Pseudocode Guide

Pseudocode refers to an artifical and informal way to describe algorithms that you *may* want to use to describe the algorithms that you write in CSCI1570. Note that you are never *required* to use pseudocode (and in fact there are many times where not doing so will allow you to have much cleaner solutions), but if you do want to at any point, this guide should help you.

1 Pseudocode in LaTeX

Built into the `cs157.cls` theme file that we use for the course are the `algorithmicx` and `algpseudocode` packages, which are our recommendations for writing pseudocode in LaTeX.

1.1 The algorithm environment

One of the two major environments in the `algpseudocode` package is the `algorithm` environment. By wrapping this in a `\begin{...}` and `\end{...}` you'll create a box to place your algorithm in! To better align things on the page it might be necessary to add additional square brackets after the `\begin{...}` tag with a `H` to place the box in the pdf as the source. You can also title your algorithm blocks using the `\caption{...}`.

```
\begin{algorithm}[H]
    \caption{Your First Pseudocode!}
\end{algorithm}
```

Algorithm 1 Your First Pseudocode!

1.2 The algorithmic environment

Here's the meat and potatoes of the `algpseudocode` package and where you will write all of your pseudocode. The `algorithmic` environment also has an optional parameter which labels line numbers. Simply pass in any number n in square brackets to number every nth line. So for example to label lines 5, 10, 15, 20, etc. pass in [5]. Note that it is *not* necessary to wrap an `algorithmic` environment in an `algorithm` environment.

```
\begin{algorithmic}[1]
  \Require{An array of items $A$}
  \Ensure{The sorted array of items}
  \Procedure{bubbleSort}{$A$}
    \State $n \gets$ length of $A$
    \Repeat
      \State swapped $\gets$ false
      \For{$i \in [1, n - 1]$}
        \If{$A[i - 1] > A[i]$}
          \State\Call{swap}
            {$A[i - 1], A[i]$}
          \State swapped $\gets$ true
        \EndIf
      \EndFor
    \Until{!swapped}
  \EndProcedure
\end{algorithmic}
```

Input: An array of items A
Output: The sorted array of items
1: **procedure** bubbleSort(A)
2: $n \leftarrow$ length of A
3: **repeat**
4: swapped $\leftarrow$ false
5: **for** $i \in [1, n-1]$ **do**
6: **if** $A[i-1] > A[i]$ **then**
7: swap($A[i-1], A[i]$)
8: swapped $\leftarrow$ true
9: **until** !swapped

Note in the algorithm above for `bubbleSort`, we use the keywords `\For` and `\EndFor` to designate a for loop. With many of these commands, you **must** include both the starting and ending tags, so that the environment knows how to indent the code blocks properly. The same is true for `\Procedure`, `\If`, etc.

```
\begin{algorithm}[H]
  \caption{Binary Search}
    \begin{algorithmic}[1]
      \Require{Sorted int array, arr
               An int $n$}
      \Ensure{The index of $n$ in arr,
              or where it would be
              inserted to keep arr
              sorted}
      \Procedure{binarySearch}{arr, n}
      \State $l \gets 0$
      \State $r \gets$ arr.length
      \While{$l < r$}
        \State $m \gets (l + r)$ // 2
        \If{arr[$m$] $\leq n$}
          \State $l \gets m + 1$
        \ElsIf{arr[$m$] $= n$}
              \State $l \gets m + 1$
        \Else
          \State $r \gets m - 1$
        \EndIf
      \EndWhile
      \State \Return $r + 1$
    \EndProcedure
\end{algorithmic}
\end{algorithm}
```

Algorithm 2 Binary Search

Input: Sorted int array, arr An int n
Output: The index of n in arr, or where it would be inserted to keep arr sorted

```
1:  procedure binarySearch(arr, n)
2:      l ← 0
3:      r ← arr.length
4:      while l < r do
5:          m ← (l + r) // 2
6:          if arr[m] < n then
7:              l ← m + 1
8:          else if arr[m] = n then
9:              l ← m + 1
10:         else
11:             r ← m − 1
12:     return r + 1
```

Here we provide an implementation of binary search. Notice that we wrapped our pseudocode in the `algorithmic` environment followed by the `algorithm` environemnt to create a nice box around our algorithm. This also provides an example of our to create `while` loops in your pseudocode using the `\While` and `\EndWhile` commands and how to create `else if` blocks using `\ElsIf`.

The above examples demonstrate most of the features that you will likely want to use for your upcoming problem sets, but there is a lot more to the `algpseudocode` package! If you are interested in what else it can do, feel free to check out the comprehensive online documentation here!

2 Tips for Writing Pseudocode

Now you know how to write pseudocode in LaTeX! Yay!

But really, that's just half of the battle. The other half is the ability to write clean, concise pseudocode that allows you to effectively communicate an algorithm to your reader. Here are a few useful tips that the TA staff came up with:

1. **Indentation is your friend** - As your pseudocode gets longer/more complicated because of additional loops or control structures (such as `if` statements) you may find indenting your source code is an effective way of organizing LaTeXdocument. It will also help in identifying which lines of code in the source correspond to which lines in the final pdf.

2. **Less is more** - As with proofs, having longer solutions doesn't get you any extra points, and usually longer solutions. You want to be concise with your writing to best convey your ideas to the reader.

3. **Pseudocode is not code** - There is a reason that you don't use a real programming language to write pseudocode! Pseudocode intends to capture *core logical steps* in an algorithm, but is **not** meant to be executed directly on any machine.

   Often, if you try to write pseudocode as if it were an actual language, then your writeup may end up being too verbose and complicated with indices and declarations that don't really matter to the algorithm itself.

4. **You don't *need* pseudocode** - As we said in the beginning of the handout, we (pretty much) never explicitly require that you write pseudocode. There are cases where a block of pseudocode might elucidate your procedure, but there are certainly cases where a more textual description of your algorithm can be far more clear to the reader. There isn't a clear way to determine whether or not you should write pseudocode