Analysis Of Algorithms: Homework #0

Enrique Ortiz Macias

Due 2020/09/15

1 Using the Fibonacci definition prove by induction that

$$F_n \geq 2^{0.5n} \text{ for } n \geq 6.$$

Given the Fibonacci definition:

$$F_i = \begin{cases} i \leq 1 & i \\ i > 1 & F_{i-2} + F_{i-1} \end{cases}$$

for $i \in \mathbb{N}_0$.

We can express $2^{0.5n}$ as $\sqrt{2^n}$.

1.1 Base Cases

Since the Fibonacci definition needs at least two smaller well-defined cases, we will use the two smallest cases F_n and F_{n+1} where $n = 6$, the smallest possible input.

1.1.1 Base case #1: $F_n \geq \sqrt{2^n}$.

$F_n \geq \sqrt{2^n}$ with $n = 6$ gives us:

$$F_6 \geq \sqrt{2^6}$$

By applying the Fibonacci definition, we have that $F_6 = 8$.

To prove $8 \geq \sqrt{2^6}$, we do:

$$\begin{aligned} 8 &\geq \sqrt{2^6} \\ 8 &\geq \sqrt{64} \\ 8 &\geq 8 \end{aligned}$$

1.1.2 Base case #2: $F_{n+1} \geq \sqrt{2^{n+1}}$.

We can express $F_{n+1}$with $n = 6$ as F_n with $n = 7$.

$F_n \geq \sqrt{2^n}$ with $n = 7$ gives us:

$$F_7 \geq \sqrt{2^7}$$

By applying the Fibonacci definition, we have that $F_7 = 13$.
To prove $13 \geq \sqrt{2^7}$, we do:

$$\begin{array}{rcl} 13 & \geq & \sqrt{2^7} \\ 13 & \geq & \sqrt{128} \\ 13 & \geq & 11.3 \end{array}$$

1.2 Induction Step

Having proven the statement true for our base cases F_n and F_{n+1}, we can hypothesize the outcome with F_{n+2}.

Our induction hypothesis is:

$$F_{n+2} \geq \sqrt{2^{n+2}}$$

We can express $\sqrt{2^{n+2}}$ as $2\sqrt{2^n}$.

We can use the Fibonacci definition to develop F_{n+2} as follows:

$$\begin{array}{rcccc} F_{n+2} & = & F_n & + & F_{n+1} \\ F_{n+2} & = & \sqrt{2^n} & + & \sqrt{2^{n+1}} \\ F_{n+2} & = & \sqrt{2^n} & + & \sqrt{2^n \times 2} \\ F_{n+2} & = & \sqrt{2^n} \times 1 & + & \sqrt{2^n} \times \sqrt{2} \\ F_{n+2} & = & \sqrt{2^n} & \times & \left(1 + \sqrt{2}\right) \\ F_{n+2} & = & \sqrt{2^n} & \times & (1 + 1.41) \\ F_{n+2} & = & \sqrt{2^n} & \times & 2.41 \\ F_{n+2} & = & 2.41\sqrt{2^n} & & \end{array}$$

And so, it is proven that:

$$\begin{array}{rcl} F_{n+2} & \geq & 2\sqrt{2^n} \\ 2.41\sqrt{2^n} & \geq & 2\sqrt{2^n} \end{array}$$

Q.E.D.

2 Prove the correctness of Euclid's Algorithm.

2.1 Algorithm Explanation

The Euclidian Algorithm for finding the G.C.D. (Greatest Common Divisor) of two numbers needs the following setup:

Given two numbers k_0 and k_1 such that $k_0 \geq k_1$, we calculate the remainder of the division $k_0 \, mod \, k_1 = k_2$.

Since $k_0 \geq k_1 \geq k_2$, we can repeat the process using $k_1$and k_2, finding the remainder k_3.

This scenario holds for $n-1$ iterations until $k_{n-2} \, mod \, k_{n-1} = k_n = 0$.

And so we can observe the principle:

$$gcd\,(k_0, k_1) = gcd\,(k_1, k_2)$$

Which holds in a general form:

$$gcd\,(k_i, k_{i+1}) = gcd\,(k_{i+1}, k_{i+2})$$

And after running the above procedure $n-1$ times and finding $k_n = 0$:

$$gcd\,(k_0, k_1) = gcd\,(k_{n-1}, k_n)$$

At that point, we have found the G.C.D.:

$$gcd\,(k_0, k_1) = k_{n-1}$$

2.2 Proof of the Principle

We will express the aforementioned equation in two different forms:

$$\begin{array}{ccccccccc} A) & k_i & - & k_{i+1} & \times & q_i & = & k_{i+2} \\ B) & k_i & = & k_{i+1} & \times & q_i & + & k_{i+2} \end{array}$$

Where q is the quotient of the division $k_i \div k_{i+1}$ and k_{i+2} is its remainder.

Let d_0 be any common divisor of k_i and k_{i+1}.
Therefore, considering equation A:

$$d_0|k_i \quad , \quad d_0|k_{i+1} \quad , \quad d_0|\,(k_i - k_{i+1} \times q_i) \quad , \quad d_0|k_{i+2}$$

Let d_1 be any common divisor of k_{i+1} and k_{i+2}.
Therefore, considering equation B:

$$d_1|k_{i+1} \quad , \quad d_1|k_{i+2} \quad , \quad d_1|\,(k_{i+1} \times q_i + k_{i+2}) \quad , \quad d_1|k_i$$

Which means that any common divisor of k_i and k_{i+1} must also divide k_{i+2}.

And also, any common divisor of k_{i+1} and k_{i+2} must also divide k_i.

Thus, we can prove the principle by stating:

$$\big(d|k_i \quad \wedge \quad d|k_{i+1}\big) \quad \iff \quad \big(d|k_{i+1} \quad \wedge \quad d|k_{i+2}\big)$$

Which means that any two pairs of contiguous terms in the sequence $k_i ... k_n$ must have the same G.C.D.

Q.E.D.

3 Move a list structure to C using ctypes.

3.1 Generate a list reader of integers to C using ctypes.

Python 3 listreader.py

```
import os
import ctypes

(...)

def listOperations():

(...)

    libc = ctypes.CDLL("./listreader.so")

    #wrap C's listReader function and set up its arg/return types
    listReader = wrap_function(
        libc,
        "listReader",
        None,
        [ctypes.POINTER(ctypes.c_int), ctypes.c_int]
    )

    the_list = [0, 1, 1, 2, 3, 5, 8, 13, 21] #python vanilla list

    #generate a sequence of c_int's,
    #its size will be the length of the_list
    cint_sequence = ctypes.c_int * len(the_list)

    #call c_int (it's a constructor)
    #pass the_list as arg
    #the whole list (the_list) is passed via varargs (*)
    cint_array = cint_sequence(*the_list)

    print("\nWELCOME TO THE LIST READER!\n")
```

```

        print("This is the list in Python!\n", the_list, "\n")
        listReader(cint_array, len(the_list)) #pass the list and its length
    #listOperations

(...)In addition, the rest of the listReader function is of constant-time (lines

    #eof
```

3.2 Generate the corresponding .h and .c files.

C listreader.h

```
    #ifndef LISTREADER_FILE
    #define LISTREADER_FILE

        /*listreader.h*/

        void listReader(int* intlist, int listlen);
    #endif

    //eof
```

C listreader.c

```
    //listreader.c

    #include <stdio.h>
    #include <string.h>
    #include <stdlib.h>
    #include "listreader.h"

    void listReader(int* intlist, int listlen) {

        //read and print the whole list!

        int i;
        printf("This is the list in C!\n[");
        for(i=0; i<listlen; i++) {
            printf(" %d ", intlist[i]);
        }
        printf("]\n\nHave a nice day! bye!\n");
    }

    //eof
```

3.3 What is the complexity of your reader?

As seen in the code of listreader.c, the reader traverses the integer array, reading (and printing) one character at a time.

Reading and printing is a constant-time operation (line 15).

The array traversal is accomplished through a for loop. This loop iterates n times, where n is the length of the array.

The for loop header (line 14) is run one more time, when the loop breaks.

Therefore, the assignment and increment of i and the evaluation of the loop condition are all constant-time operations which run $n + 1$ times.

We can therefore bound the complexity of the reader as $O(n)$.

As a side note, there is a similar operation in the Python code (listreader.py, line 55) which does the same in theory, but should be several times slower in practice, due to the complexity of the Python List as compared to the simple C Array.