Exercises Problem Set 3

Question 1. Explain why measuring the running time empirically is not a good method for comparing the complexity of different algorithms. Explain how the methodology of counting operations overcomes these obstacles.

Recall that a function $f(n)$ is said to be in $O(g(n))$ if $f(n)$ is bounded, or dominated, by some constant multiple of $g(n)$ for large values of n. In other words, we can find a number c_0 such that for sufficiently large input, n, the value of $f(n)$ is always less than $c_0 \cdot g(n)$.

Expressed mathematically,[1]

$$f(n) \in O(g(n))$$

if there exists some constants c_0 and n_0 such that for all $n \geq n_0$ the following inequality holds:

$$f(n) \leq c_0 \cdot g(n)$$

Question 2. Determine whether the following statements[2] are true of false:

1) $3n \in O(n)$
2) $18n^3 + 4n^2 + 2 \in O(n^3)$
3) $n! \in O(n^n)$
4) $n! \in O(2^n)$
5) $6^{3n} \in O(6^n)$

Question 3. Given that $f(n) \in O(g(n))$.

a) Can we say anything about f and g with regards to Ω?

b) Under what additional condition is it is also true that $f(n) \in \Theta(g(n))$?

Question 4. Below is a function written in Python that is supposed to find the maximal element given a list of integers.

```
def sloppy_list_max(lst):
    "Takes any list of integers and returns the maximal element"
    maxElem = lst[0]
        for i in range(len(lst)+1):
            if maxElem > lst[i]:
                maxElem = lst[i]
    return maxElem
```

a. The code contains three bugs, examine the code carefully and debug the code. Then rewrite a correct `list_max(lst)` function.

b. What is the basic operation of the code?

c. What is the cost, $T(n)$, of this algorithm for an input of size n? In your analysis, use constants for every operation.

Question 5. The following Python code is an attempt to define an algorithm which answers whether an item exists in a given 2D array. However, in its current state the code is not functional.

[1] Note that the $\in$, pronounced "in", is just a shorthand symbol for membership of a set. E.g. Einstein $\in$ Humans is read "Einstein is a member of the set of Humans".

[2] The exclamation mark denotes the factorial function, e.g. $5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$. In general: $m! = m \cdot (m-1) \cdot (m-2) \cdot ... \cdot 1$ for a nonnegative integer m. By definition $(0! = 1)$.

```
def careless_matrix_search(matrix, elem):
    """
    Accepts a 2D array (matrix) and an element (elem),
    return True if element is found and False otherwise.
    """
    for x in range(len(matrix[1])):
        for y in range(len(matrix[2])):
            if matrix[y][x] == elem:
                return True
            else:
                return False
    return False
```

a. Examine the code carefully and find the two or three bugs. Rewrite a correct implementation of `matrix_search(matrix, ele`

b. What is the worst-case scenario for the algorithm `matrix_search(matrix, elem)`, in terms of time-complexity? That is, the maximum number of operations with regards to the dimensions ($n \times m$) of the 2D array.

A *linked list* is a data structure where each node contains two pieces of information: The value of the node, a reference to the next node (if any) in the list. Similar to a tree, a linked list is accessed through the first element, called **head**.

Question 6.
In this exercise you are tasked to analyze a search algorithm. We search for a value, x, if the value is contained within the data structure return `True`, otherwise return `False`.

a) Design a pseudocode algorithm for a list implemented as a Linked list, (let `LL.head` denote the head of the linked list) then determine the worst-case time complexity of your algorithm.

b) Last week you did this for a Binary Search Trees, what is the worst-case time complexity of your algorithm?

c) What is the worst-case time complexity for the search algorithm over a Perfect Binary Search Tree[3]?

Question 7. Solve this week's programming work sheet.

[3]Recall from last week that a binary tree is called *perfect* if all nodes have two children and all leaves have the same height