**HW2**

**1a.**

**[T(n)]**

**/ (4 branches)**

**[T(n/4)] ->**

**| (4 branches for each, 16 total)**

**[T(n/16)] ->**

**… and so on …**

**Height of tree =**

**I don’t know the induction part.**

**1b.**

**[T(n)]**

**/ (8 branches)**

**[T(n/2)] ->**

**| (8 branches for each, 64 total)**

**[T(n/4)] ->**

**…and so on…**

**|**

**[T(n/)] ->**

**Contribution of row:**

**# of rows: (solve )**

**total:**

**1c.**

**[T(n)]**

**/ (5 branches)**

**[T(n/4)] -> )**

**| (4 branches for each, 16 total)**

**[T(n/16)] ->**

**… and so on …**

**|**

**[T(n/)] ->**

**Contribution of row:**

**# of rows:**

**2.**

**SET left to start of array  
SET right to end of array**

**FUNCTION FindMatch(left, right)**

**LOOP through array elements while left < right =>**

**SET mid as (left + right) / 2  
 IF A[mid] = mid  
                   RETURN mid                
           ELSE IF A[mid] > mid**

RETURN FindMatch(left, mid − 1) **ELSE**

RETURN FindMatch(mid + 1, right)

END ELSE-IF

END LOOP

RETURN statement saying “no solution”

END FUNCTION

Since the array is sorted in ascending order, we are creating a binary search. It goes to the midpoint of each iteration of the array and makes a comparison between the midpoint slot number and the value inside. For example, if array A goes from slots 0-10 and A[5] = 7, then the array will be cut in half and the old mid becomes the new right boundary because the value inside is higher than the slot. This process repeats until A[i] = i or left < right where it returns no solution. So with every iteration, the array is divided in half, so the total complexity is O(logN). I don’t know the recurrence formula part.

3.

SET index to start of array

SET n to length of array

SET majority to 2n/3

**FUNCTION FindMajorityElement(index)**

**WHILELOOP index goes through loop until index value is past end of array**

**SET beginIndex to index**

SET count to 0

**FORLOOP through elements until index hits end of array, increase index by 1 each iteration**

**IF A[beginIndex] = A[index]**

**ADD count by 1**

**ENDIF**

**ENDFORLOOP**

**IF count >= majority**

**RETURN A[index] as the majority element**

**ELSE**

**RETURN FindMajorityElement(index + 1)**

ENDWHILE

RETURN statement saying “no solution”

END FUNCTION

We first establish the beginning of the array. Then we iterate through and compare that set beginning index with every index down the line (including itself as the first number). Every time the numbers match, it adds one more to count. After it loops entirely through, it checks to see if the amount of same numbers matching beginIndex is higher than the majority threshold. If so, it returns that value. If not, it calls the method back and restarts the process on a new beginIndex one slot over. It repeats this iteration until it either gets a count that is equal or greater than the majority threshold or until it gets past the final index, where it returns no solution. Since we are iterating the array every single time and we’re to recurse up until the amount of n slots it has, the runtime is O(nlogN). I don’t know the recurrence formula part.

4. Part 1

SET index to start of array

SET count to 1

**FUNCTION FindHighestIncrement(index)**

**WHILELOOP index goes through loop until index value is past end of array**

**IF A[index] < A[index + 1]**

**ADD count by 1**

**ELSE**

**IF count > highestIncrement**

**SET highestIncrement to count**

**ENDIF**

**SET count to 0**

**ENDELSE-IF**

**RETURN FindHighestIncrement(index + 1)**

ENDWHILE

END FUNCTION

I don’t know the recurrence formula part.

4. Part 2

I don’t know.