
*On the cover page of your assignment, you must list everyone with whom you discussed this assignment, and which problems you discussed with each person. You must also write **and sign** the following statement: “I have read and understood the policy on collaboration on homework assignments stated in the Course Information handout.” Without these, your homework will not be marked.*

In the first part of the assignment, we will consider calculating the elements of the Fibonacci sequence discussed in class using recursive procedures. Recall that the Fibonacci function is the function F that satisfies

$$F(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F(n-1) + F(n-2) & \text{if } n \geq 2, n \in \mathbb{N} \end{cases}$$

The most obvious way of calculating the elements of this sequence is using a recursive procedure like the following:

```
PROC fib1(n)
  IF n < 2 THEN RETURN n
  ELSE RETURN fib1(n-1) + fib1(n-2)
END
END
```

A less obvious recursive procedure is the following:

```
PROC fib2(n)
  RETURN g(n,1,0)
END
PROC g(n,x,y)
  IF n == 0 THEN RETURN y
  ELSE RETURN g(n-1,x+y,x)
END
END
```

The fact that both procedures compute the same thing would be a mere curiosity except for the following: fib_2 is extremely efficient, while fib_1 is horribly inefficient. From what you will prove below, for example, $fib_2(1000)$ would run to completion in a fraction of a second, but $fib_1(1000)$ would take over 10^{100} years on any computer!

1. Prove that fib_2 and fib_1 compute the same thing. More precisely, prove that for all $n \in \mathbb{N}$, $fib_2(n)$ terminates and returns $F(n)$ as defined above. *Hint:* First prove a lemma by induction on n that for every $x, y, n \in \mathbb{N}$, such that $n \geq 1$, the procedure $g(n, x, y)$ terminates and returns $x \cdot F(n) + y \cdot F(n-1)$. You do *not* need to use the closed-form solution for F to prove this, only the recurrence property of F above. [10]

2. Let $T_2(n)$ stand for the number of additions and subtractions performed by $g(n, x, y)$, and thus by $fib_2(n)$. Write a recurrence relation for T_2 , guess at a closed-form solution for it, and prove it correct. It will then follow that $T_2 \in \mathcal{O}(n)$. [10]
3. Let $T_1(n)$ stand for the number of additions and subtractions performed by $fib_1(n)$. Write a recurrence relation for T_1 , and without trying to solve it, prove that for $n \geq 2$, $T_1(n) \geq 2^{n/2}$. It then follows that $T_1 \in \Omega(2^{n/2})$. [10]
4. Consider the following recurrence relation: [5]

$$T(n) = \begin{cases} 0 & \text{if } n = 0 \\ 0 & \text{if } n = 1 \\ 2 \cdot T(n - 2) + 1 & \text{if } n \geq 2, n \in \mathbb{N} \end{cases}$$

(This is a simplified variant of the recurrence for T_1 above.) Use the method of repeated substitutions to derive a closed-form solution for $T(n)$. You may use different expressions for the odd and even cases of n , or handle both cases with a single expression using “floor” or “ceiling.” *Hint:* In eliminating the T from the RHS of the equation, there will be two ways of doing this, using the values of $T(0)$ and $T(1)$, and leading to the odd and even cases for n .

You do not need to prove your solution correct. (But you should perform some sort of sanity check to convince yourself that your solution works.)

In the second part of the assignment, we will consider the problem of calculating the average of an array A of m positive numbers. The obvious way to do this is to sum all the numbers and divide by m . But suppose m is large and some of the elements of A are also large. We may be concerned about *overflow*. Although the average will never be larger than the largest element of A , we might prefer not to calculate the sum or any intermediate result that is bigger than this largest element. Here is one way of doing so:

Precondition: $m \in \mathbb{N}$ $m \geq 1$, array $A[1..m]$, where $A[i] \in \mathbb{R}^+$.

Postcondition: $u = \frac{1}{m} \sum_{i=1}^m A[i]$.

```

c := 0
u := 0.0
WHILE c != m DO
    c := c+1
    u := u*((c-1)/c) + A[c]/c
END

```

Note that we are not concerned about roundoff errors in this example. We assume that all the necessary arithmetic operations are exact.

5. Prove that the following is a loop invariant of this program: [10]

$$c \cdot u = \sum_{i=1}^c A[i].$$

The partial correctness of the program is then immediate since $c = m$ on termination. Termination of the program is easy to see since the program does precisely m iterations.

6. Suppose we had written the assignment to u in the body of the WHILE loop as follows: $u := (u \cdot (c-1))/c + A[c]/c$. The program is still correct, but it is no longer suitable for our purposes. Why not? Do some algebra and simplify the assignment to u to get a statement that is both correct and suitable, but uses only 1 addition, 1 subtraction, 1 division and no multiplications. [5]

One disadvantage of the proposed iterative program above is that it requires $\Theta(m)$ divisions, even with the optimization mentioned in the previous question. We might instead try a “divide and conquer” approach, like the following program:

```

PROC avg(k,n)
  IF n == 1 THEN RETURN A[k]
  ELSE g := n DIV 2
        RETURN avg(k,g)*(g/n) + avg(k+g,n-g)*((n-g)/n)
  END
END
/* main */
u := avg(1,m)

```

7. Prove that this program is correct for the same precondition and postcondition as the iterative program. *Hint:* First prove by complete induction on n that for all $k, n \in \mathbb{N}$ such that $n \geq 1$, $k \geq 1$, and $(k + n - 1) \leq m$, the procedure $avg(k, n)$ terminates and returns a number u such that [10]

$$nu = \sum_{i=k}^{k+n-1} A[i].$$

So $avg(k, n)$ returns the average of n elements of the array, starting at index k .

8. Suppose the IF statement in this program had been `IF n == 0 THEN RETURN 0` with the rest of the program the same. Would this work? If so, explain how your proof of correctness needs to change to handle all $n \geq 0$; if not, explain why your proof cannot be fixed. [5]
9. Does this recursive procedure do better with divisions than the iterative one? Write a recurrence relation for the number of divisions performed by $avg(k, n)$, and use the Divide and Conquer Theorem to justify your answer. *Hint:* For any $n \in \mathbb{N}$, $n \geq 2$, $(n - \lfloor n/2 \rfloor) = \lceil n/2 \rceil$. [5]