Pseudocode Guide

1 Pseudocode in LaTeX

In our study of algorithms, we'll need to be able to communicate the instructions that compose our algorithms efficiently. As computer scientists, we tend to communicate these precise instructions through the use of programming languages. However, the major downside of using a programming language is that it requires the person you are trying to communicate with to understand the language's syntax. To combat this, many algorithms are instead communicated via *pseudocode*.

In the `cs1570.cls` class file we've already imported the `algorithm` and `algpseudocodex` packages! If you have any questions about this document, or pseudocode in general ask on EdStem or stop by TA hours!

1.1 The `algorithm` environment

The `algorithm` package provides the `algorithm` environment which gives you a fancy-looking block to place your algorithm(s) in. You can title the block using the `\caption{...}` command. You might also find it helpful to pass the optional `H` parameter to the environment to position the box in approximately the same place it appears relative to the source TeX code.

```
\begin{algorithm}[H]
    \caption{Very Important Algorithm!}
\end{algorithm}
```

Algorithm 1 Very Important Algorithm!

1.2 The `algorithmic` environment

You'll be writing your pseudocode in the `algorithmic` environment provided by `algpseudocodex`. The `algorithmic` environment has an optional parameter that labels every nth line number. For example, passing in `[5]` will label lines 5, 10, 15, etc.
Note: It is not necessary to wrap an `algorithmic` environment in an `algorithm` environment.

Here's a list of helpful commands you might want to use in the algorithmic environment!

- `\Require`, `\Ensure` – defines the input/output of a function (alias `\Input/\Output`)
- `\Function...EndFunction` – defines a function block
- `\If`, `\ElsIf`, `\Else`, `\EndIf` – defines an if/else if/else block.
- `\While{condition}...\EndWhile` – defines a while block
- `\For{condition}...\EndFor` – defines a for block
- `\Call{function name}{args}` – calls a function
- `\State` – makes a new line
- `\Return` – makes a return statement (use `\State` to put this on a new line!)
- `\chigh{color}{...}`, `\cstring{color}{...}` – highlights a block/string of code

```
\begin{algorithm}[H]
\caption{``Best'' Sorting Algorithm}
\begin{algorithmic}[1]
    \Require{An array, $A$}
    \Ensure{The sorted array}
    \Function{BubbleSort}{$A$}
        \State $n \gets$ length of $A$
        \State swapped $\gets$ true
        \chigh{yellow}{\While{swapped}
            \State swapped $\gets$ false
            \For{$i \in [1, n - 1]$}
            \If{\cstring{red!50}{
                $A[i - 1] > A[i]$}}
            \State\Call{Swap}
            {$A[i - 1], A[i]$}
            \State swapped $\gets$ true
            \EndIf
            \EndFor
        \EndWhile}
    \EndFunction
    \ end{algorithmic}
\end{algorithm}
```

Algorithm 2 "Best" Sorting Algorithm

```
Input: An array, A
Output: The sorted arrays
function BubbleSort(A)
    n ← length of A
    swapped ← true
    while swapped do
        swapped ← false
        for i ∈ [1, n − 1] do
            if A[i − 1] > A[i] then
                Swap(A[i − 1], A[i])
                swapped ← true
```

2 Tips for Writing Pseudocode

Now that you know how to write pseudocode, you'll want to be sure you write clear, concise, and effective pseudocode! Nothing's worse than spending forever on TEX'ing something, only to find out that you weren't able to get your idea across. Here are a couple of tips!

1. **Indentation is your friend.** – As your pseudocode gets longer/deeply nested because of additional loops or control structures you may find that it helps to indent your LATEX source code as if it were actual code! This especially helps with making sure you match all of your start and end tags.

2. **Less is more.** – As with proofs, having longer solutions doesn't get you any extra points, but can lose you points if we don't understand what you are doing! You want to try and be concise to best convey your ideas to the reader.

3. **Code best practices still apply.** – The same practices you may use while programming applies when writing pseudocode! One very important example is to name your variables appropriately. Not everything needs to (or should be) a single letter!

4. **Pseudocode is *not* code.** – Pseudocode intends to capture *core logical steps* in an algorithm, *not* to be run on a machine. If you tried to write pseudocode as if you were programming, then it might end up too verbose/complicated than it needs to be!

5. **Pseudocode is *not* a justification.** – Don't simply present pseudocode and expect the reader to understand what's happening; use your words as well!