Assignment 2

Jasraj Singh Johal | 400434346 | johalj11

Question 1: Big Θ Proofs Using only the formal definition of Big O, Big Omega, and Big Theta, prove the following. You do not have to show any pre-proof analysis, but make sure that you justify every step in the proof.

a. $\frac{12n^3+5n-1}{n-1} \in \Theta(n^2)$

$\frac{12n^3+5n-1}{n-1} \in \Theta(n^2)$

To prove $f(n) = \frac{12n^3+5n-1}{n-1}$ is $\Theta(n^2)$, we need to show it $O(n^2)$ and $\Omega(n^2)$

To prove $f(n) \in O(n^2)$

$f(n)$ is $O(g(n))$ if there exists constants $c>0$ and $n_0 \geq 0$ such that $0 \leq T(n) \leq c.f(n)$ for all $n \geq n_0$.

$\rightarrow \frac{12n^3+5n-1}{n-1}$

$= 12n^2 + 12n + \frac{17n-1}{n-1}$ $\quad n \geq 2$ to

$0 \leq \frac{17n-1}{n-1}$ and $\frac{17n-1}{n-1} \leq C_0 n$

$0 \leq \frac{17n-1}{n-1} \leq C_0 n \quad (n \leq 2)$

$0 \leq 12n + \frac{17n-1}{n-1} \leq 12n + C_0 n$

$$0 \leq 12n + \frac{17n-1}{n-1} \leq 12n^2 + C_0 n^2 \quad [n \leq n^2]$$

$$0 \leq \frac{12n^3 + 5n - 1}{n-1} \leq 12n^2 + 12n + C_0 n$$

$$0 \leq \frac{12n^3 + 5n - 1}{n-1} \leq 24n^2 + C_0 n^2$$

$$0 \leq f(n) \leq 30n^2 \quad (\text{where } C_0 = 6)$$

By defn $f(n) \in O(n^2)$, where $n_0 = 2$ and $C = 30$

To prove $f(n) \in \Omega(n^2)$

$f(n)$ is $\Omega(g(n))$ if there exists constants $c > 0$ and $n_0 \geq 0$ such that $f(n) \geq c \cdot f(n)$ for all $n \geq n_0$.

Simplifying $f(n)$

$$12n^2 + 12n + 17 + \frac{16}{n-1}$$

$$\Rightarrow f(n) \geq C n^2$$

$$12n^2 \geq 12n^2$$

$$12n^2 + 12n + 17 \geq 12n^2$$

and $\frac{16}{n-1} > 0$

$$\Rightarrow 12n^2 + 12n + 17 + \frac{16}{(n-1)} \geq 12n^2$$

By defn of Ω, $f(n) \in \Omega(n^2)$

where $c = 12$ and $n \geq 2$

Given that $f(n)$ is $O(n^2)$ and $\Omega(n^2)$, so it is $\Theta(n^2)$ as well.

$$f(n) \in \Theta(n^2)$$

b. $5n^2 - 3n + 2 \notin \Theta(n^3)$

$5n^2 - 3n + 2 \notin \Theta(n^3)$

For $f(n) \notin \Theta(n^3)$, according to defn. of Big O and Big Omega, we need to prove its neither $O(n^3)$ or $\Omega(n^3)$.

For proving $f(n) \in O(n^3)$

$f(n) = 5n^2 - 3n + 2$

$f(n) \leq c \cdot n^3$

$5n^2 - 3n + 2 \leq c \cdot n^3$
$5n^2 + 3n^2 + 2n^2 \leq c \cdot n^3$
$10n^2 \leq c \cdot n^3$
$\therefore c = 10$

So, $f(n) \leq c \cdot n^3$

where $c = 10$
and $n \geq 1$

$\therefore f(n) \in O(n^3)$

For proving $f(n) \notin \Omega(n^3)$

Using defn. of Big Omega negation, $f(n) \notin \Omega(g(n))$ if for every $c > 0$ and $n_0 \geq 0$ there exists

an $n \geq n_0$ such that $f(n) < c \cdot g(n)$

$$f(n) < c \cdot n^3$$

$$5n^2 - 3n + 2 < c \cdot n^3$$

$$10n^2 < c \cdot n^3$$

so this becomes, in terms of n

$$\frac{10}{c} < n \qquad \text{for all } n \geq 1$$

Let C and n_0 be any no. such that $c > 0$ and $n_0 \geq 0$. Then any arbitrary no. such that $n > \frac{10}{c}$ and $n \geq 1$

$\because \quad n > \frac{10}{c}, \quad n^3 > \frac{10n^2}{c}$

$$10n^2 < cn^3$$

Also $\quad 5n^2 - 3n + 2 < 10n^2$

$$f(n) \leq cn^3$$

Hence $f(n) \notin \Omega(n^3)$, for arbitrary c and n_0.

As given, $f(n)$ is $O(n^3)$ but not $\Omega(n^3)$, it cannot be $\Theta(n^3)$

Using the formal definition of Big O, prove the transitive property:

c. If $d(n) \in O(f(n))$ and $f(n) \in O(g(n)$ then $d(n) \in O(g(n))$

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

then $d(n) \in O(g(n))$

$f(n)$ is $O(g(n))$ if there exists constants $c > 0$ and $n_0 \geq 0$ such that $0 \leq T(n) \leq c.f(n)$ for all $n \geq n_0$.

By defn, $d(n) \in O(f(n))$

$0 \leq d(n) \leq c_1 . f(n)$

where $c_1 > 0$ and $n \geq n$

where $n_1 \geq 0$

By defn, $f(n) \in O(g(n))$

$0 \leq f(n) \leq c_2 . g(n)$

where $c_2 > 0$ and $n \geq n_2$

where $n_2 \geq 0$

Multiplying c_1 by the above eq

$\Rightarrow c_1 . 0 \leq c_1 . f(n) \leq c_1 . c_2 . g(n)$

$\Rightarrow 0 \leq c_1 . f(n) \leq (c_1 . c_2) . g(n)$

where $c_1 > 0$, $c_2 > 0$ and $n \geq n_2$
$(n_2 \geq 0)$

So according to both the above inequalities

$$0 \leq d(n) \leq c_1 \cdot f(n) \leq (c_1 \cdot c_2)\, g(n)$$

$c_1, c_2 > 0$ and $n_3 \geq \max(n_1, n_2)$

and $n_1, n_2 \geq 0$

$$\therefore \quad 0 \leq d(n) \leq (c_1 \cdot c_2) \cdot g(n)$$

$c_1, c_2 > 0$ and $n \geq n_3$ and $n_3 \geq 0$

So, by defn. of Big O

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

Using the rules on slide 9 of the Week 4 Merge Sort slides, prove the following. Make sure you cite the rules you are using at each step:

d. $2n^3 + 4n^2 \log n \in O(n^3)$

$2n^3 + 4n^2 \log n \in O(n^3)$

= By rule 5: Polynomial

$2n^3 \in O(n^2)$

= By rule 8 : Log Powers

$\log n \in O(n)$

= By rule 5 : Polynomial

$4n^2 \in O(n^2)$

= By rule 3 : Multiplication

$4n^2 \cdot \log n \in O(n \cdot n^2) = O(n^3)$

= By rule 2: Addition

$2n^3 + 4n^2 \log n \in$

$O(\max(n^3, n^3))$

$= O(n^3)$

Question 2: Basic Sorting Write pseudocode for a linked list swap operation that accepts three parameters – a doubly linked list L, and two pointers a and b that point to Node elements inside the linked list. Change all the relevant previous and next links so that Nodes a and b swap places within the list. You may assume that Node b comes after Node a in the list, and that they are not the same Node. Make sure you cover all relevant special cases

```
function swapNodes(L, a, b):
    # Handle edge cases where 'a' or 'b' is the head or tail of the list.
    if a == L.head:
        # If 'a' is the head, update the head to 'b'
        L.head = b
    if b == L.tail:
        # If 'b' is the tail, update the tail to 'a'
        L.tail = a

    # Store previous and next nodes of 'a' and 'b'.
    aPrevious = a.previous
    bNext = b.next
    aNext = a.next
    bPrevious = b.previous

    # Handle the case where 'a' and 'b' are adjacent.
    if aNext == b:
        aPrevious.next = b
        b.previous = aPrevious
        a.next = bNext
        bNext.previous = a
        a.previous = b
        b.next = a
    else:
        # Handle the general case.
        a.next = bNext
        if bNext != NONE:
            bNext.previous = a
        a.previous = bPrevious
        bPrevious.next = a
        b.next = aNext
        aNext.previous = b
        b.previous = aPrevious
        if aPrevious != NONE:
            aPrevious.next = b

    return
```

Question 3: Quicksort Even with a random shuffle, quicksort can get unlucky. Give 4 different size 5 arrays of integers that will cause quicksort to exhibit its worst-case running time of O(n^2). Each example should show a different relative ordering of the elements in the array, not just different numbers. For example [0, 10, 20, 5, 2] and [1, 4, 5, 3, 2] are not different examples because the relative order of the elements is the same. In each case, explain why this arrangement will lead to O(n^2) running time.

Example 1 (Descending Order): Array: [5, 4, 3, 2, 1]

When the array is sorted in descending order with different numbers, using the last element (1) as the pivot consistently results in unbalanced partitions. This is because all other elements are greater than the pivot, leading to a worst-case time complexity of O(n^2).

Example 2 (Ascending Order): Array: [-2, -1, 3, 4, 5]

The array is sorted in ascending order with different numbers. If we use the first element (-2) as the pivot, it still creates unbalanced partitions. This is because all other elements (ranging from -1 to 5) are greater than the pivot. Consequently, the partitioning step consistently produces partitions where one side contains all the elements, resulting in a worst-case time complexity of O(n^2).

Example 3 (Alternating Order): Array: [10, 30, 20, 50, 40]

With an alternating order of different numbers, selecting the first element (10) as the pivot also leads to unbalanced partitions. The varying distribution of numbers around the pivot results in an inefficient partitioning process and a worst-case time complexity of O(n^2).

Example 4 (All the Same Element): Array: [1, 1, 1, 1, 1]

When all elements are the same (in this case, all are 1), and any element is chosen as the pivot, it leads to unbalanced partitions every time. This is because every element in the array is equal, so one side of the partition will always be empty, and the other side will have all the elements. Consequently, the algorithm will make many recursive calls, as it effectively makes no progress towards sorting the array, resulting in a worst-case time complexity of O(n^2).

Question 4: Recurrence Relations Suppose I have an algorithm that takes an array as input, does work proportional to n^2 on the data, then splits the data into three groups. The size of each group depends on the exact input and can vary between $[n/6]$ and $[n/2]$. Then it recursively calls itself on each of the 3 groups. The base case is $n \leq 3$ (the algorithm does $O(1)$ work in this case). Model the worst-case time complexity of this algorithm with a recurrence relation, then use a recursion tree to explain why the algorithm runs in $O(n$^2 log $n)$ (not big Theta) time.

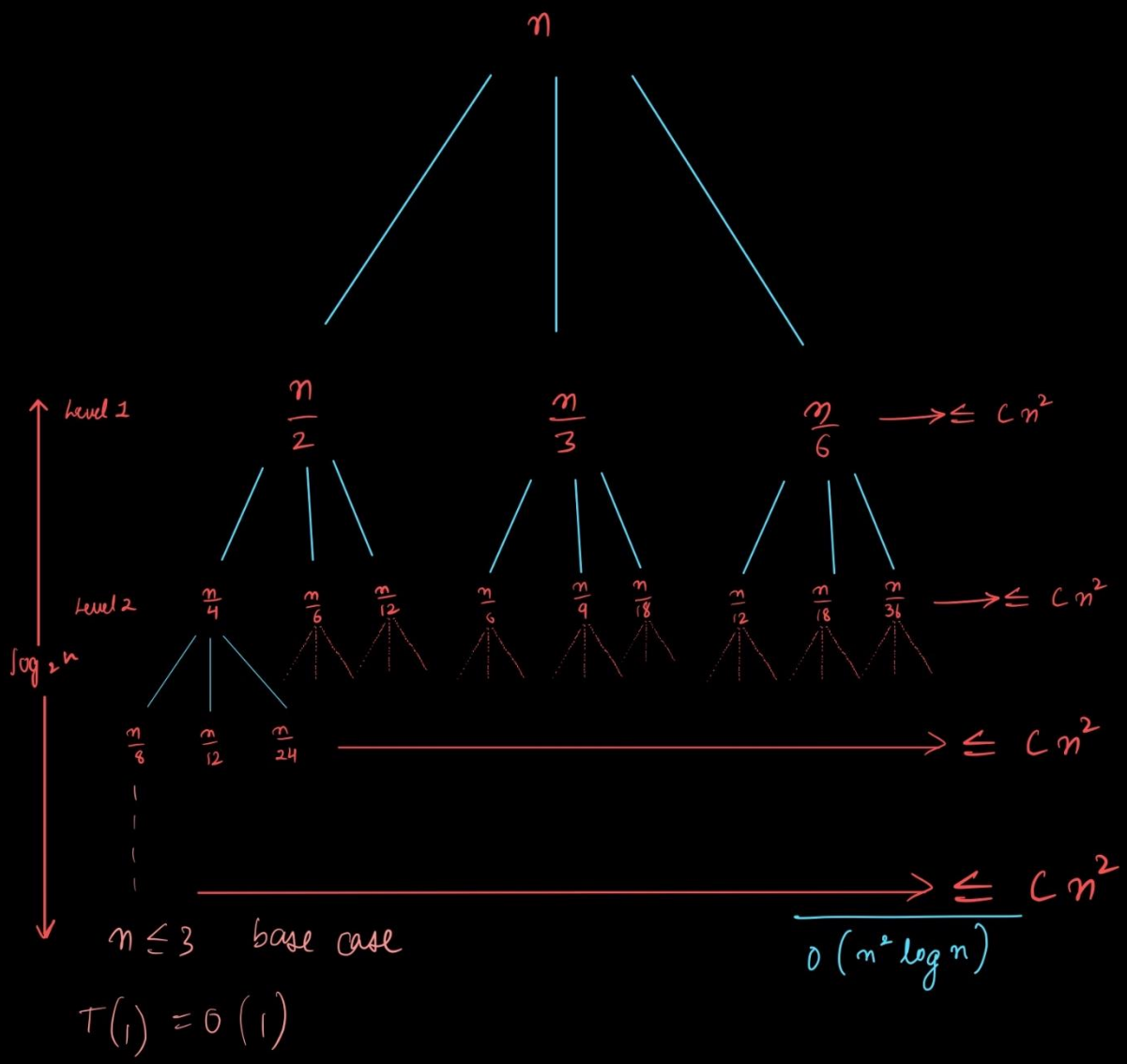
Lets denote the time complexity of the algorithm for an input size of 'n' as $T(n)$

It does $o(n^2)$ work on the data before splitting it into three groups.

Worst case would be unbalanced distribution of groups from the work, here one work will get most of the work.

For eg. $\left(\frac{n}{2}\right), \left(\frac{n}{3}\right), \left(\frac{n}{6}\right)$

Therefore $T(n) = c.n^2 + T\left(\frac{n}{2}\right) + T\left(\frac{n}{3}\right) + T\left(\frac{n}{6}\right)$


n
n/2
n/3
n/6
≤ cn²
Level 1
Level 2
log₂ n
n/4
n/6
n/12
n/6
n/9
n/18
n/12
n/18
n/36
≤ cn²
n/8
n/12
n/24
≤ cn²
≤ cn²
n ≤ 3 base case
T(1) = O(1)
O(n² log n)

Level 1 biggest leaf is $\left(\frac{n}{2}\right)$

Level 2 biggest leaf is $\left(\frac{n}{4}\right)$

At max height :

$$\frac{n}{2^i} = 1 \quad \Rightarrow \quad \frac{n}{2^i} = 1$$

$$\Rightarrow \quad n = 2^i$$

$$\log_2 n = i$$

$$i = \log_2 n$$

Therefore height of the tree is $(\log_2 n)$

At each level $\leq c \cdot n^2$

$$T(n) \leq c n^2 \cdot \log_2 n$$

$$T(n) \leq c \cdot n^2 \cdot \log_2 (n)$$

$$\therefore \quad \boxed{T(n) \in O(n^2 \log_2 n)}$$