# Lab2

## Prob1

int[] arrays(int n) {  
 int[] arr = new int[n];  
 for(int i = 0; i < n; ++i){  
 arr[i] = 1;  
 }  
 for(int i = 0; i < n; ++i) {  
 for(int j = i; j < n; ++j){  
 arr[i] += arr[j] + i + j;  
 }  
 }  
 return arr;  
}

**Answer:**

The run time of

for(int i = 0; i < n; ++i){  
 arr[i] = 1;  
}

is O(n)

The run time of

for(int i = 0; i < n; ++i) {  
 for(int j = i; j < n; ++j){  
 arr[i] += arr[j] + i + j;  
 }  
}

is O(n2)

The total run time is O(n) + O(n2) = O(n2)

## Prob2

1. Algorithms

I wrote 2 algorithms for this problem

Algorithm merge(A, B)  
 Input: 2 sorted arrays need to merged A & B  
 Output: The sorted array contains all elements from A & B  
  
 C = new Array[A.length + B.length]  
 for i <- 0 to A.length - 1 do  
 C[i] = A[i]  
  
 for i <- 0 to B.length - 1 do  
 C[A.length + i] = B[i]  
  
 for i <- 0 to C.length - 1 do  
 for j <- i to C.length - 1 do  
 if C[i] > C[j] then  
 temp <- C[i]  
 C[i] <- C[j]  
 C[j] <- temp  
  
 return C

Algorithm merge2(A, B)  
 Input: 2 sorted arrays need to merged A & B  
 Output: The sorted array contains all elements from A & B  
  
 C = new Array[A.length + B.length]  
 i <- 0 // keep track index of A  
 j <- 0 // keep track index of B  
 p <- 0 // keep track index of C  
  
 while i < A.length & j < B.length do  
 if A[i] <= B[j] then  
 C[p++] <- A[i++]  
 else  
 C[p++] <- B[j++]  
  
 while i < A.length  
 C[p++] <- A[i++]  
  
 while j < B.length  
 C[p++] <- B[j++]  
  
 return C

1. Run time:

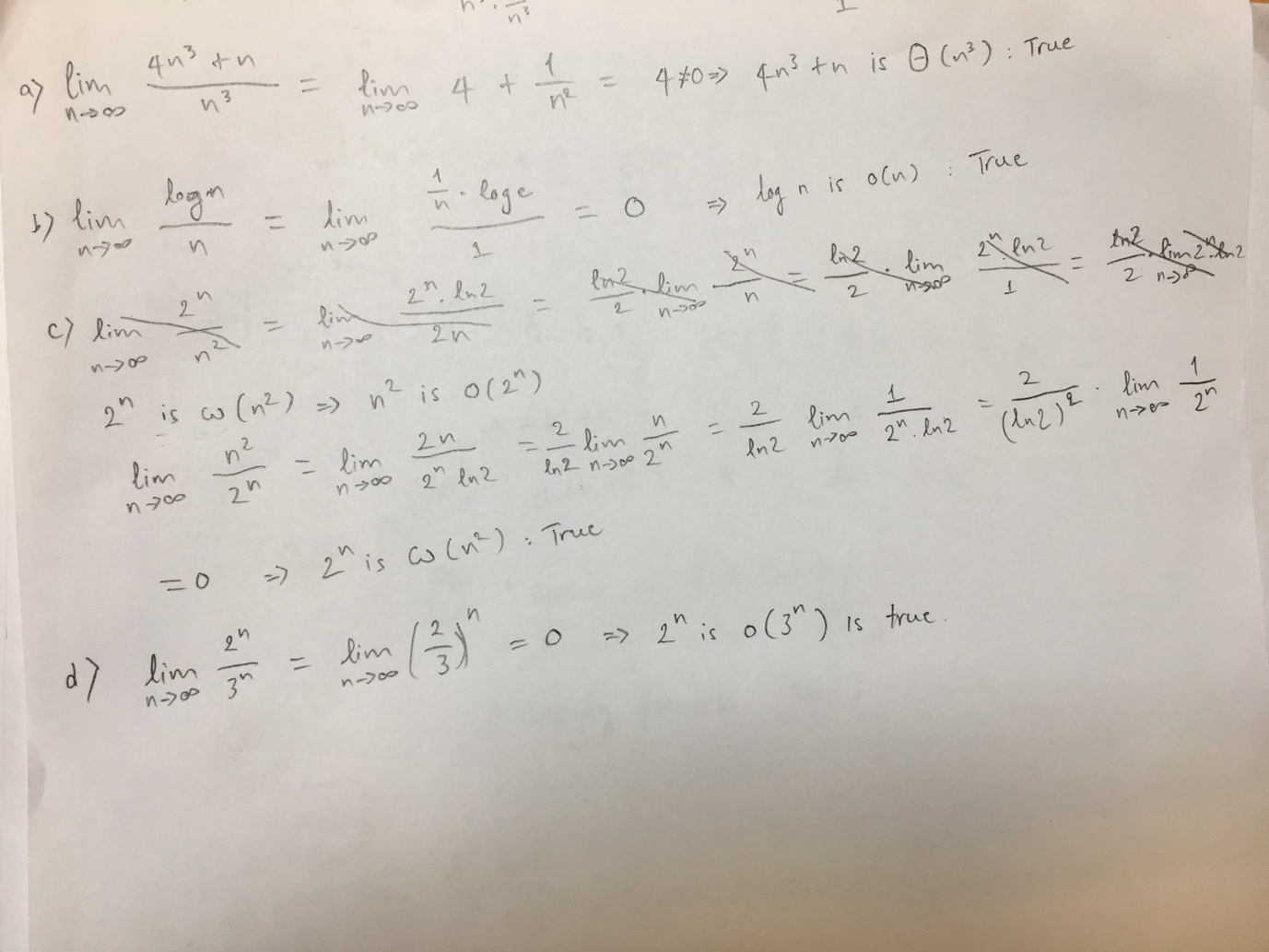
The run time of merge is O(n) + O(n) + O(n2) = O(n2)

The run time of merge2 is O(n)

1. Implement in Java

public static int[] merge(int[] arr1, int[] arr2) {  
 int[] arr = new int[arr1.length + arr2.length];  
 for (int i = 0; i < arr1.length; i++) {  
 arr[i] = arr1[i];  
 }  
  
 for (int i = 0; i < arr2.length; i++) {  
 arr[arr1.length + i] = arr2[i];  
 }  
  
 for (int i = 0; i < arr.length; i++) {  
 for (int j = i; j < arr.length; j++) {  
 if (arr[i] > arr[j]) {  
 int temp = arr[i];  
 arr[i] = arr[j];  
 arr[j] = temp;  
 }  
 }  
 }  
 return arr;  
}  
  
public static int[] merge2(int[] arr1, int[] arr2) {  
 int[] arr = new int[arr1.length + arr2.length];  
 int i = 0, j = 0, pos = 0;  
  
 while (i < arr1.length && j < arr2.length) {  
 if (arr1[i] <= arr2[j]) {  
 arr[pos++] = arr1[i++];  
 } else {  
 arr[pos++] = arr2[j++];  
 }  
 }  
  
 while (i < arr1.length) {  
 arr[pos++] = arr1[i++];  
 }  
  
 while (j < arr2.length) {  
 arr[pos++] = arr2[j++];  
 }  
  
 return arr;  
}  
  
public static void printArr(int[] arr) {  
 for (int i = 0; i < arr.length; i++) {  
 System.*out*.print(arr[i] + " ");  
 }  
 System.*out*.println();  
}  
  
public static void main(String[] args) {  
 int[] arr1 = {1, 4, 5, 8, 17};  
 int[] arr2 = {2, 4, 8, 11, 13, 21, 23, 25};  
 *printArr*(*merge*(arr1, arr2));  
 *printArr*(*merge2*(arr1, arr2));  
}

## Prob3



## Prob4

public static List<Set<Integer>> powerSet(List<Integer> list) {  
  
 List<Set<Integer>> P = new ArrayList<>();  
 Set<Integer> S = new HashSet<>();  
 P.add(S);  
 if (list.isEmpty()) {  
 return P;  
 }  
  
 while (!list.isEmpty()) {  
 int f = list.remove(0);  
  
 List<Set<Integer>> temp = new ArrayList<>();  
  
 for (Set<Integer> x: P) {  
 temp.add(x);  
 }  
  
 for (Set<Integer> x: temp) {  
 Set<Integer> T = new HashSet<>();  
 T.addAll(x);  
 T.add(f);  
 P.add(T);  
 }  
 }  
  
 return P;  
  
}

## Prob5

for (Integer i: arr) {  
 if (!distinctList.contains(i)) {  
 distinctList.add(i);  
 }  
}

The **contains** method of List loop over the List and compare with each element to determine the List contain i or not so the run time is O(n2)

# Lab2 - Continue

## Prob1

Algorithm: RecursiveMNum(n)

Input: A positive integer n

Output: M(n) (defined above)

if(n = 1 or n = 2) then return n

return 2 \* RecursiveMNum (n - 1) \* RecursiveMNum(n - 2)

1. Verify that the recursion is valid: there should be a base case and recursive calls must eventually lead to the base case

- There is a base case: n = 1 or n = 2

- The recursive is eventually lead to base case: RecursiveMNum (n - 1), RecursiveMNum(n - 2)

2. Show that the values given by the base case are correct outputs for the function

Input: 1 => Output: 1

Input: 2 => Output: 2

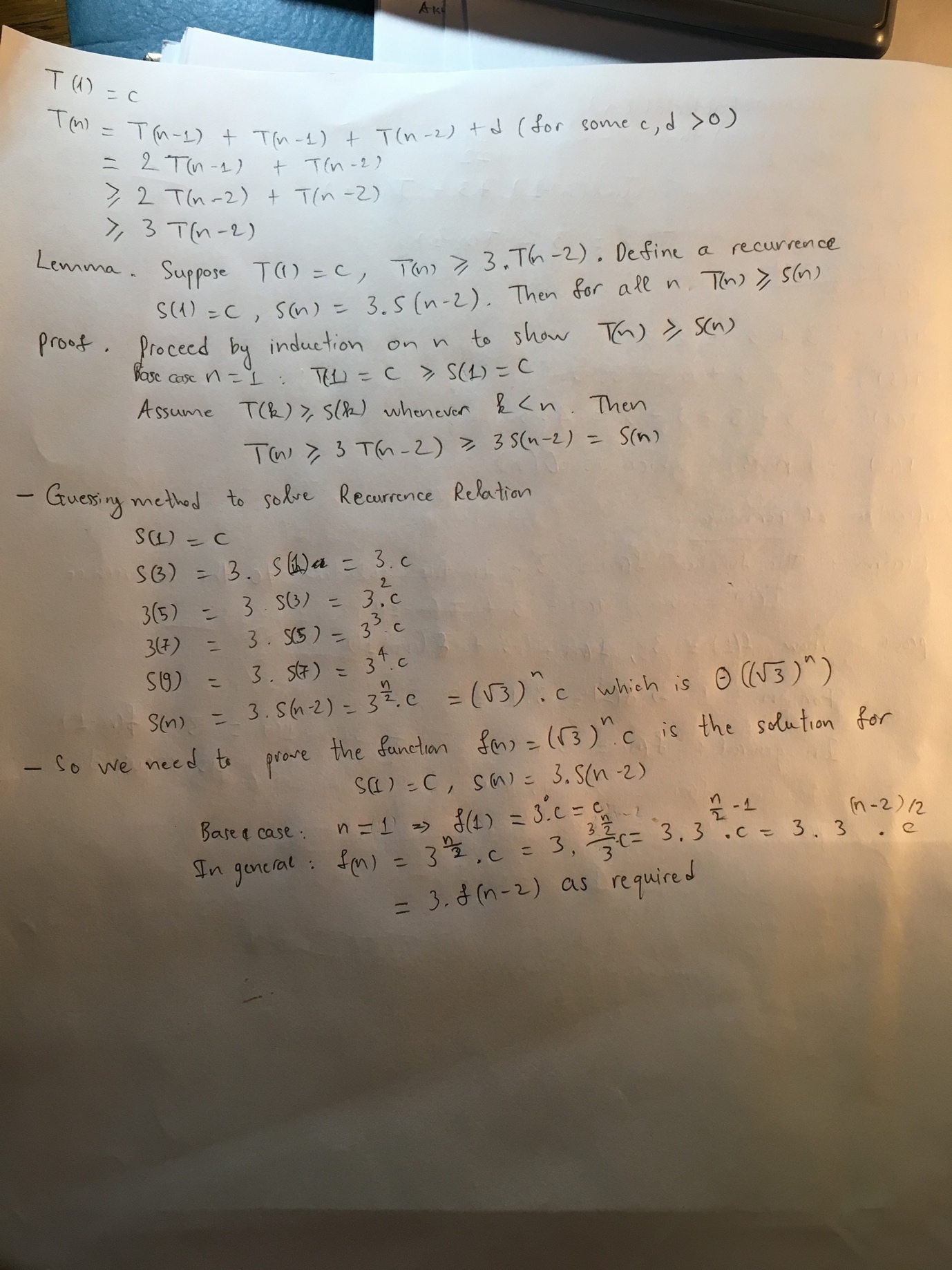
3. Show that, if you assume the output value of the algorithm on input j is correct, for all j < n, then output value on input n is correct.

Assume RecursiveMNum(j) is correct for every j < n

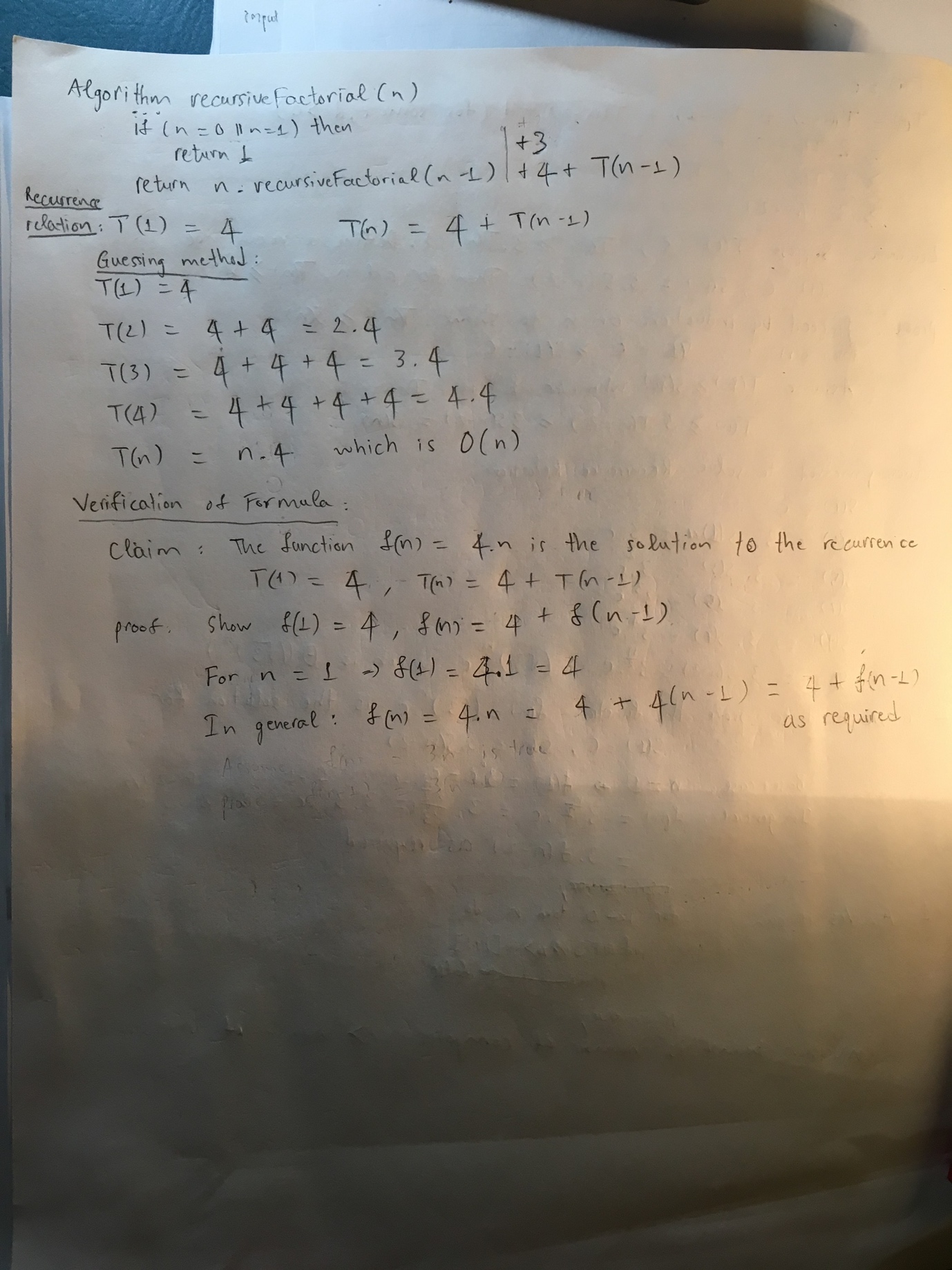
RecursiveMNum(n) = 2 \* RecursiveMNum (n - 1) \* RecursiveMNum(n - 2)

= 2 \* M(n-1) \* M(n-2) // n-1 < n & n-2 < 2

## Prob2



## Prob3

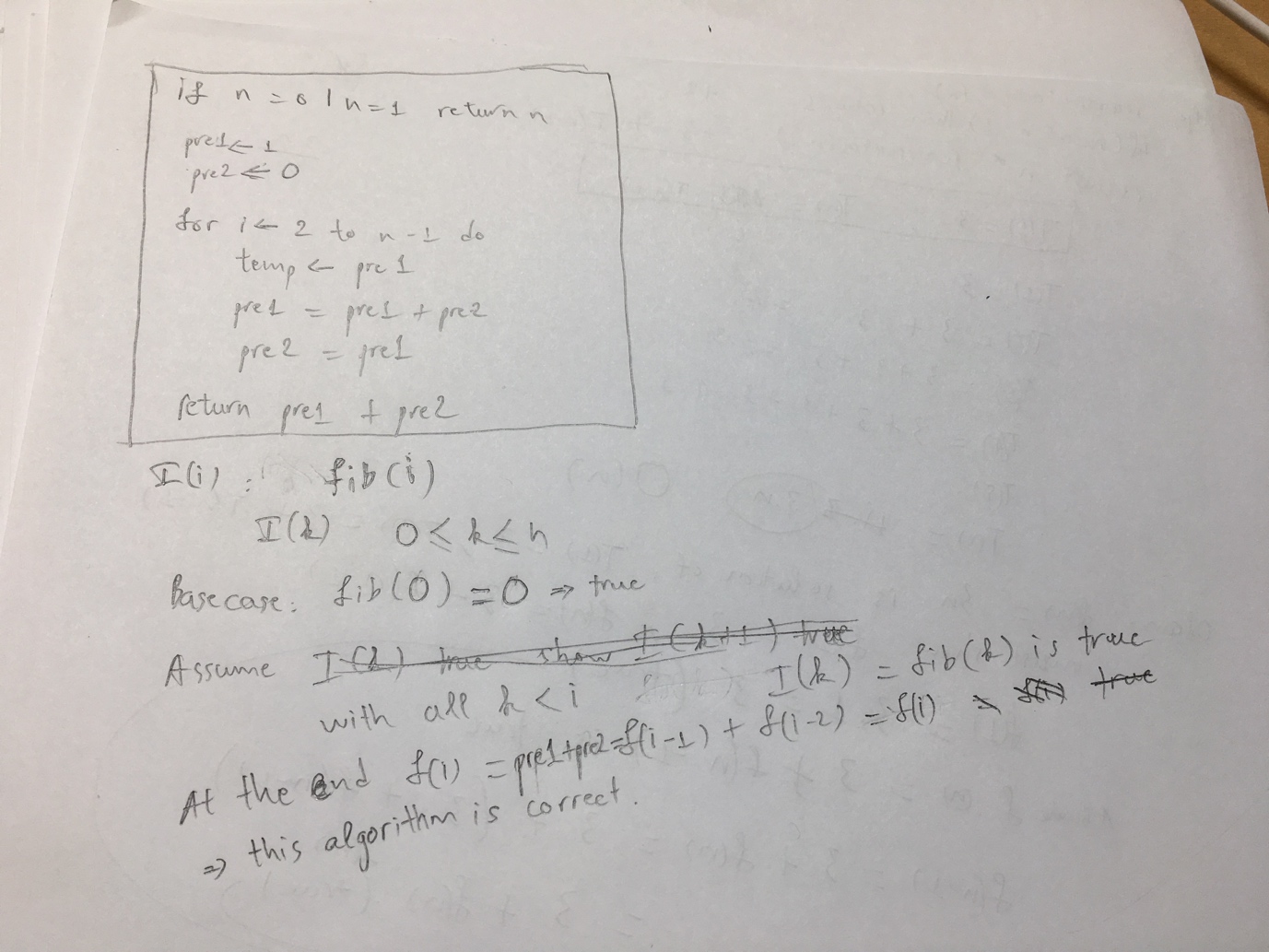


## Prob4

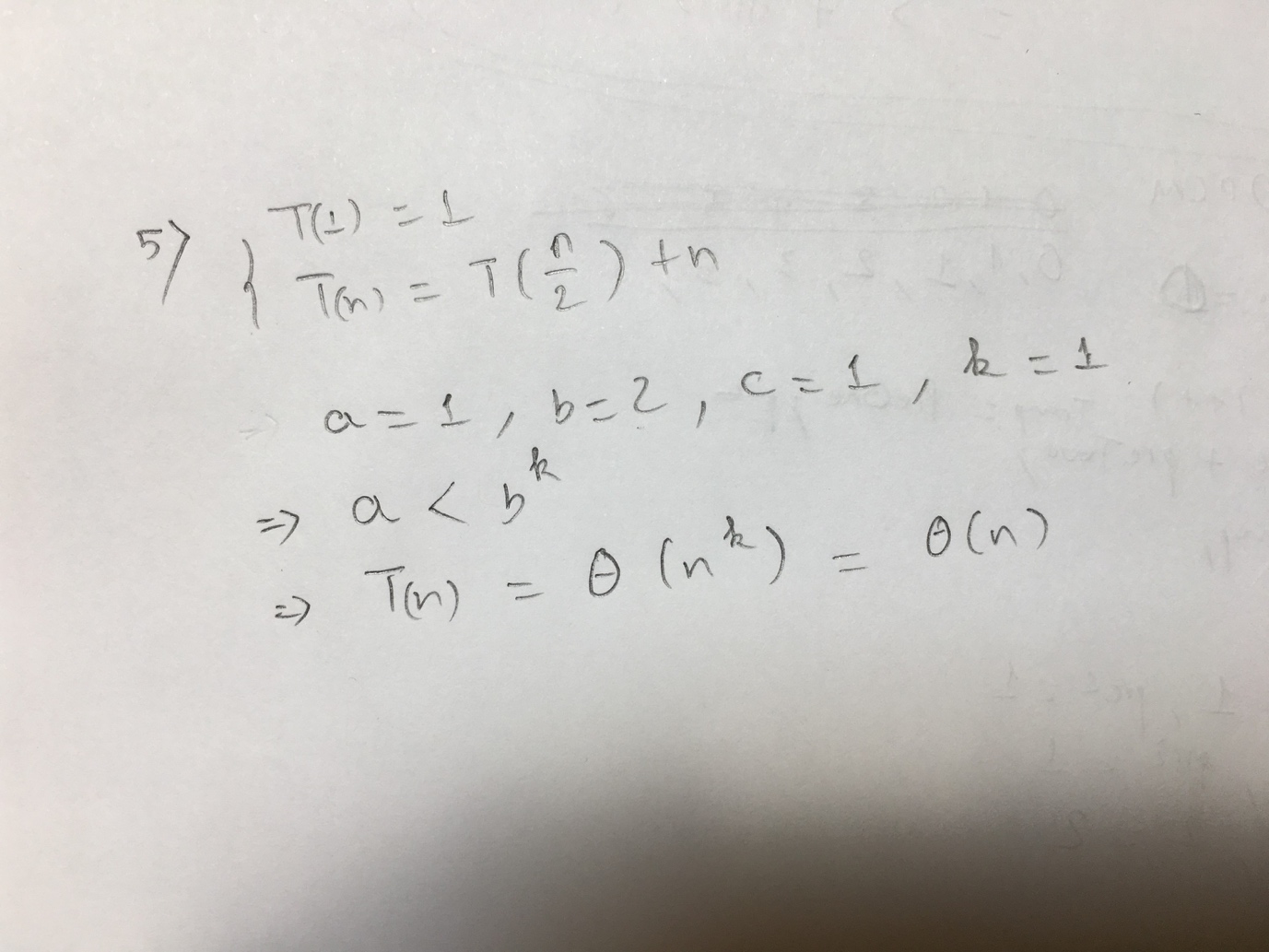
* Algorithms

Algorithm IterativeFib(n)  
 Input: A non-negative integer n  
 Output: fib(n)  
  
 if n = 0 || n = 1 then  
 return n  
  
 preOne <- 1  
 preTwo <- 0  
  
 for i <- 2 to n - 1 do  
 temp <- preOne  
 preOne <- preOne + preTwo  
 preTwo <- temp  
  
 return preOne + preTwo

* Running time is O(n)
* Prove correctness



## Prob5



## Prob6

static public int[] zeroAndOne(int[] A) {  
 int[] zeroAndOne = new int[2];  
  
 if (A.length == 0) {  
 zeroAndOne[0] = 0;  
 zeroAndOne[1] = 0;  
 return zeroAndOne;  
 }  
  
 if (A[0] == 1) {  
 zeroAndOne[0] = 0;  
 zeroAndOne[1] = A.length;  
 return zeroAndOne;  
 }  
  
 if (A[A.length - 1] == 0) {  
 zeroAndOne[0] = A.length;  
 zeroAndOne[1] = 0;  
 return zeroAndOne;  
 }  
  
 return *findBreakPoint*(A, 0, A.length - 1);  
}  
  
static public int[] findBreakPoint(int[] A, int bot, int top) {  
 int[] zeroAndOne = new int[2];  
  
 int mid = (bot + top) / 2;  
  
 if (A[mid] == 0 && A[mid + 1] == 1) {  
 zeroAndOne[0] = mid + 1;  
 zeroAndOne[1] = A.length - (mid + 1);  
 return zeroAndOne;  
 }  
  
 if (A[mid] == 0 && A[mid + 1] != 1) {  
 return *findBreakPoint*(A, mid + 1, top);  
 }  
  
 if (A[mid] == 1) {  
 return *findBreakPoint*(A, bot, mid - 1);  
 }  
  
 return null;  
}

This solution using logic of Binary Search so the run time is O(log n) (is o(n))