# 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

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

The run time of merge2 is O(n)

## 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)

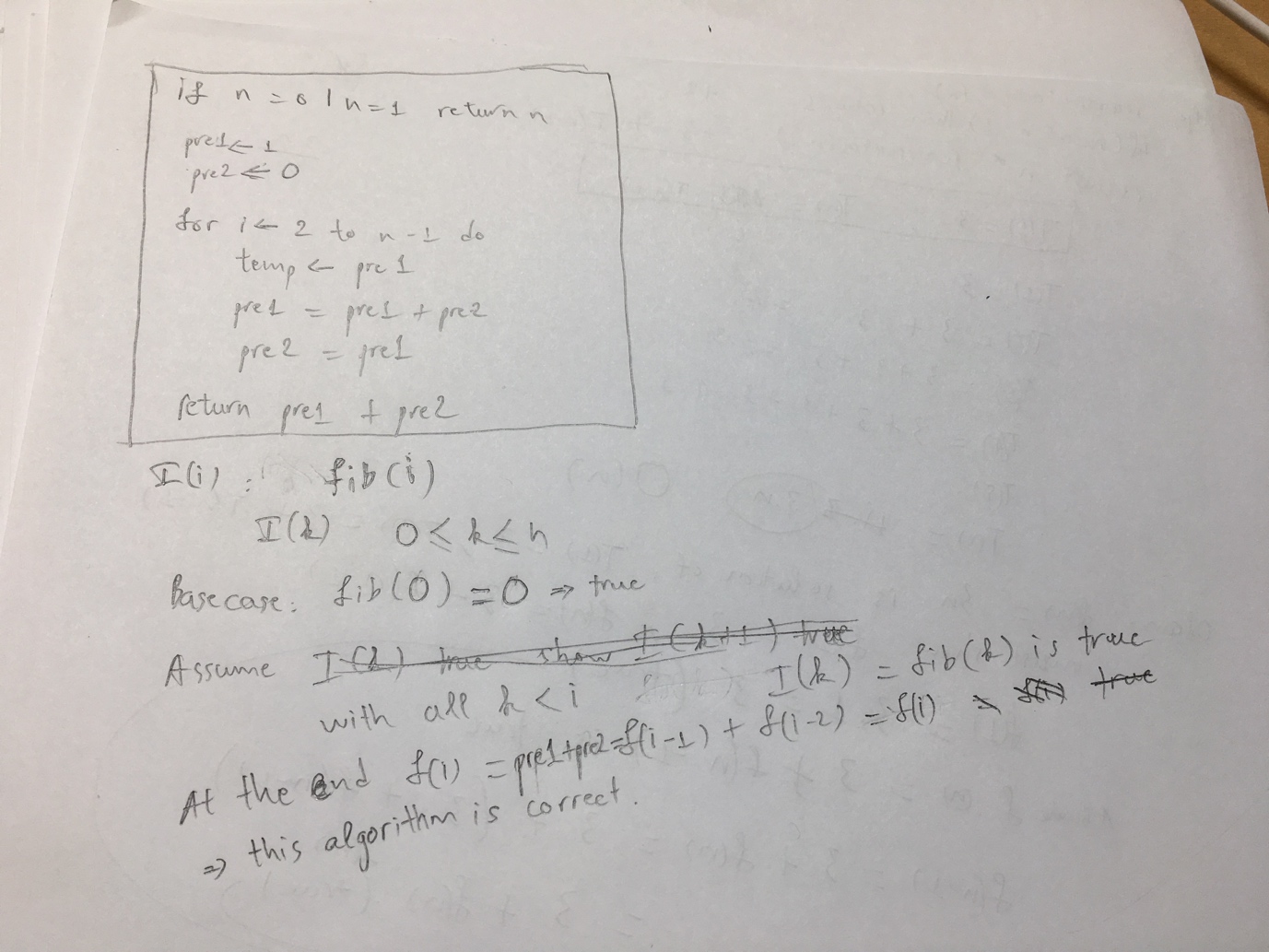
= 2 \* M(n-1) \* M(n-2) // n-1 < n & n-2 < 2

## 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

