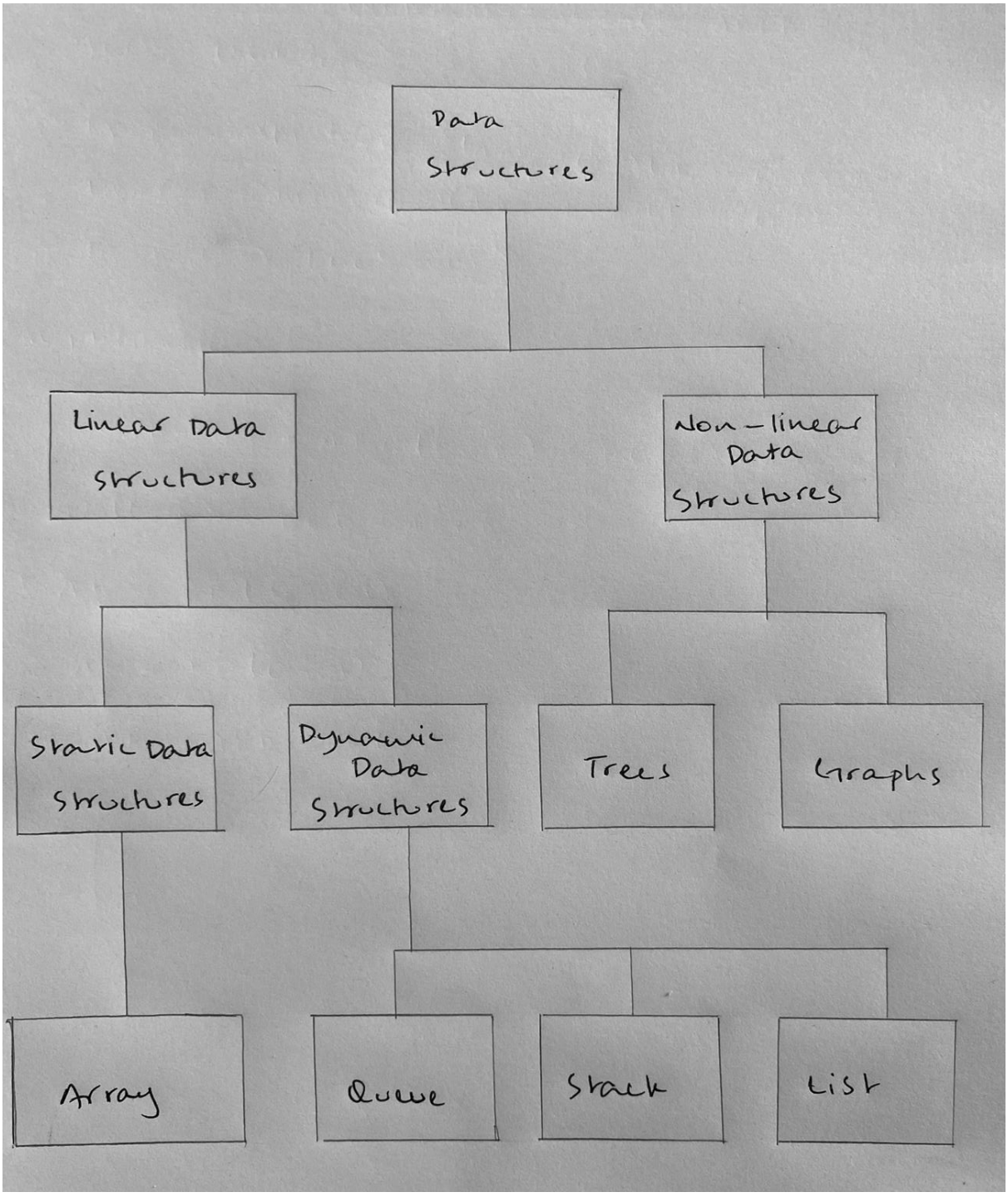
21BDS0340

Abhinav Dinesh Srivatsa

Data Structures and Algorithms

Assignment – I

Question 1



Data structures are classified by how they store data, whether they have limits and how they function.

Maps and graphs store data with nodes and links between them.

Arrays store data by fixing a limit on how large it is and stores contiguously.

Queues and stacks implement arrays but change how elements are inserted into and exit the data structure.

Lists use nodes of data that are not necessarily contiguous, which is its main difference from an array.

Question 2

```
for i = 0 to N
    for j = 0 to N
        - do something -
for k = 0 to N
    - do something -
```

The above algorithms time complexity can be found by knowing how long each loop runs

Nested loop: N times for outer loop

N times for inner loop

$\therefore N^2$ times

2^{nd} loop: N times

$\therefore$ Total is $N^2 + N$

But as N increases $N^2 >> N$

$\therefore$ Time complexity is $O(N^2)$

Question 3

Recurrence tree method goes through a recursive algorithm and plots out the various time complexities. Then we can sum the time for each operation to find the time complexity.

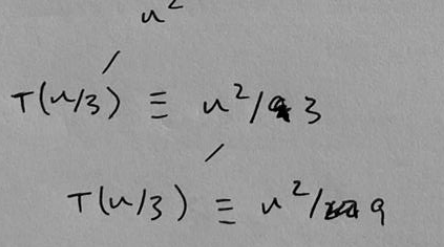
Master method is a derivative of recurrence relation that can directly tell us the time complexity based on the values of a and b in a recurrence of the following type:

$$T(n) = a\,T(n/b) + f(n), \quad a \geq 1,\ b > 1$$

Sample Algorithm:

$$T(n) = T(n/3) + n^2$$

Recurrence Tree:



Formula GP: $n^2 + \frac{n^2}{3} + \frac{n^2}{9} + \frac{n^2}{27} + \ldots$

Sum = $n^2 \cdot \frac{1(1-(1/3)^m)}{1-1/3} = \frac{n^2}{1-1/3}$ as $n \to \infty$

∴ Time complexity = $O(n^2)$

Question 4

a. A ^ Y / ((* Z) + E

Symbol	Stack	Postfix
A		A
^	^	A
Y	^	AY
/	/	AY^
(	/(	AY^
(	/(	AY^(
*	/(*	AY^(
Z	/(*	AY^(Z
)	/	AY^(Z*
+	+	AY^(Z*/
E		AY^(Z*/E +

∴ Postfix of A^Y/((*Z) + E

= AY^(Z*/E+

6. A * B + C / D

Reversed:

D / C + B * A

Symbol	Stack	Postfix
D		D
/	/	D
C	/	DC
+	+	DC/
B	+	DC/B
*	+*	DC/B
A		DC/BA*+

Prefix = +*AB/CD

c. 2 5 3 6 + * * 15 / 2 -

Symbol	Stack
2	2
5	2 5
3	2 5 3
6	2 5 3 6
+	2 5 9
*	2 45
*	90
15	90 15
/	6
2	6 2
-	4

∴ The answer is 4

Question 5

Simple

Insertion at the end and deletion at the start, strictly follows first in first out rule.

Circular

Queue created with nodes where the last one points to the first as it's next. Better memory utilisation when the first elements are deleted.

Priority

Queue where each element is executed according to the priority associated with it. Same priority elements are executed by order.

Deque

A queue where insertion and deletion can happen at the rear and front. This does not strictly follow first in first out rule

Operations

Enqueue - insert element at start

Dequeue - delete element from end

Peek - show the next element to be deleted

Display - show the full queue