CS5800: Algorithms — Virgil Pavlu

Homework 1
Due : September 22

Name: Deenadayalan Dasarathan
Collaborator:

Instructions:

- Make sure to put your name on the first page. If you are using the LaTeX template we provided, then you can make sure it appears by filling in the `yourname` command.
- Please review the grading policy outlined in the course information page.
- You must also write down with whom you worked on the assignment. If this changes from problem to problem, then you should write down this information separately with each problem.
- Problem numbers (like Exercise 3.1-1) are corresponding to CLRS 3^{rd} edition. While the 2^{nd} edition has similar problems with similar numbers, the actual exercises and their solutions are different, so make sure you are using the 3^{rd} edition.

1. ***(20 points)***

Two linked lists (simple link, not double link) heads are given:headA, and headB;it is also given that the two lists intersect, thus after the intersection they have the same elements to the end. Find the first common element, without modifying the lists elements or using additional data structures.

(a) A linear algorithm is discussed in the lecture: count the lists first, then use the count difference as an offset in the longer list, before traversing the lists together. Write a formal pseudocode (the pseudocode in the lecture is vague), using "next" as a method/pointer to advance to the next element in a list.

Solution: Linear Algorithm

```
Step 1: Initialize listA and listB
Step 2: Assign currentA = headA ; currentB = headB
Step 2: Assign lengths lenA = 0 ; lenB = 0
Step 3: Calculate length of listA
    while( currentA.next != NULL)
        lenA += 1
    endwhile
Step 4: Calculate length of listB
    while( currentB.next != NULL)
        lenB += 1
    endwhile
Step 5: Calculate offset
    offset = absolute(lenA - lenB)
Step 6: Offsetting the bigger list
    if (lenA > lenB)
        for (int i=0; i<=offset; i++ )
            currentA = currentA.next
        endfor
    else if (lenB > lenA)
        for (int i=0; i<=offset; i++ )
            currentB = currentB.next
        endfor
    endif
Step 7:Iterating listA and ListB to find the first common element
    while(currentA != None)
        if (currentA == currentB)
            return currentA
        endif
        currentA = currentA.next
        currentB = currentB.next
    endwhile
```

Result: The pseudocode returns the first common element at the intersection of two lists.

(b) Write the actual code in a programming language (C/C++, Java, Python etc) of your choice and run it on a made-up test pair of two lists. A good idea is to use pointers to represent the list linkage.

Solution:

```
class Node:
""" A class for a node in a singly-linked list, storing a data payload and links to next node.
"""
    def __init__(self, data = None, next = None):
    """Initialize the node with data payload and link to next node."""
        self.data = data
        self.next = next


    def getdata(self):
    """Get the node's data payload."""
        return self.data


    def setdata(self, data = None):
    """Set the node's data payload."""
        self.data = data


    def getnext(self):
    """Get the next linked node."""
        return self.next


    def setnext(self, node = None):
    """Set the next linked node."""
        self.next = node
## End of class Node


class LinkedList:
"""A singly linked list."""
    def __init __(self, data=None, head=None):
        self.data = data
        self.head = head


    def __iter __(self):
    """Returns a forward iterator over the list."""
        node = self.head
        while node is not None:
            yield node.getdata()
            node = node.getnext()
```

```
    def __str__(self):
    """Returns a string representation of the list."""
        return "→".join([str(x) for x in self])
    def __repr__(self):
    """Returns a printable representation of the list."""
        return str(self)
    def __len__(self):
    """Returns the length of the list."""
        size = 0
        for i in self:
            size += 1
        return size
    def push(self, data):
    """Adds a new item to the end of the list.
    param data: The new item to append to the list.
    returns: None """
        if self.head is None:
            node = Node()
            node.setdata(data)
            node.setnext(None)
            self.head = node
        else:
            node = Node()
            node.setdata(data)
            itr = self.head
            while itr.next:
                itr = itr.next
            itr.next = node
## End of class LinkedList

def mergeLists(baseList, tailList):
    itr = baseList.head
    while itr.next:
        itr = itr.next
    itr.next = tailList.head
    return baseList

def intersection(listA, listB):
    lenA = len(listA)
    lenB = len(listB)
    offset = abs(lenA-lenB)
    print("Offset: ",offset)
    currA = listA.head
    currB = listB.head
```

```
        ## length of listA greater then listB; then offset the listA
        if lenA > lenB:
            for i in range(offset):
                currA = currA.next
        ## length of listB greater then listA; then offset the listB
        elif lenB > lenA:
            for i in range(offset):
                currB = currB.next
        while currA != None:
            if currA == currB:
                return currA
            else:
                currA = currA.next
                currB = currB.next

if __name__== "__main __":
    mergeList = LinkedList()
    mergeList.push(130)
    mergeList.push(180)
    mergeList.push(190)

    listA = LinkedList()
    listA.push(10)
    listA.push(11)
    listA.push(13)
    listA.push(12)
    listA.push(15)
    listA.push(16)
    listA.push(18)
    listA=mergeLists(listA,mergeList)
    listB = LinkedList()
    listB.push(14)
    listB.push(17)
    listB.push(19)
    listB.push(20)
    listB=mergeLists(listB,mergeList)
    ## two lists created listA and listB
    print("List A: ",listA)
    print("List A length",len(listA))
    print("List B: ",listB)
    print("List B length",len(listB))
    common = intersection(listA, listB)
    print("First Common element: ",common.data)
```

2. ***(10 points)*** *Exercise 3.1-1*

Let f(n) and g(n) be asymptotically non-negative functions. Using the basic definition of Θ-notation, prove that max(f(n),g(n)) = Θ(f(n) + g(n)).

Solution:

By Θ definition,

$0 \leq C_1$G(n) ≤ F(n) ≤ C_2G(n)

To prove:

$0 \leq C_1$ (f(n) + g(n)) ≤ max(f(n),g(n)) ≤ C_2(f(n) + g(n))

Proof:

Upper bound

f(n) + g(n) will always be greater than max(f(n),g(n))

Let $C_2 = 1$

max(f(n),g(n)) ≤ (f(n) + g(n))

Upper bound condition satisfies.

Lower bound

We know that,

f(n) ≤ max(f(n),g(n)) – ①

g(n) ≤ max(f(n),g(n)) – ②

Adding equations ① and ②

f(n) + g(n) ≤ 2max(f(n),g(n))

½(f(n) + g(n)) ≤ max(f(n),g(n))

Therefore C_1 = ½

Lower bound condition satisfies.

Hence, max(f(n),g(n)) = Θ(f(n) + g(n))

3. ***(5 points)*** *Exercise 3.1-4*

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

Solution:

f(n) = 2^{n+1}

g(n) = 2^n

By O definition,

0 ≤ f(n) ≤ Cg(n)

Substituting f(n) and g(n),

$2^{n+1} \leq C2^n$

$2^n \cdot 2 \leq C2^n$

$2 \leq C$

Hence the condition satisfies and $2^{n+1} = O(2^n)$ for constant $C \geq 2$

Is $2^{2n} = O(2^n)$?

Solution:

f(n) = 2^{2n}
g(n) = 2^n
By O definition,
0 ≤ f(n) ≤ Cg(n)
Substituting f(n) and g(n),
$2^{2n} \leq C2^n$
$2^n \,.\, 2^n \leq C2^n$
$2^n \leq C$
C cannot be ≥ 2^n for very large values of 'n'
Hence the condition fails and $2^{2n} \neq O(2^n)$

4. ***(15 points)***

Rank the following functions in terms of asymptotic growth. In other words, find an arrangement of the functions $f_1, f_2, ...$ such that for all i, $f_i = \Omega(f_{i+1})$.

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

Solution:

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

5. ***(40 points)*** *Problem 4-1 (page 107)*

Give asymptotic upper and lower bounds for T(n) in each of the following recurrences. Assume that T(n) is constant for $n \leq 2$. Make your bounds as tight as possible, and justify your answers.

Master Theorem:
If $T(n) = aT(n/b) + n^c$ and constants $a \geq 1$; $b > 1$; $c \geq 0$
then,
Case(1) : $T(n) = \Theta(n^{log_b a})$; if $c < log_b a$
Case(2) : $T(n) = \Theta(n^c logn)$; if $c = log_b a$
Case(3) : $T(n) = \Theta(n^c)$; if $c > log_b a$

(a) $T(n) = 2T(n/2) + n^4$

Solution:

Applying Simplified Master Theorem,
$a = 2$; $b = 2$; $c = 4$
$log_2 2 = 1$
$c > log_b a$
So, by case(3), $T(n) = \Theta(n^c)$
Therefore, $T(n) = \Theta(n^4)$

Iteration method:
$T(n/2) = 2T(n/4) + (n/2)^4$
$T(n/4) = 2T(n/8) + (n/4)^4$
$T(n) = 8T(n/8) + 4(n/4)^4 + 2(n/2)^4 + n^4$
$T(n) = 2^3T(n/8) + 2^2(n/4)^4 + 2(n/2)^4 + n^4$
$T(n) = 2^kT(n/2^k) + 2^{(k-1)}(n/2^{(k-1)})^4 + 2^{(k-2)}(n/2^{(k-2)})^4 + .. + n^4$
$T(n) = 2^kT(n/2^k) + n^4((1/2^{(k-1)})^3 + (1/2^{(k-2)})^3 + .. + 1)$
$T(n) = 2^kT(n/2^k) + n^4/2^{3k}((1/2^{(-3)})^1 + (1/2^{(-3)})^2 + .. + (1/2^{(-3)})^k)$
General Form
$T(n) = 2^kT(n/2^k) + \frac{n^4}{2^{3k}}\frac{2^3((2^3)^k-1)}{(2^3-1)}$
Let $n = 2^k$
$k = logn$
After substituting value of n $T(2^k/2^k) = 1$
Substituting value of k and $T(2^k/2^k)$ in general form
$T(n) = 2^{logn} \times (1) + \frac{n^4}{2^{3logn}}\frac{2^3((2^3)^{logn}-1)}{(2^3-1)}$
$T(n) = n \times (1) + \frac{n^4}{n^3}\frac{2^3((n^3-1)}{(2^3-1)}$
$T(n) = n + 2^3n\frac{((n^3-1)}{(2^3-1)}$
$T(n) = \frac{7n+8n^4-8n}{7}$
$T(n) = \frac{8n^4-n}{7}$
Therefore $T(n) = \Theta(n^4)$
By Θ definition
$C_1g(n) \le f(n) \le C_2g(n)$
$f(n) = \frac{8n^4-n}{7}$; $g(n) = n^4$
$C_1n^4 \le \frac{8n^4-n}{7} \le C_2n^4$
$C_1n^4 \le \frac{8n^4-n}{7}$
$C_17n^4 \le 8n^4 - n$
$C_1 = 1$

$C_2n^4 \ge \frac{8n^4-n}{7}$
$C_27n^4 \ge 8n^4 - n$
$C_27n^4 \ge 8n^4 - n$
$C_2 = 1.5$
$C_1 = 1$ and $C_2 = 1.5$ satisfy boundary conditions,
$1 \times n^4 \le \frac{8n^4-n}{7} \le 1.5 \times n^4$

(b) $T(n) = T(7n/10) + n$

Solution:

Applying Simplified Master Theorem,
$a = 1$; $b = 10/7$; $c = 1$
$log_{10/7} 1 = 0$
$c > log_b a$
So, by case(3), $T(n) = \Theta(n^c)$
Therefore, $T(n) = \Theta(n)$

Iteration method:
Let $m = 10/7$
$T(n) = T(n/m) + n$
$T(n/m) = T(n/m^2) + \frac{n}{m}$
$T(n/m^2) = T(n/m^3) + \frac{n}{m^2}$
Therefore,
$T(n) = T(n/m^3) + \frac{n}{m^2} + \frac{n}{m^1} + \frac{n}{m^0}$
General form,
$T(n) = T(n/m^k) + \frac{n}{m^(k-1)} + \frac{n}{m^(k-2)} + .. + \frac{n}{m^(k-k)}$
$T(n) = T(n/m^k) + n(\frac{1}{m^0} + \frac{1}{m^1} + .. + \frac{1}{m^(k-2)} + \frac{1}{m^(k-1)})$
$T(n) = T(n/m^k) + n(1 + (m^{-1})^1 + .. + (m^{-1})^{(k-2)} + (m^{-1})^{(k-1)})$
$T(n) = T(n/m^k) + n(\frac{1-(m^{-1})^{k-1}}{1-\frac{1}{m}})$
$T(n) = T(n/m^k) + \frac{n \times m}{m-1}(1 - (m^{-1})^{k-1})$
Let $n = m^k$; $logn = klogm$
$m = 10/7$; $k = 2logn$
$T(n) = T(n/m^k) + \frac{n \times m}{m-1}(1 - (m^{-1})^{k-1})$
$T(m^k/m^k) = 1$
By substituting all the values, we get,
$T(n) = 1 + 3.5 \times n$
Therefore $T(n) = \Theta(n)$
By Θ definition
$C_1 g(n) \le f(n) \le C_2 g(n)$
$C_1 \times n \le 1 + 3.5 \times n \le C_2 \times n$
$C_1 = 3$ and $C_2 = 4$ satisfy the boundary conditions
$3 \times n \le 1 + 3.5 \times n \le 4 \times n$

(c) $T(n) = 16T(n/4) + n^2$
Solution:

Applying Simplified Master Theorem,
$a = 16$; $b = 4$; $c = 2$
$log_4 16 = 2$
$c = log_b a$
So, by case(2), $T(n) = \Theta(n^c logn)$
Therefore, $T(n) = \Theta(n^2 logn)$

Iteration method:
$T(n) = 16T(n/4) + n^2$
$T(n/4) = 16T(n/16) + (n/4)^2$
$T(n/16) = 16T(n/64) + (n/16)^2$
$T(n/64) = 16T(n/256) + (n/64)^2$

$T(n) = 16^4 T(n/4^4) + n^2(\frac{16^3}{64^2} + \frac{16^2}{16^2} + \frac{16^1}{4^2} + 1)$
$T(n) = 16^4 T(n/4^4) + n^2(1 + 1 + 1 + 1)$

General form,
$T(n) = 16^k T(n/4^k) + n^2 \sum_{i=0}^{k-1}(1 + 1 + 1 + ... + 1)$
$T(n) = 16^k T(n/4^k) + n^2(k - 1)$

Let $n = 4^k$; $2k = logn$; $k = \frac{1}{2} logn$
$T(n) = 16^{\frac{1}{2} logn}(C) + n^2(\frac{1}{2} logn - 1)$
$T(n) = Cn^2 + n^2(\frac{1}{2} logn - 1)$
$T(n) = Cn^2 + \frac{1}{2} n^2 logn - n^2$

$T(n) = \frac{1}{2} n^2 logn + n^2(C - 1)$
let $C - 1 = 0$
$T(n) = \frac{1}{2} n^2 logn$
Therefore ,
$T(n) = \Theta(n^2 logn)$

By Θ definition
$C_1 g(n) \leq f(n) \leq C_2 g(n)$
$C_1 = \frac{1}{4}$; $C_2 = \frac{3}{4}$ satisfy the boundary conditions,
$\frac{1}{4} n^2 logn \leq \frac{1}{2} n^2 logn \leq \frac{3}{4} n^2 logn$

(d) $T(n) = 7T(n/3) + n^2$
Solution:

Applying Simplified Master Theorem,
$a = 7$; $b = 3$; $c = 2$
$log_3 7 = 1.77$
$c > log_b a$
So, by case(3), $T(n) = \Theta(n^c)$
Therefore, $T(n) = \Theta(n^2)$

(e) $T(n) = 7T(n/2) + n^2$
Solution:

Applying Simplified Master Theorem,
$a = 7$; $b = 2$; $c = 2$
$log_2 7 = 2.8$
$c < log_b a$
So, by case(1), $T(n) = \Theta(n^{log_b a})$
Therefore, $T(n) = \Theta(n^{log_2 7})$

(f) $T(n) = 2T(n/4) + \sqrt{n}$
Solution:

Applying Simplified Master Theorem,
$a = 2$; $b = 4$; $c = \frac{1}{2}$
$log_4 2 = 0.5$
$c = log_b a$
So, by case(2), $T(n) = \Theta(n^c logn)$
Therefore, $T(n) = \Theta(\sqrt{n} logn)$

(g) $T(n) = T(n-2) + n^2$
Solution:

Iteration method:
$T(n) = T(n-2) + n^2$
$T(n-2) = T(n-4) + (n-2)^2$
$T(n-4) = T(n-6) + (n-4)^2$
$T(n-6) = T(n-8) + (n-6)^2$

$T(n) = T(n-8) + (n-6)^2 + (n-4)^2 + (n-2)^2 + n^2$
General Form,
$T(n) = T(n-2k) + \sum_{i=0}^{k-1}(n-2i)^2$
Let $n - 2k = 0$; $k = \frac{n}{2}$
$T(n) = T(0) + \sum_{i=0}^{\frac{n}{2}-1}(n-2i)^2$
$T(n) = T(0) + \sum_{i=0}^{\frac{n}{2}-1}(n^2 - 4ni + 4i^2)$

$T(n) = T(0) + n^2\sum_{i=0}^{\frac{n}{2}-1} 1 - 4n\sum_{i=0}^{\frac{n}{2}-1} i + 4\sum_{i=0}^{\frac{n}{2}-1} i^2$

$T(n) = 1 + n^2(\frac{n}{2} - 1) - 4n.\frac{n}{2}.\frac{1}{2}(\frac{n}{2} - 1) + 4.\frac{1}{6}(\frac{n^3}{4} - \frac{3n^2}{4} + \frac{n}{2})$
$T(n) = \frac{1}{6}(n^3 - 3n^2 + 3n + C)$
Therefore,
$T(n) = \Theta(n^3)$
By Θ definition
$C_1 g(n) \leq f(n) \leq C_2 g(n)$
$C_1 = \frac{1}{7}$; $C_2 = \frac{1}{3}$
$\frac{1}{7}(n^3) \leq f(n) \leq \frac{1}{3}(n^3)$

6. ***(30 points)*** *Problem 4-3 from (a) to (f) (page 108)*

Give asymptotic upper and lower bounds for T(n) in each of the following recurrences. Assume that T(n) is constant for sufficiently small n. Make your bounds as tight as possible, and justify your answers.

(a) $T(n) = 4T(n/3) + n \lg n$
Solution:

$a = 4$; $b = 3$; $f(n) = n \log n$
Compare $n^{\log_b a}$ with $f(n)$
$n^{\log_b a} = n^{\log_3 4} = n^{1.26}$
$\lim_{n\to\infty} \frac{f(n)}{n^{1.26}}$
$\lim_{n\to\infty} \frac{n \log n}{n^{1.26}}$
$\lim_{n\to\infty} \frac{\log n}{n^{0.26}}$
$\lim_{n\to\infty} \frac{\log n}{n^x}$
Applying L'Hopital's rule,
$\lim_{n\to\infty} \frac{1}{xn^x} = 0$
Therefore A = 0; $f(n) = O(n^{\log_3 4})$
By Master theorem case(1),
$T(n) = \Theta(n^{\log_3 4})$

(b) $T(n) = 3T(n/3) + n/\lg n$
Solution:

Iteration method,
$T(n) = 3T(n/3) + n/\log n$
$T(n/3) = 3T(n/9) + n/3\log(n/3)$
$T(n/9) = 3T(n/27) + n/9\log(n/9)$
$T(n) = 3^3T(n/3^3) + n/\log(n/3^2) + n/\log(n/3^1) + n/\log(n/3^0)$
General form,

$T(n) = 3^kT(n/3^k) + n/\log(n/3^{k-1}) + n/\log(n/3^{k-2}) + n/\log(n/3^{k-k})$
$T(n) = 3^kT(n/3^k) + n\sum_{i=0}^{k-1} /\log(n/3^i)$
$T(n) = 3^kT(n/3^k) + n\sum_{i=0}^{k-1} 1/(\log n - i\log 3)$

Let $n = 3^k$; $k = \log_3 n$

$$T(n) = 3^{\log_3 n} T(1) + n/\log 3 \sum_{i=0}^{\log_3 n - 1} 1/(\log_3 n - i)$$

$\sum_{i=0}^{\log_3 n-1} 1/(\log_3 n - i)$ is Harmonic series, is equal to $\log \log n$

$$T(n) = n.T(1) + \frac{n}{log3} \log \log n$$

Therefore,

$$T(n) = \Theta(n \log \log n)$$

(c) $T(n) = 4T(n/2) + n^2\sqrt{n}$

Solution:

$a = 4$; $b = 2$; $c = 2.5$

Compare $n^{\log_b a}$ with c

$n^{\log_b a} = n^{\log_2 4}$

$= n^2$

$c > n^{\log_b a}$

By Master Theorem case(3),

$$T(n) = \Theta(n^c)$$

Therefore.

$T(n) = \Theta(n^{2.5})$

$T(n) = \Theta(n^2\sqrt{n})$

(d) $T(n) = 3T(n/3 - 2) + n/2$

Solution:

As 'n' grows to very large value, '-2' term in $3T(n/3 - 2)$ will become less significant. So we can ignore that term.

The recursion equation becomes, $T(n) = 3T(n/3) + n/2$

$a = 3$; $b = 3$; $c = 1$

Compare $n^{\log_b a}$ with c

$n^{\log_b a} = n^{\log_3 3}$

$= n^1 = n$

$c = n^{\log_b a}$

By Master Theorem case(2),

$$T(n) = \Theta(n^c \log n)$$

Therefore.

$$T(n) = \Theta(n \log n)$$

(e) $T(n) = 2T(n/2) + n/\lg n$

Solution:

This problem is similar to problem 6(b).
So we can arrive general form as,

$T(n) = 2^k T(n/2^k) + n\sum_{i=0}^{k-1} 1/(\log n - i \log 2)$

$T(n) = 2^k T(n/2^k) + n\sum_{i=0}^{k-1} 1/(\log n - i)$

Let $n = 2^k$; $k = \log n$

$T(n) = 2^{\log n} T(1) + n\sum_{i=0}^{\log n - 1} 1/(\log n - i)$

$T(n) = n.T(1) + n\sum_{i=0}^{\log n - 1} 1/(\log n - i)$

$T(n) = n.T(1) + n.\log\log n$

Therefore,

$T(n) = \Theta(n \log\log n)$

(f) $T(n) = T(n/2) + T(n/4) + T(n/8) + n$

Solution:

Substitution Method:

Guess: $T(n) = O(n)$; $T(n) \leq c_2 n$ for $c_2 > 0$ and $n \geq n_0$

Assume: True for $m < n$ i.e $T(m) \leq cm$

To Prove: $T(n) \leq c_2 n$

Proof: Let $m = \frac{n}{2} < n$

$T(\frac{n}{2}) \leq c\frac{n}{2}$

$T(n) = T(n/2) + T(n/4) + T(n/8) + n$

$T(n) \leq c\frac{n}{2} + c\frac{n}{4} + c\frac{n}{8} + n$

$= c(\frac{n}{2} + \frac{n}{4} + \frac{n}{8}) + n$

$= nc(\frac{7}{8}) + n$

$= n(c\frac{7}{8} + 1)$

$T(n) \leq C_2 n$

Hence proved.

Similarly we can prove $T(n) = \Omega(n)$

$C_1 n \leq T(n)$

Therefore, $T(n) = \Theta(n)$