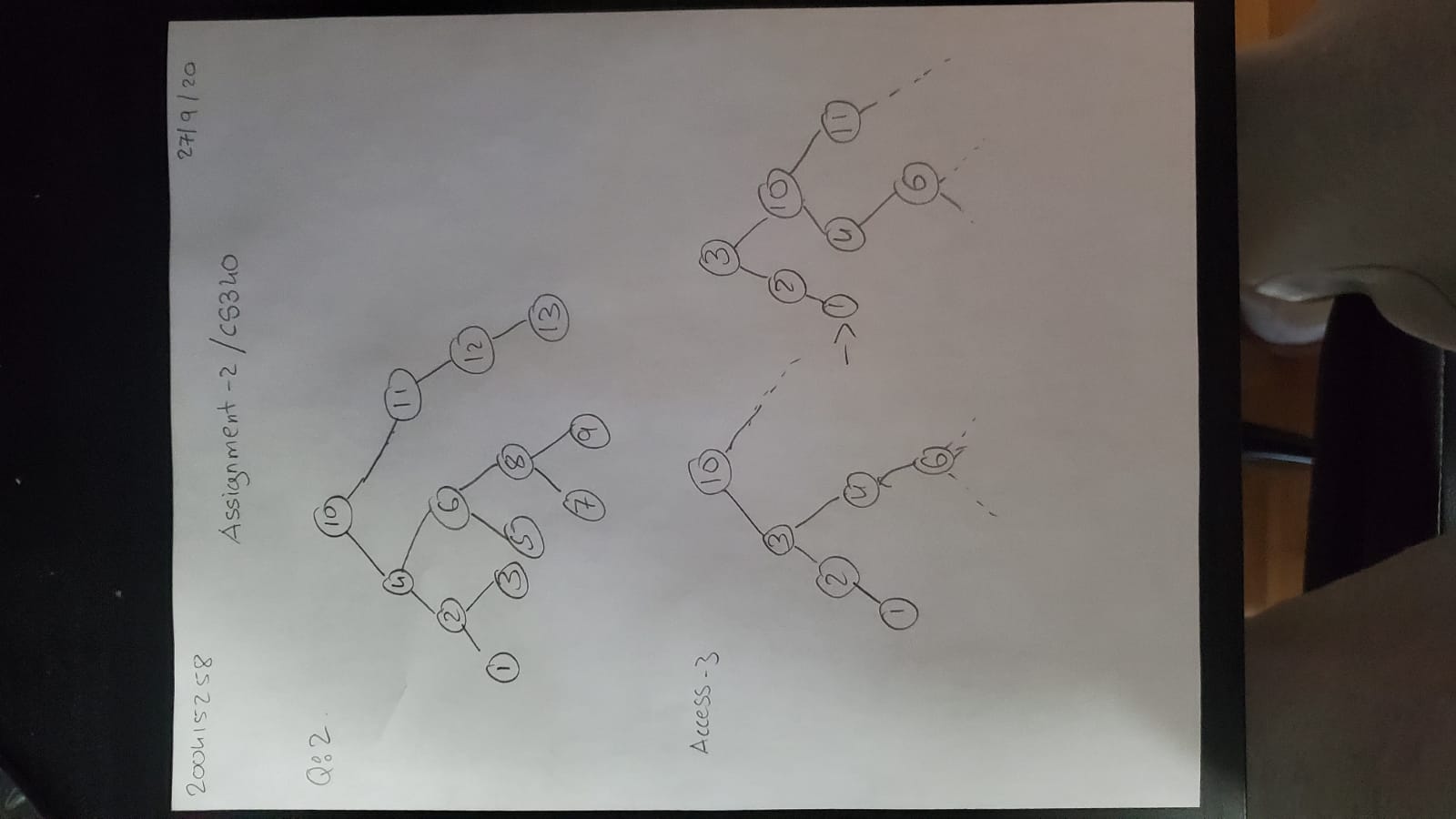
**Assignment 2: CS 340**

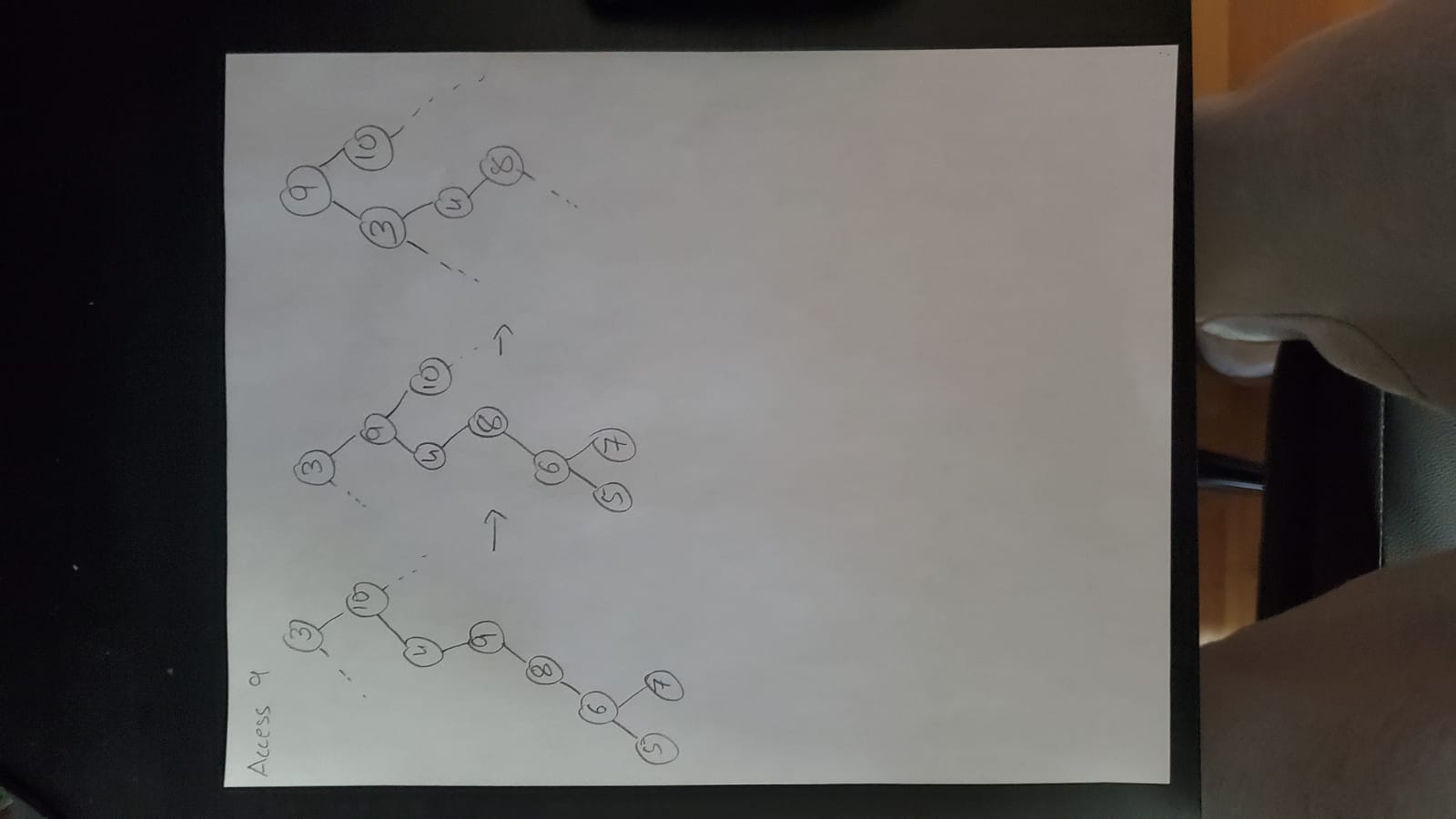
**Problem 1 (6 marks). Design an algorithm (in pseudocode) that lists out the nodes of a binary tree in level-order (breadth-first): it first lists the root, then the nodes at depth 1, then the nodes at depth 2, and so on. Assume that every node of a tree is represented in a structure that contains the data element stored at the node, a pointer to the left child of the node, and a pointer to the right child of the node. It is required that your algorithm has a worst case running time of O(N), where N is the number of nodes in the tree. Explain why your algorithm fulfills this requirement.**

Ans: *Please see end-of-the file. (word was messing everything so I had to do it that way, hope I was not too frustrating☺).*

Explanation: If you see line 16 to 18 they all are assigning values so time taken will be constant for all values of N. For while loop, we are printing and reassigning values therefore every visit(each iteration of loop) we have constant time C. Now for each visit, we are visiting every node just once with the help of a queue. So time will be relative to N which is O(N).

**Problem 2 (6+2 marks). (a) Show the result of accessing the keys 3, 9, 1, 5 in order in the splay tree in textbook gure 4.76 (page 185). Additionally, show the result of each splaying step (i.e., show intermediate trees that illustrate the process). (This is an extension of textbook problem 4.27.) (b) Show the result of deleting the element with key 6 in the resulting splay tree for Problem 2(a). (This is textbook problem 4.28.)**

Ans: 



**Problem 3 (3+4 marks). (a) What is the worst case, average case, and best case running time of insertion into slot a[0] of an (unsorted) array a containing N integers, representing a stack? Give bounds and explain your answer. (b) What is the amortized running time of insertion of N integers into an initially empty array a, representing a stack, if every insertion takes place at a[0]? Give a tight O bound and explain your answer.**

Ans:

1. Considering an array of size 10. In this case inserting at a[0] we need two things to perform. First we have to place all the elements to their next position i.e. a[0] should be at a[1], a[1] at a[2] and so on, that is to make an empty slot for insertion at first position. Second insertion at index 0. Now time complexity for worst, average and bst case is same as we are not increasing the number of inputs. For any input number of elements in a will be N so the number of operations will be N as well. So even if a has 100 million numbers in it, with any input at a[0] has to perform constant steps which is N->100 million. Now as the worst case and best case has same complexity, the algorithme has Θ(N). Also it has a tight upper and lower bound.
2. Now worst case average time will always be less expensive than some worst case time of specific operation. So the worst case for N numbers would be N^2 at upper bound which is O(N^2). Now amortized analysis is average of worst time i.e. N^2/N = N = O(N).

**Problem 4 (2+2+4 marks). Consider the problem of implementing a k-bit binary counter that counts upward from 0. We use an array a[0..k-1] of k bits. A binary number stored in the array has its lowest-order bit in a[0] and its highest-order bit in a[k-1].**

Ans: a) worst case scenario would be when all the bits are 1 i.e. O(k) as all the bits till the end needs to be flipped. For f(k) = Θ(k),

C1 \* k < f(k) < C2 \* k where C1 and C2 are constants.

i.e. f(k) is upper as well as lower bounded in between C1 and C2 by k.

b)

a[0], a[1], a[2], a[3], a[4], a[5]

0, 0 0, 0, 0, 0 -> Exc 0

1, 0 0, 0, 0, 0 -> Exc 1

0, 1 0, 0, 0, 0 -> Exc 2

1, 1 0, 0, 0, 0 -> Exc 3

0, 0 1, 0, 0, 0 -> Exc 4

1, 0 1, 0, 0, 0 -> Exc 5

0, 1 1, 0, 0, 0 -> Exc 6

1, 1 1, 0, 0, 0 -> Exc 7

0, 0 0, 1, 0, 0 -> Exc 8

1, 0 0, 1, 0, 0 -> Exc 9

0, 1 0, 1, 0, 0 -> Exc 10

1, 1 0, 1, 0, 0 -> Exc 11

0, 0 1, 1, 0, 0 -> Exc 12

1, 0 1, 1, 0, 0 -> Exc 13

0, 1 1, 1, 0, 0 -> Exc 14

1, 1 1, 1, 0, 0 -> Exc 15

0, 0 0, 0, 1, 0 -> Exc 16

c) So amortized analysis is the worst case average time on upper bound and so we can analyze individual case all flips from part B of this question. Average run time would be flips / number of execution

i.e (1 + 2 + 1 + 3 + 1 + 2 +1 + 4 + 1 + 2 + 1 + 3 + 1 + 2 +1 + 5 ) / 16 = 1.9375 ≈ 2

So amortized time complexity for M execution in IncrCounter is,

At = ( (1 \* (M / 2)) + (2 \* (M / 4)) + .... ) / M

#include <stdio.h>

using namespace std;

// N is number of nodes

const int N = 20;

struct Node

{

int value = NULL;

Node\* leftChild = nullptr;

Node\* rightChild = nullptr;

};

void traverse(Node root){

Node\* queue[N]; // <--- line 16

queue[0] = root.leftChild;

queue[1] = root.rightChild; // <--- line 18

while (queue[0] != nullptr)

{

visit(queue[0]->value);

queue.push(queue[0]->leftChild);

queue.push(queue[0]->rightChild);

queue.pop();

}

}

void visit(int value){

cout << "Node" << value << " visited\n";

}