COMPSCI 2C03 Assignment 2

Avyya Singh | 400432568 | singa274

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

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

First lets prove $f(n) \in O(n^2)$

$f(n)$ is $O(g(n))$ if there exists a constant $C>0$ and $n_0>0$ such that for all $n>n_0$,

$0 \leq f(n) \leq c \cdot g(n)$

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

$\Rightarrow 12n^2+12n+\frac{17n-1}{n-1}$ [$n \geq 2$ to prevent division by 0]

$0 \leq \frac{17n-1}{n-1}$ and $\frac{17n-1}{n-1} \leq c_0 n$

$0 \leq \frac{17n-1}{n-1} \leq c_0 n$ $(n \leq 2)$

$0 \leq 12n + \frac{17n-1}{n-1} \leq 12n + c_0 n$

Since $n^2 \geq n$,

$0 \leq 12n + \frac{17n-1}{n-1} \leq 12n^2 + c_0 n^2$

$0 \leq f(n) \leq 12n^2 + 12n^2 + c_0 n^2$

$0 \leq f(n) \leq 24n^2 + c_0 n^2$ [$c_0 = 10$]

$\rightarrow 0 \leq f(n) \leq 34n^2$

By definition

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

$n_0 = 2$ and $c = 34$

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

after simplifying $f(n)$:

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

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

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

and $\frac{16}{n-1} > 0$ for all $n \geq 2$

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

So by definition of Big Omega, there exists a 'c' and 'n' such that $f(n) \geq c \cdot n$

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

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

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

To prove that $f(n) = 5n^2 - 3n + 2 \notin \Theta(n^3)$ we need to prove it is neither $O(n^3)$ or $\Omega(n^3)$.

To prove $f(n) \in O(n^3)$

$5n^2 + 3n^2 + 2n^2 \leq c \cdot n^3$

$10n^2 \leq cn^3$

so, $c = 10$

Therefore, $5n^2 - 3n + 2 \leq 10n^3$

for any $n \geq 1$

so, $f(n) \in O(n^3)$

To prove $f(n) \notin \Omega(n^3)$

Using definition of Big Omega, for every $c > 0$ and $n_0 \geq 0$ there exists an $n \geq n_0$ such that

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

Since, $5n^2 + 3n^2 + 2n^2 < cn^3$

$10n^2 < cn^3$

$\frac{10n^2}{c} < n^3$, $\frac{10}{c} < n$

and $n \geq 1$.

Let c and n_0 be any numbers such that $c > 0$ and $n_0 \geq 0$. Then any arbitrary number such that $n > \frac{10}{c}$ and $n \geq 1$.

Since, $n > \frac{10}{c}$, $n^3 > \frac{10n^2}{c}$ so $10n^2 < cn^3$

And since $5n^2 - 3n + 2 < 10n^2$,
$f(n) \leq cn^3$

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

Given that $f(n)$ 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 $d(n) \in O(g(n))$

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

$0 \leq d(n) \leq c_1 \cdot f(n)$
where $c_1 > 0$ and $n \geq n_1$
where $n_1 \geq 0$

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

$0 \leq f(n) \leq c_2 \cdot g(n)$
where $c_2 > 0$ and $n \geq n_2$
where $n_2 \geq 0$

Multiplying c_1 by the above inequality equation

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

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

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

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

So by definition 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
$\Rightarrow 2n^3 \in O(n^2)$

By rule 8: Log Powers
$\Rightarrow \log n \in O(n)$

By rule 5: Polynomial
$\Rightarrow 4n^2 \in O(n^2)$

By rule 3: Multiplication
$\Rightarrow 4n^2 \cdot \log n \in O(n \cdot n^2) = O(n^3)$

By rule 2: Addition
$\Rightarrow 2n^3 + 4n^2 \log n \in O(\max(n^3, n^3))$
$= O(n^3)$

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.

1. Handle edge cases where *a* or *b* is the head or tail of the list.
2. Handle the case where *a* and *b* are adjacent.
3. Handle the general case.

```
function swapNodes(l, a, b):

    # Step 1: Handle cases where a or b is the head or tail
    if a == l.head:
        l.head = b
    elif b == l.tail:
        l.tail = a

    # Step 2: Handle cases where a and b are adjacent
    if a.next == b:
        # swap pointers for a and b when they're adjacent
        if a.previous != NONE:  # check if a is not head
            a.previous.next = b
        b.previous = a.previous
        a.next = b.next
        if b.next != NONE:  # check if b is not tail
            b.next.previous = a
        b.next = a
        a.previous = b

    # Step 3: Handle the general case where a and b are not adjacent
    else:
        aNext = a.next
        bNext = b.next
        aPrevious = a.previous
        bPrevious = b.previous

        # Update pointers for node a
        if aPrevious != NONE:
            aPrevious.next = b
        b.previous = aPrevious
        if aNext != NONE:
            aNext.previous = b
        b.next = aNext

        # Update pointers for node b
        if bPrevious != NONE:
            bPrevious.next = a
        a.previous = bPrevious
        if bNext != NONE:
            bNext.previous = a
        a.next = bNext

    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.

- Quicksort's worst-case time complexity of $O(n^2)$ arises when the chosen pivot is always the smallest or the largest element, leading to unbalanced partitioning at each step. For a size 5 array, this means that each recursive call of quicksort will be dealing with a subarray of size 4, then 3, then 2, etc. The partitioning process will have to compare the pivot with every other element in the subarray, resulting in $O(n^2)$ comparisons in total.

  Given this, here are 4 different size 5 arrays that will cause quicksort to exhibit its worst-case running time of $O(n^2)$ if the pivot is chosen as the first element:

1. Descending Order:

- Array: [5, 4, 3, 2, 1]
- Explanation: When the first element (5) is the pivot, all other numbers are to its right. This continues in each step, always selecting the largest number, causing maximum comparisons.

2. Ascending Order:

- Array: [-3,-2,-1, 0, 1]
- Explanation: Using the first number (-3) as the pivot, all others are to its right. This pattern repeats, always picking the smallest number, leading to maximum comparisons.

3. Alternating, starting with high:

- Array: [4,-3, 3,-2, 2]
- Explanation: With 4 as the pivot, the remaining numbers are to its left. The trend continues, as the biggest number is consistently chosen, resulting in unbalanced splits.

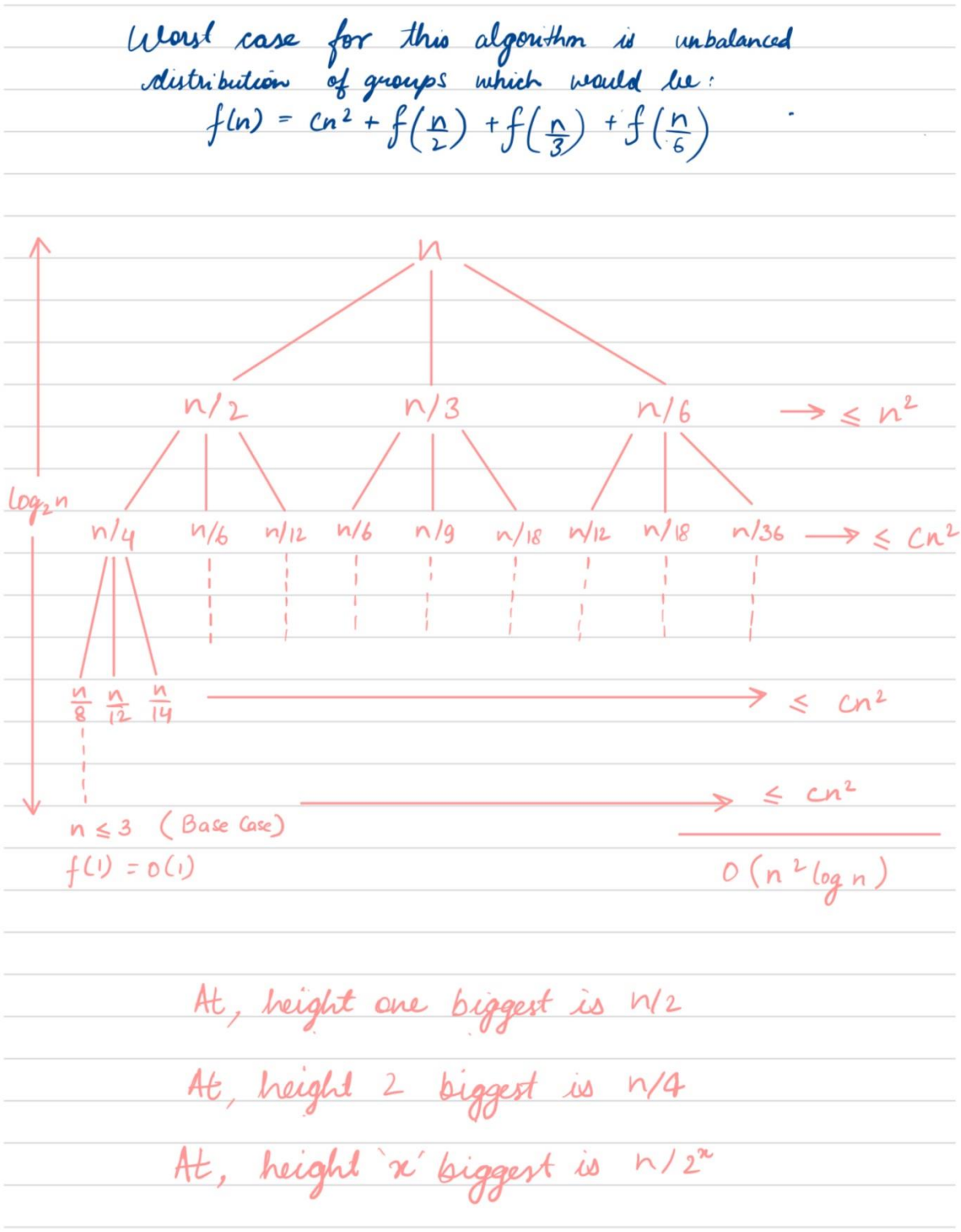
4. Same Number:

- Array: [10, 10, 10, 10, 10]
- Explanation: With 10 as the pivot (regardless of the element chosen), all other elements are equal. This means in each step, the partitioning does no actual division, leading to repetitive, inefficient operations.

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 $\lceil n/6 \rceil$ and $\lceil n/2 \rceil$. 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.



At max height,

$$\rightarrow \frac{n}{2^n} = 1$$

$$\Rightarrow n = 2^n$$

$$\rightarrow n = \log_2 n$$

Total height of the tree is $\log_2 n$

At each level $\leq cn^2$

$$f(n) \leq cn^2$$

Therefore, $f(n) \in O(n^2 \log_2 n)$