VG101: Introduction to Computer and Programming

Week6 Checklist

Recursion

Definition

In mathematics and computer science, a class of objects or methods exhibit recursive behavior when they can be defined by two properties:

- A simple base case (or cases)—a terminating scenario that does not use recursion to produce an answer
- A set of rules that reduce all other cases toward the base case

Reference: https://en.wikipedia.org/wiki/Recursion

In recusion, we only solve a small part of the problem in each step, and leave other parts to be solved by recursion.

- Mathematical Induction
- Movie *Inception*
- Any algorithm written in recursion can be re-written in iteration. However, using iteration sometimes can be extremely difficult

Example

- Calculate factorial

```
function value = factorial(n)
if n <= 1
    value = 1;
else
    value = n * factorial(n-1);
end
end
```

- Judge whether a string is palindrome

```
function is_or_not = isPalindrome(input_string)
% input should be a string
% output is a logical value whether the string is a palindrome
len = length(input_string);
if len <= 1
    is_or_not = 1;
elseif input_string(1) ~= input_string(len)
    is_or_not = 0;
else
    is_or_not = isPalindrome(input_string(2: len - 1));
end
end
```

- Quick Sort
    - Pick an element, called a *pivot*, from the array.
    - Partitioning: reorder the array so that all elements with values less than the pivot come before the pivot, while all elements with values greater than the pivot come after it (equal values can go either way). After this partitioning, the pivot is in its final position. This is called the partition operation.
    - Recursively apply the above steps to the sub-array of elements with smaller values and separately to the sub-array of elements with greater values.

```
function [sorted_array] = qsort(unsorted_array)
len = length(unsorted_array);
if len <= 1
    sorted_array = unsorted_array;
    return;
end
pivot = unsorted_array(1);
sorted_array = zeros([1, len]);
left = 1;
right = len;
for i = 2: len
    if unsorted_array(i) <= pivot
        sorted_array(left) = unsorted_array(i);
        left = left + 1;
    else
        sorted_array(right) = unsorted_array(i);
        right = right - 1;
    end
end
% now actually left == right
sorted_array = [qsort(sorted_array(1: left - 1)), pivot,
qsort(sorted_array(right + 1: len))];
end
```