Pseudocode

In lectures, algorithms will often be expressed in pseudocode, a mixture of code and English. While understanding pseudocode is usually not difficult, writing it can be a challenge.

One example of pseudocode, used in this course, is presented in Section 2. Section 3 contains examples of pseudocode found in various textbooks.

1 Guidelines for writing pseudocode

Why use pseudocode at all? Pseudocode strikes a sometimes precarious balance between the understandability and informality of English and the precision of code. If we write an algorithm in English, the description may be at so high a level that it is difficult to analyze the algorithm and to transform it into code. If instead we write the algorithm in code, we have invested a lot of time in determining the details of an algorithm we may not choose to implement (as we typically wish to analyze algorithms before deciding which one to implement). The goal in writing pseudocode, then, is to provide a high-level description of an algorithm which facilitates analysis and eventual coding (should it be deemed to be a “good” algorithm) but at the same time suppresses many of the details that vanish with asymptotic notation. Finding the right level in the tradeoff between readability and precision can be tricky. If you have questions about the pseudocode you are writing on an assignment, please ask one of the course personnel to look it over and give you feedback (preferably before you hand it in so you can change it if necessary). Just as a proof is written with a type of reader in mind (hence proofs in undergraduate textbooks tend to have more details than those in journal papers), algorithms written for different audiences may be written at different levels of detail. In assignments and exams for the course, you need to demonstrate your knowledge without obscuring the big picture with unneeded detail. Here are a few general guidelines for checking your pseudocode:

1. Mimic good code and good English. Using aspects of both systems means adhering to the style rules of both to some degree. It is still important that variable names be mnemonic, comments be included where useful, and English phrases be comprehensible (full sentences are usually not necessary).

2. Ignore unnecessary details. If you are worrying about the placement of commas, you are using too much detail. It is a good idea to use some convention to group statements (begin/end, brackets, or whatever else is clear), but you shouldn’t obsess about syntax.

3. Don’t belabour the obvious. In many cases, the type of a variable is clear from context; unless it is critical that it is specified to be an integer or floating point number, for example, it is often unnecessary to make it explicit.

4. Take advantage of programming shorthands. Using branching or looping structures is more concise than writing out the equivalent in English; general constructs that are not
peculiar to a small number of languages are good candidates for use in pseudocode. Using parameters in specifying procedures is concise, clear, and accurate, and hence should not be omitted from pseudocode.

5. Consider the context. If you are writing an algorithm for quicksort, the statement Use quicksort to sort the values is hiding too much detail; if we have already studied quicksort in class and later use it as a subroutine in another algorithm, the statement would be appropriate to use.

6. Don’t lose sight of the underlying model. It should be possible to “see through” your pseudocode to the model below; if not (that is, you are not able to analyze the algorithm easily), it is written at too high a level.

7. Check for balance. If the pseudocode is hard for a person to read or difficult to translate into working code (or worse yet, both!), then something is wrong with the level of detail you have chosen to use.

2 Pseudocode used in the course

The pseudocode used in the course will adhere to the following conventions:

- Assignment statements are shown using ←, like in Example 3. Other common conventions are the use of = (Example 4) and := (Example 2).
- Types of variables are not listed explicitly, unlike in Examples 2 (which uses integer) and 4 (which uses int).
- Indentation is used to group statements, like in Python.
- The word return is used to indicate that a value is returned.
- Branching and looping mimic Python; thus, if and else are used, but unlike in Examples 1–3, not then.
- The use of a function or procedure is indicated by putting the arguments in parentheses, separated by commas, after the name of the function or procedure.
- For the ease of analysis, almost always a line will contain a finite number of simple tasks (cost $O(1)$) or the use of a function or procedure.

3 Pseudocode style examples

Various styles of pseudocodes can be observed in these examples of the binary search algorithm. Not all of these examples are worth emulating, as some are too detailed and some are hard to understand.
procedure SEARCH(a, f, ℓ):
if f > ℓ then return "no"
else
    if a = A⌊(f+ℓ)/2⌋ then return "yes"
    else
        if a < A⌊(f+ℓ)/2⌋ then
            return SEARCH(a, f, ⌊(f+ℓ)/2⌋ - 1)
        else return SEARCH(a, ⌊(f+ℓ)/2⌋ + 1, ℓ)


function binarysearch(v:integer):integer;
var x, ℓ, r:integer;
begin
    ℓ:=1; r:=N;
    repeat
        x:=(ℓ+r)div 2;
        if v<a[x].key then r:=x-1 else ℓ:=x+1
    until (v=a[x].key) or (ℓ > r);
    if v=a[x].key
        then binarysearch:=x
        else binarysearch:=N + 1
end;


function BinarySearchLoopUp(key K, table T[0..n-1]): info
{Return information stored with key K in T, or Λ if K is not T}
    Left ← 0
    Right ← n-1
    repeat forever
        if Right < Left then
            return Λ
        else
            Middle ← ⌊ (Left + Right)/2 ⌋
            if K = Key(T[Middle]) then return info(T[Middle])
            else if K < Key(T[Middle]) then Right ← Middle - 1
            else Left ← Middle + 1

Example 4 Computer Algorithms: Introduction to Design and Analysis, Baase and Van Gelder, 2000, p. 129
int binarySearch(int[], E, int first, int last, int K)
    if (last < first)
        index = -1;
    else
        int mid = (first + last)/2;
        if (K == E[mid])
            index = mid;
        else if (K < E[mid])
            index = binarySearch(E, first, mid-1, K);
        else
            index = binarySearch(E, mid+1, last, K);
    return index;