborative development group. High-level software architects will often include pseudocode into their designs to help solve a complex problem they see their programmers running into. If you are developing a program along with other coders, you may find that pseudocode helps make your intentions clear.

## REMEMBER THAT PSEUDOCODE IS SUBJECTIVE AND NONSTANDARD.

- There is no set syntax that you absolutely must use for pseudocode, but it is common professional courtesy to use standard pseudocode structures that other programmers can easily understand.
- If you are coding a project by yourself, then the most important thing is that the pseudocode helps you structure your thoughts and enact your plan.

- If you are working with others on a project—whether they are your peers, junior programmers, or non-technical collaborators—it is important to use at least some standard structures so that everyone else can easily understand your intent.
    - If you are enrolled in a programming course at a university, a coding camp, or a company, you will likely be tested against a taught pseudocode "standard". This standard often varies between institutions and teachers.

- Clarity is a primary goal of pseudocode, and it may help if you work within accepted programming conventions. As you develop your pseudocode into actual code, you will need to transcribe it into a programming language – so it can help to structure your outline with this in mind.

## UNDERSTAND ALGORITHMS.

- An algorithm is a procedure for solving a problem in terms of the actions that a program will take and the order in which it will take those actions.
- An algorithm is merely the sequence of steps taken to solve a problem.

- The steps are normally "sequence," "selection, " "iteration," and a case-type statement.
  - In C, "sequence statements" are imperatives.
  - The "selection" is the "if then else" statement.
  - The iteration is satisfied by a number of statements, such as the "while," " do," and the "for."
  - The case-type statement is satisfied by the "switch" statement.

## REMEMBER THE THREE BASIC CONSTRUCTS THAT CONTROL ALGORITHM FLOW.

- If you can implement a "sequence" function, a "while" (looping) function, and an "if-then-else" (selection) function, then you have the basic tools that you need to write a "proper" algorithm.
  - SEQUENCE is a linear progression where one task is performed sequentially after another. For example:
    - READ height of rectangle
    - READ width of rectangle
    - COMPUTE area as height times width

- WHILE is a loop (repetition) with a simple conditional test at its beginning. The beginning and ending of the loop are indicated by two keywords WHILE and ENDWHILE. The loop is entered only if the condition is true. For example:
    - WHILE Population < Limit
        - Compute Population as Population + Births - Deaths
    - ENDWHILE

- IF-THEN-ELSE is a decision (selection) in which a choice is made between two alternative courses of action. A binary choice is indicated by these four keywords: IF, THEN, ELSE, and ENDIF. For example:
  - IF HoursWorked > NormalMaximum THEN
    - Display overtime message
  - ELSE
    - Display regular time message
  - ENDIF

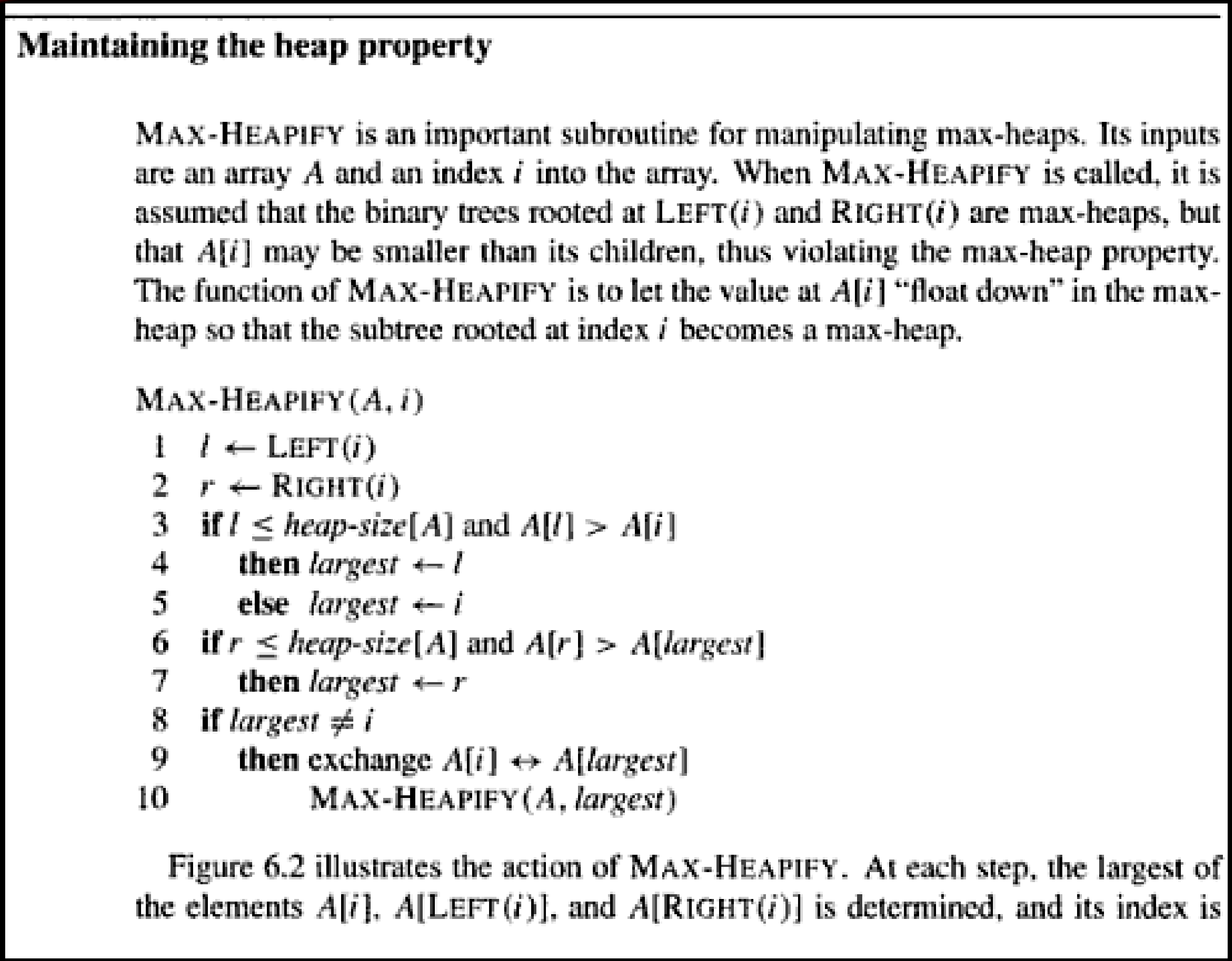
## REFERENCES

- http://www.wikihow.com/Write-Pseudocode

## EXAMPLE

**Algorithm 0.0.1:** REDUCE$(projection, x, y, f)$

```
for $i \leftarrow 1$ to $y/f$
  for $j \leftarrow 1$ to $x/f$
    $sum \leftarrow 0$
    for $m \leftarrow 1$ to $f$
      for $n \leftarrow 1$ to $f$
        $sum = sum + projection[i * f + m][j * f + n]$
    $reducedProjection[i][j] = sum/(f * f)$
return $(reducedProjection)$
```



**Maintaining the heap property**

MAX-HEAPIFY is an important subroutine for manipulating max-heaps. Its inputs are an array $A$ and an index $i$ into the array. When MAX-HEAPIFY is called, it is assumed that the binary trees rooted at LEFT($i$) and RIGHT($i$) are max-heaps, but that $A[i]$ may be smaller than its children, thus violating the max-heap property. The function of MAX-HEAPIFY is to let the value at $A[i]$ "float down" in the max-heap so that the subtree rooted at index $i$ becomes a max-heap.

```
MAX-HEAPIFY($A, i$)
$l \leftarrow$ LEFT($i$)
$r \leftarrow$ RIGHT($i$)
if $l \le heap\text{-}size[A]$ and $A[l] > A[i]$
   then $largest \leftarrow l$
   else $largest \leftarrow i$
if $r \le heap\text{-}size[A]$ and $A[r] > A[largest]$
   then $largest \leftarrow r$
if $largest \ne i$
   then exchange $A[i] \leftrightarrow A[largest]$
        MAX-HEAPIFY($A, largest$)
```

Figure 6.2 illustrates the action of MAX-HEAPIFY. At each step, the largest of the elements $A[i]$, $A[\text{LEFT}(i)]$, and $A[\text{RIGHT}(i)]$ is determined, and its index is

## Q & A