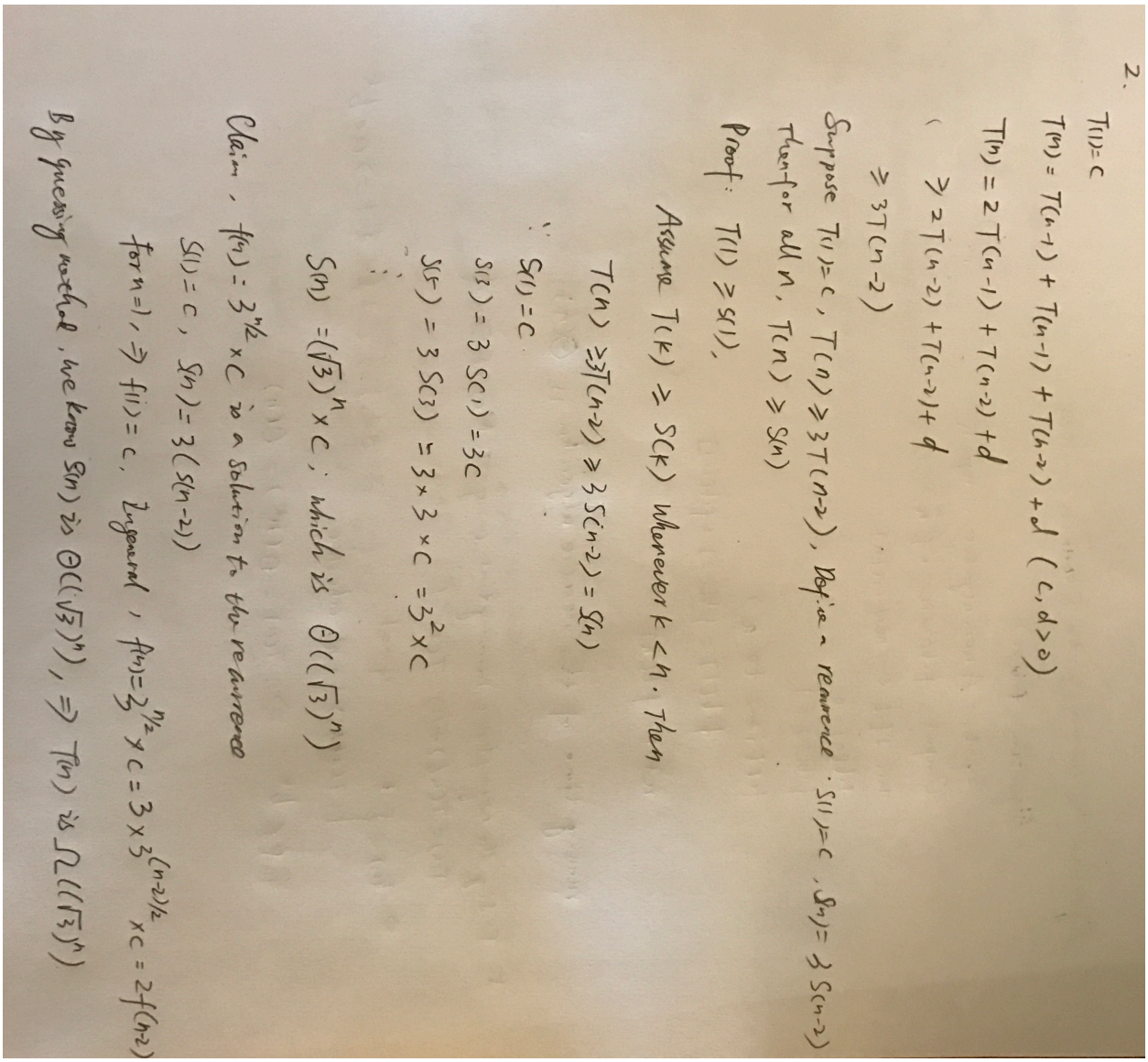
Problem 2

Assume the running time T(n) for a particular algorithm satisfies the following recurrence relation: T(1) = c T(n) = T(n-1) + T(n-1) + T(n-2) + d (for some c, d > 0) Use the technique of computing running time for the Fib algorithm discussed in class to solve the recurrence.



2.

$T(1) = c$

$T(n) = T(n-1) + T(n-1) + T(n-2) + d \quad (c, d > 0)$

$T(n) = 2\,T(n-1) + T(n-2) + d$

$\geq 2\,T(n-2) + T(n-2) + d$

$\geq 3\,T(n-2)$

Suppose $T(1) = c$, $T(n) \geq 3\,T(n-2)$, Define a recurrence $S(1) = c$, $S(n) = 3\,S(n-2)$ then for all n, $T(n) \geq S(n)$

Proof: $T(1) \geq S(1)$,

Assume $T(k) \geq S(k)$ whenever $k < n$. Then

$T(n) \geq 3T(n-2) \geq 3\,S(n-2) = S(n)$

$S(1) = c$

$S(3) = 3\,S(1) = 3c$

$S(5) = 3\,S(3) = 3 \times 3 \times c = 3^2 \times c$

$S(n) = (\sqrt{3})^n \times c$, which is $\Theta((\sqrt{3})^n)$

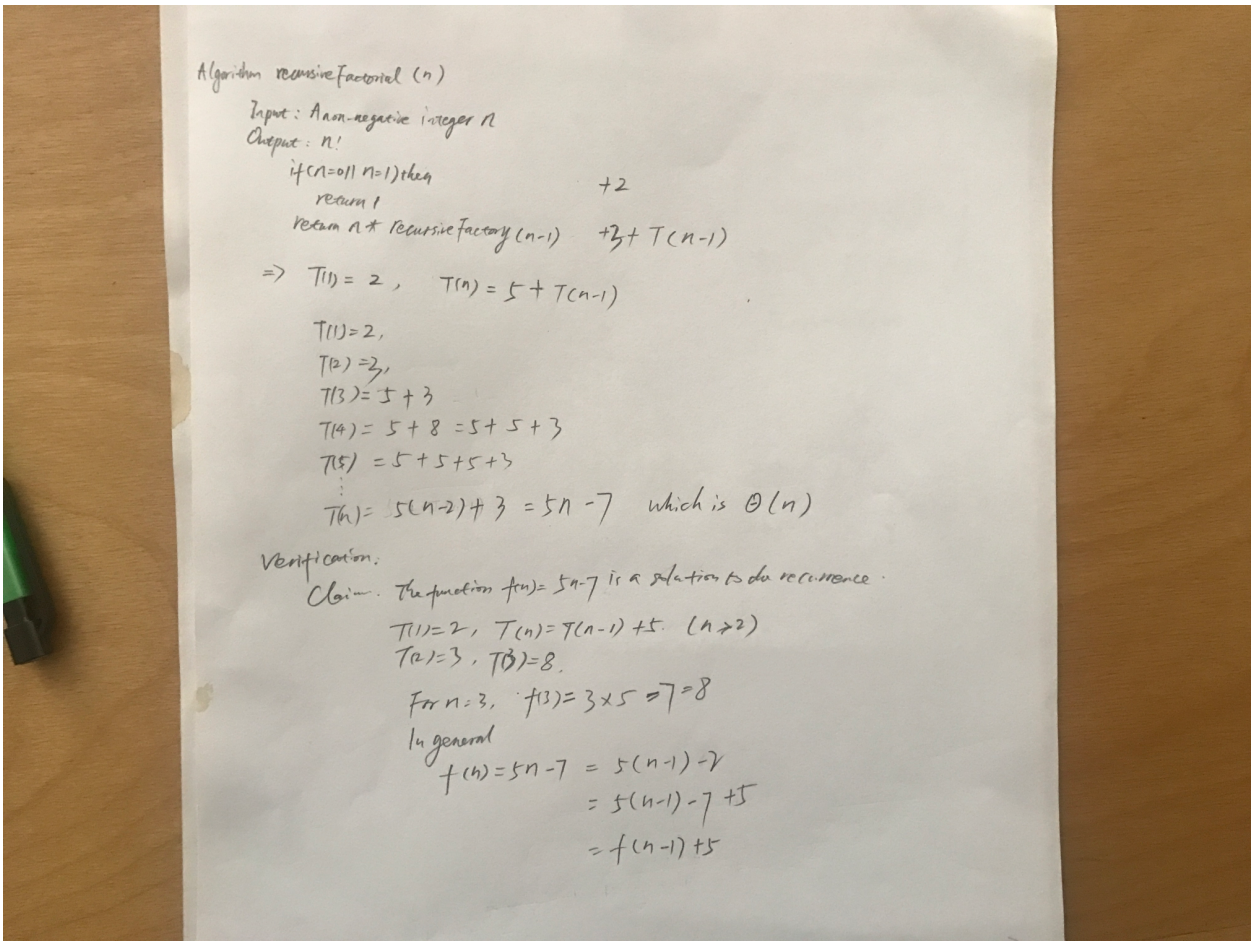
Claim, $f(n) = 3^{n/2} \times c$ is a solution to the recurrence

$S(1) = c$, $S(n) = 3(S(n-2))$

for $n = 1$, $\Rightarrow f(1) = c$, In general, $f(n) = 3^{n/2} \times c = 3 \times 3^{(n-2)/2} \times c = 3f(n-2)$

By guessing method, we know $S(n)$ is $\Theta((\sqrt{3})^n)$, $\Rightarrow T(n)$ is $\Omega((\sqrt{3})^n)$

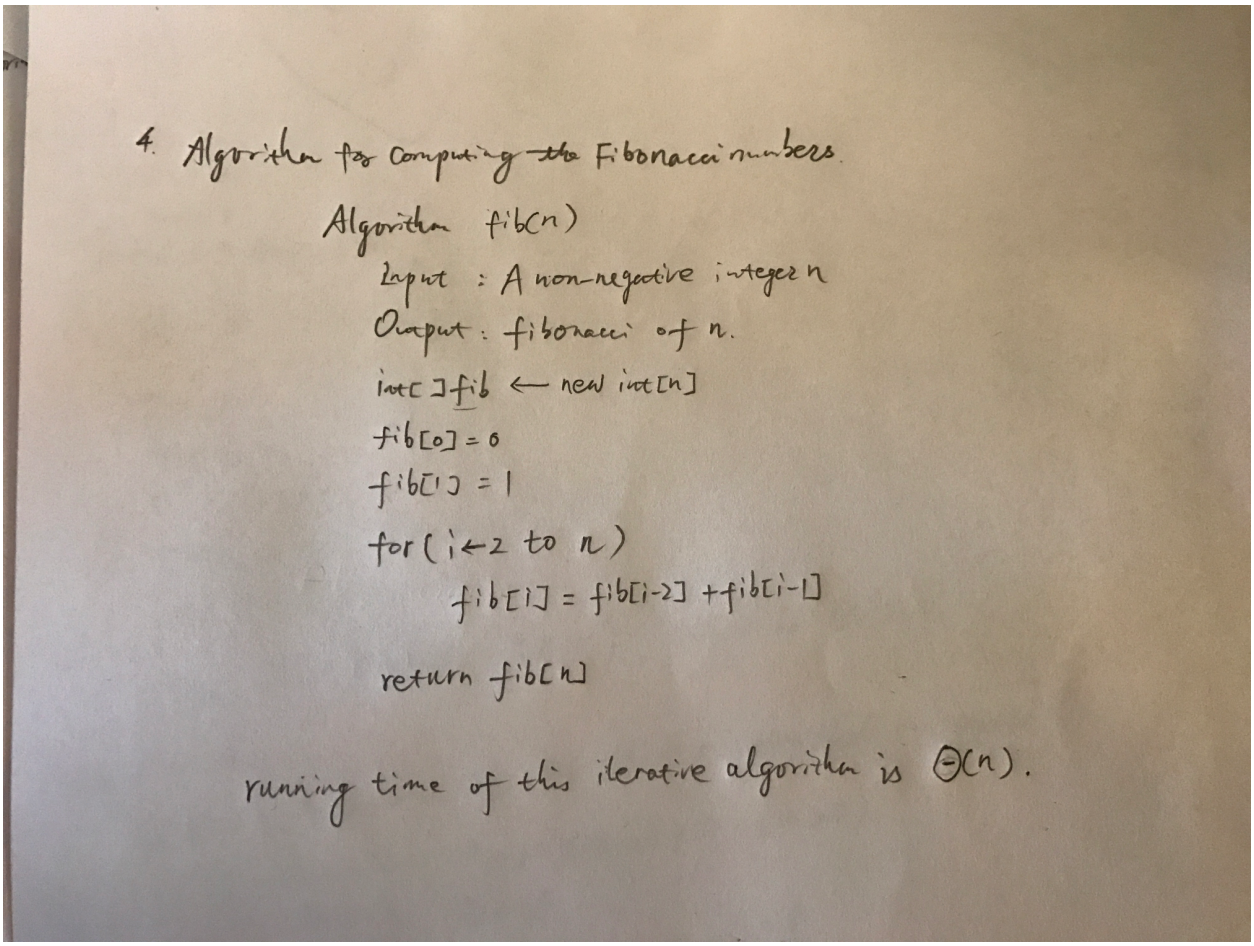
Problem 3



Problem 4

Devise an iterative algorithm for computing the Fibonacci numbers and compute its running

time.



4. Algorithm for Computing Fibonacci numbers.

```
Algorithm fib(n)
    Input : A non-negative integer n
    Output : fibonacci of n.
    int[] fib ← new int[n]
    fib[0] = 0
    fib[1] = 1
    for(i←2 to n)
        fib[i] = fib[i-2] + fib[i-1]
    return fib[n]
```

running time of this iterative algorithm is $\Theta(n)$.

5

F. Finding the asymptotic running time using the Master Formula:

$$T(n) = T(n/2) + n; \quad T(1) = 1$$

$$\Rightarrow T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 1 \cdot T(\lfloor n/2 \rfloor) + 1 \cdot n^1 & \text{otherwise} \end{cases} \quad \Rightarrow a=1, b=2, c=1, k=1, d=1 \Rightarrow a < b^k$$

$$\because a < b^k \Rightarrow T(n) = \Theta(n^k) = \Theta(n)$$

Problem 6

You are given a length-n array A consisting of 0s and 1s, arranged in sorted order. Give an o(n) algorithm that counts the total number of 0s and 1s in the array. Your algorithm may not make use of auxiliary storage such as arrays or hashtables (more precisely, the only additional space used, beyond the given array, is O(1)). You must give an argument to show that your algorithm runs in o(n) time. **Answer:**: Use the binary search to find the index of the first number 1, then the length of the array subtracted by the index is the number of the 1s. Time complexity is O(logn) which is also o(n).

```
public static int countNumber(int[] ints) {
    int len = ints.length;
    if(ints[0] == 1) return len;
    if(ints[len-1] == 0) return 0;
    int index = findFirstOne(ints, 0, ints.length);
    return ints.length - index;
  }

  private static int findFirstOne(int[] ints, int low, int high) {
    int mid = (low + high) / 2;
    if(ints[mid] == 1 && ints[mid-1] == 0) {
      return mid;
    } else if(ints[mid] == 1) {
      return findFirstOne(ints, low, mid -1);
    } else {
      return findFirstOne(ints, mid + 1, high);
    }
  }
```