cs 5800 - hw1

Yijing Xiao

September 22, 2021

1 Problem 1

1.1 1(a)

Pseudocode 1 Find the first common element

```
function TraversalList(node)
    numberCount ← 0
    tempPtr ← node
    while tempPtr ≠ None do
        numberCount ← numberCount + 1
        tempPtr ← tempPtr.next
    end while
    return numberCount
end function

function GetIntersection(head1, head2)
    num1 ← TraversalList(head1)
    num2 ← TraversalList(head2)
    if num1 > num2 then
        offset ← num1 − num2
        for i = 0 → offset − 1 do
            head1 ← head1.next
        end for
        while (head1 ≠ None)and(head2 ≠ None) do
            if head1 = head2 then
                return head1.data
            end if
        end while
    else
        offset ← num2 − num1
        for i = 0 → offset − 1 do
            head2 ← head2.next
        end for
        while (head1 ≠ None)and(head2 ≠ None) do
            if head1 = head2 then
                return head1.data
            end if
        end while
    end if
    return −1
end function
```

1.2 1(b)

```
# link list node
class Node:
    def __init__(self, data):
        self.data = data # assign data
        self.next = None # initialize next as NULL
        self.head = None


class LinkedList:
    def __init__(self):
        self.head = None

    def append(self, newNode):
        if self.head is Node:
            self.head = newNode
            return

        last = self.head

        while(last.next):
            last = last.next
        last.next = newNode

    def printList(self):
        temp = self.head

        while(temp):
            print(temp.data)
            temp = temp.next


def TraversalList(node):# how many nodes are in the list
    num = 0
    ptr = node

    while(ptr):
        num += 1
        ptr = ptr.next
    return num


def getIntersection(head1, head2):

    numA = TraversalList(head1)
    numB = TraversalList(head2)

    if numA > numB:
        offset = numA - numB
        return helper(offset, head1, head2)

    else:
        offset = numB - numA
        return helper(offset, head2, head1)

```

```python
def helper(num, head1, head2):
    ptr1, ptr2 = head1, head2
    for i in range(num):
        if(ptr1 == None):
            return -1
        ptr1 = ptr1.next

    while(ptr1 != None and ptr2 != None):
        if ptr1 is ptr2: # check address is same
            return ptr1.data
        ptr1 = ptr1.next
        ptr2 = ptr2.next

    return -1

if __name__ == '__main__':
    intersection = Node(7) # define intersection

    # frist: 1->3->5->7->9
    listA = LinkedList()
    listA.head = Node(1)
    listA.append(Node(3))
    listA.append(Node(5))
    listA.append(intersection)
    listA.append(Node(9))
    #listA.printList()

    # second: 2->7->12
    listB = LinkedList()
    listB.head = Node(2)
    listB.append(intersection)
    listB.append(Node(12))
    #listB.printList()

    print(getIntersection(listA.head, listB.head))
    # the result should be 7
```

2 Problem 2

According to the definition of θ - notation:

$$0 \leq C_1 * (f(n) + g(n)) \leq max(f(n), g(n)) \leq c_2 * (f(n) + g(n))$$

Show that there exist positive constants c_1, c_2 and n_0 for all $n \leq n_0$
Since $max(f(n), g(n)) \geq f(n)$ and $max(f(n), g(n) \geq g(n)$:

$$f(n) + g(n) \leq 2max(f(n), g(n))$$

$$\frac{1}{2}(f(n) + g(n)) \leq max(f(n), g(n))$$

Since $f(n)$ and $g(n)$ are asymptotically non-negative functions: it means that $f(n) \geq 0, g(n) \geq 0$

$$f(n) + g(n) \geq max(f(n), g(n))$$

Therefore, we can combine the above two inequalities as follow:

$$0 \leq \frac{1}{2}(f(n) + g(n)) \leq max(f(n), g(n)) \leq (f(n) + g(n)) \; for \; n \geq n_0$$

So, $max(f(n), g(n)) = \Theta(f(n) + g(n))$ **because there exists** $c_1 = \frac{1}{2}, c_2 = 1$

3 Problem 3

3.1 Is $2^n + 1 = O(2^n)$?

According to the definition of O-notation:

$$0 \leq f(n) \leq cg(n) \; for \; all \; n \geq n_0$$

show that there exist positive constant c **and** n_0 **for all** $n \geq n_0$

$$2^n + 1 = 2 * 2^n$$

So for any $c \geq 2$**, there exists** $0 \leq 2^n + 1 \leq c * 2^n \; for \; n \geq 1$
Therefore, $\mathbf{2}^n + 1 = O(2^n)$

3.2 Is $2^2n = O(2^n)$?

According to the definition of O-notation:

$$0 \leq f(n) \leq cg(n) \; for \; all \; n \geq n_0$$

show that there exist positive constant c **and** n_0 **for all** $n \geq n_0$

$$2^2n = 2^n * 2^n$$

So if $c = O(2^n)$**, we will need a constant** c **such that** $0 \leq 2^n * 2^n$ **which is** $c = 2^n$**. However,** 2^n **is not a constant. Therefore,** $2^2n \neq O(2^n)$

4 Problem 4

$n^{0.001}$ $\quad \ln \ln n$ $\quad 2^{2 \ln n}$ $\quad 2^{\ln^2 n}$ $\quad \sqrt{n} \ln n$ $\quad n!$ $\quad (\ln n)!$

5 Problem 5

5.1 a. $T(n) = 2T(\frac{n}{2}) + n^4$

According to the simple master theorem: $a = 2, b = 2, c = 4:$
$\log_b a = 1 \longrightarrow c > \log_b a$ **so case 3:** $T(n) = \Theta(n^4)$

5.2 b. $T(n) = T(\frac{7n}{10}) + n$

According to the simple master theorem: $a = 1, b = \frac{10}{7}, c = 1:$
$\log_b a = 0 \longrightarrow c > \log_b a$ **so case 3:** $T(n) = \Theta(n)$

5.3 c. $T(n) = 16T(\frac{n}{4}) + n^2$

According to the simple master theorem: $a = 16, b = 4, c = 2:$
$\log_b a = 2 \longrightarrow c = \log_b a$ **so case 2:** $T(n) = \Theta(n^2 \log n)$

5.4 d. $T(n) = 7T(\frac{n}{3}) + n^2$

According to the simple master theorem: $a = 7, b = 3, c = 2:$
$\log_b a \approx 1.77 \longrightarrow c > \log_b a$ **so case 3:** $T(n) = \Theta(n^2)$

5.5 e. $T(n) = 7T(\frac{n}{2}) + n^2$

According to the simple master theorem: $a = 7, b = 2, c = 2:$
$\log_b a \approx 2.8 \longrightarrow c < \log_b a$ **so case 1:** $T(n) = \Theta(n^{\log_2 7}$ **)**

5.6 f. $T(n) = 2T(\frac{n}{4}) + \sqrt{n}$

According to the simple master theorem: $a = 2, b = 4, c = \frac{1}{2}:$
$\log_b a = \frac{1}{2} \longrightarrow c = \log_b a$ **so case 2:** $T(n) = \Theta(n^{\frac{1}{2}} \log n)$

5.7 g. $T(n-2) + n^2$

The subtraction occurring inside to T won't change the asymptotics of the solution. According to the simple master theorem, a = 1, b = 1, c=2, $\log_b a = 1, c > \log_b a$**, so case 3**
$T(n) = \Theta(n^2)$

6 Problem 6

6.1 a. $T(n) = 4T(\frac{n}{3}) + n \lg n$

According to the master theorem: $a = 4, b = 3, f(n) = n \lg n$
$n^{\log_b a} = n^{\log_3 4} \approx n^{1.26}$
So case 1, $T(n) = \Theta(n^{\log_3 4}$**)**

6.2 b. $T(n) = 3T(\frac{n}{3}) + n/\lg n$

$T(n) = 3(3T(\frac{n}{3^2}) + \frac{\frac{n}{3}}{\lg \frac{n}{3}}) + \frac{n}{\lg n}$
$= 3^k T(\frac{n}{3^k}) + \frac{n}{\lg \frac{n}{3^{k-1}}} + ... + \frac{n}{\lg \frac{n}{3}} + \frac{n}{\lg n}$
Assume $n = 3^k, k = \log_3 n$
$T(n) = 3^k T(1) + \frac{n}{\lg \frac{n}{3^{k-1}}} + ... + \frac{n}{\lg \frac{n}{3}} + \frac{n}{\lg n}$
$= 3^k T(1) + n * \sum_{i=0}^{k-1} \frac{1}{\lg \frac{n}{3^i}}$
$= 3^k T(1) + n * \sum_{i=0}^{k-1} \frac{1}{\lg n - i * \lg 3}$

6.3 c. $T(n) = 4T(\frac{n}{2}) + n^2\sqrt{n}$

$T(n) = 4T(\frac{n}{2}) + n^2 * n^{\frac{1}{2} = 4T(\frac{n}{2}) + n^{\frac{5}{2}}}$
According to the simple master theorem: $a = 4, b = 2, c = \frac{5}{2} = 2.5$
$\log_b a = log_2 4 = 2, c > log_b a$ **So case 3,** $T(n) = \Theta(n^{\frac{5}{2}}$**)**

6.4 d. $T(n) = 3T(\frac{n}{3} - 2) + \frac{n}{2}$

The subtraction occurring inside to T won't change the asymptotic of the solution
According to the simple master theorem: $a = 3, b = 3, c = 1$
$\log_b a = \log_3 3 = 1, c > \log_b a$
So case 2, $T(n) = \Theta(n \log n)$

6.5 e. $T(n) = 2T(\frac{n}{2}) + \frac{n}{\lg n}$

$T(n) = 2(2T(\frac{n}{4}) + \frac{\frac{n}{2}}{\lg \frac{n}{2}}) + \frac{n}{\lg n}$
$= 2^k T(\frac{n}{2^k}) + \frac{n}{\lg \frac{n}{2^{k-1}}} + ... + \frac{n}{\lg \frac{n}{2}} + \frac{n}{\lg n}$
Assume $n = 2^k, k = \lg n$
$T(n) = 2^k T(1) + \frac{n}{\lg \frac{n}{2^{k-1}}} + ... + \frac{n}{\lg \frac{n}{2}} + \frac{n}{\lg n}$
$= 2^k T(1) + n * \sum_{i=0}^{k-1} \frac{1}{\lg \frac{n}{2^i}}$
$= 2^k T(1) + n * \sum_{i=0}^{k-1} \frac{1}{\lg n - \lg 2^i}$
$= 2^k T(1) + n * \sum_{i=0}^{k-1} \frac{1}{\lg n - i}$**, since** $k = \lg n$
$= 2^k T(1) + n * \sum_{i=0}^{k-1} \frac{1}{k-i}$
Since $\sum_{i=0}^{k-1} \frac{1}{k-i} = \frac{1}{k} + \frac{1}{k-1} + ... + \frac{1}{2} + 1$ **is harmonic series, therefore it can be approximated as** $\lg k$
Since $k = \lg n$**, the result** $T(n) = n * \lg \lg n$

6.6 f. $T(n) = T(\frac{n}{2}) + T(\frac{n}{4}) + T(\frac{n}{8}) + n$

Assume that $T(n) \leq cn$ **where c is a positive constant**
$T(n) = T(\frac{n}{2}) + T(\frac{n}{4}) + T(\frac{n}{8}) + n$
$\leq c * (\frac{n}{2}) + c * (\frac{n}{4}) + c * (\frac{n}{8}) + n$
$= (\frac{7}{8} * c + 1) * n$
$\leq cn (c \geq 8)$