**Problem 1**

F(n) = F(n-1) + F(n-2)

F (n) = F(n - 1) + F(n -2)

>= 2\*F(n - 2) = 2\*( F( n - 3) + F(n-4) )

>= 2\*2\*F(n-4) = 2^2\*F(n-4)

>= 2^3\*F(n-6)

>= 2^(n/2)\*F(0) = 2^(n/2) > (3/4)^n

**Problem 2**

a. False:

b. True:

c. True:

**Problem 3**

* Formulate the recurrence relation. when n = 0

T(n) = 4 if n = 1,T(n) = T(n − 1) + 4 if n > 1

* Calculate value from 1 to n.

T(1) = 4

T(2) = T(1) + 4 = 4 + 4

T(3) = T(2) + 4 = 4 + 4 + 4

T(n) = T(n − 1) + 4 = 4n.

* Prove the formula above.

f(n) = 4n.

f(1) = 4 and f(n) = f(n − 1) + 4. The first part is obviously true.

And f(n) = 4n = 4(n − 1) + 4 = **f(n − 1) + 4**.

* Prove correctness.
  + *Verify valid recursion.* n == 0 || n == 1 is the base case and repeated self- call reduces the input size by 1.
  + *Verify base case outputs are correct*. 0! = 1 and 1! = 1.
  + *Verify inductively that outputs are correct for all n.* recursiveFactorial(j) outputs j! for all j < n, where n > 1 and (n-1)! \* n = n!

**Problem 4:**

**int** fibonacci(**int** n) {

**int** f0 = 0;

**int** f1 = 1;

**if** (n == 0) {

**return** f0;

}

**if** (n == 1) {

**return** f1;

}

**for** (**int** i = 1; i < n; i++) {

f1 = f0 + f1;

f0 = f1 - f0;

}

**return** f1;

}

The running time is: O(n)

Proof:

* The base case: n = 0 or n = 1, it is correct
* For case k: f1 is computed to new value equal to previous f0, f1 and f0 receive previous value of f1, the assignment running k times and f(k) = f1
* For case k = n: f(n) = f1

**Problem 5**

a = 1, b = 2, c = 1, d = 1, k = 1 (a < bk)=> The Master Formula T(n) (n).

**Problem 6**

Count 0 and 1

* Input: a length-n array A consisting of 0s and 1s, arranged in sorted order
* Output: total number of 0s and 1s in the array

Array count\_0\_1(Array arr) {

c0 = 0

c1 = 0

size <- length of arr

i <- 0

for i = 0 to size do {

if arr[i] == 1 then {

break

}

}

c0 <- i

If (i != size) {

c1 <- size – i

}

return { c0, c1 };

}

**Proof:**

|  |  |
| --- | --- |
| Algorithm | Operations |
| Array count\_0\_1(Array arr) {  c0 = 0  c1 = 0  size <- length of arr  i <- 0  for i = 0 to size do {  if arr[i] == 1 then {  break  }  }  c0 <- i  If (i != size) {  c1 <- size – i  }  return { c0, c1 };  } |  |
| 1 |
| 1 |
| 1 |
| 1 |
|  |
| n |
| 2 |
| 1 |
|  |
|  |
|  |
| 1 |
| 1 |
| 2 |
|  |
|  |
| 2 |
|  |
|  |
| **Summary:** Operations = n + 13 🡪 Algorithm runs in O(n) time | |