inefficientSort

This is an example of a very inefficient sorting algorithm that only supports sorting numeric values in descending order.

This function is only intended to show how you can easily break up your code into smaller, more manageable tasks by using subfunctions. Doing so makes your code a little easier to read and a little easier to manage when making changes. For example, this algorithm would have normally been a nested for-loop, and while it still technically is, it makes the code much easier to follow by putting the nested for loop in its own function so the code is much cleaner.

Example

From the Command Window, run the following lines of code

x = [1:10, -20:3:-5, 12:2:36];

sortedX = inefficientSort(x);

```
function result = inefficientSort(x)
lenX = length(x);
result = zeros(1,lenX);

for n=1:lenX
    % Get the position of the maximum value
    maxValPos = getMaxValPos(x);

    % Update the value at the current index and clear the value
    result(n) = x(maxValPos);
    x(maxValPos) = [];
end

end
```

This is a sub function, it is only accessible from within this function fiile. This is also a good example of variable scope, in both the above and this function, I am using the value n in a for loop, because these are different functions, this is allowed and won't cause any collision.

```
function pos = getMaxValPos(x)
    % Start with maxVal being the lowest value possible and position being 0
    maxVal = -1*realmax;
    pos = 0;

    % Loop through and find the maximum value of vector x
    for n = 1:length(x)
        % If the value at the current index is greater than the current max
        if x(n) > maxVal
```

```
            % Update maxVal with the new larger number and record the pos
            maxVal = x(n);
            pos = n;
        end
    end
end
```