Math 374

Homework 2

Name: Gianluca Crescenzo

(i) The following code "**fib.py**" determines the largest Fibonacci number which can be represented by an unsigned nibble, byte, short, int, and long. It also creates a table measuring the speed of one run of **largestFib()**, and the average time of 100000 runs of **largestFib()**.

```python
import time
from tabulate import tabulate

def largestFib(k):
    if k <= 2:
        return 0, 0
    a, b = 1, 1
    count = 2
    while b < k:
        a, b = b, a + b
        count += 1
    return a, count - 1

def time_fib(val):
    start = time.perf_counter()
    fib_val, fib_count = largestFib(val)
    end = time.perf_counter()
    total_time = end - start
    return fib_val, fib_count, total_time

print("")
fib_val_1, fib_count_1, total_time_1 = time_fib(2**4)
print(f"F({fib_count_1}) = {fib_val_1} < 2^4")
fib_val_2, fib_count_2, total_time_2 = time_fib(2**8)
print(f"F({fib_count_2}) = {fib_val_2} < 2^8")
fib_val_3, fib_count_3, total_time_3 = time_fib(2**16)
print(f"F({fib_count_3}) = {fib_val_3} < 2^16")
fib_val_4, fib_count_4, total_time_4 = time_fib(2**32)
print(f"F({fib_count_4}) = {fib_val_4} < 2^32")
fib_val_5, fib_count_5, total_time_5 = time_fib(2**64)
print(f"F({fib_count_5}) = {fib_val_5} < 2^64")

print("")
data1 = [
    ["Nibble",              4,  fib_val_1,  total_time_1],
    ["Byte",                8,  fib_val_2, total_time_2],
    ["Unsigned short int",  16, fib_val_3, total_time_3],
    ["Unsigned int",        32, fib_val_4, total_time_4],
    ["Unsigned long",       64, fib_val_5, total_time_5],
]
header1 = ["Data Type", "n bits", "F(n)", "Computation Time"]
print(tabulate(data1, headers=header1, tablefmt="grid"))

def time_fib_avg(val, runs=100000):
    start = time.perf_counter()
    fib_val = fib_count = 0
    for _ in range(runs):
        fib_val, fib_count = largestFib(val)
    end = time.perf_counter()
    total_time = end - start
    avg_time = total_time / runs
    return fib_val, fib_count, total_time, avg_time

fib_val_1, fib_count_1, total_run_time_1, avg_time_1 = time_fib_avg(2**4, 100000)
fib_val_2, fib_count_2, total_run_time_2, avg_time_2 = time_fib_avg(2**8, 100000)
fib_val_3, fib_count_3, total_run_time_3, avg_time_3 = time_fib_avg(2**16, 100000)
fib_val_4, fib_count_4, total_run_time_4, avg_time_4 = time_fib_avg(2**32, 100000)
fib_val_5, fib_count_5, total_run_time_5, avg_time_5 = time_fib_avg(2**64, 100000)

print("")
data2 = [
    ["Nibble",              4,  fib_val_1,  total_run_time_1, avg_time_1],
    ["Byte",                8,  fib_val_2, total_run_time_2, avg_time_2],
    ["Unsigned short int",  16, fib_val_3, total_run_time_3, avg_time_3],
    ["Unsigned int",        32, fib_val_4, total_run_time_4, avg_time_4],
    ["Unsigned long",       64, fib_val_5, total_run_time_5, avg_time_5],
```

```python
]
header2 = ["Data Type", "n bits", "F(n)", "Total time (100000 runs)", "Avg time per run"]
print(tabulate(data2, headers=header2, tablefmt="grid"))

```

```
(base) gcrescenzo@EigenMac LaTeX Documents % /opt/homebrew/bin/python3 "/Users/gcrescenzo/Documents/School
/LaTeX Documents/Class Notes/Spring 2025/Computational Math/374 Homework 2/fib.py"

F(7) = 13 < 2^4
F(13) = 233 < 2^8
F(24) = 46368 < 2^16
F(47) = 2971215073 < 2^32
F(93) = 12200160415121876738 < 2^64

+--------------------+----------+----------------------+--------------------+
| Data Type          |   n bits |                 F(n) |   Computation Time |
+====================+==========+======================+====================+
| Nibble             |        4 |                   13 |        1.375e-06   |
+--------------------+----------+----------------------+--------------------+
| Byte               |        8 |                  233 |        9.57996e-07 |
+--------------------+----------+----------------------+--------------------+
| Unsigned short int |       16 |                46368 |        1.333e-06   |
+--------------------+----------+----------------------+--------------------+
| Unsigned int       |       32 |           2971215073 |        2.625e-06   |
+--------------------+----------+----------------------+--------------------+
| Unsigned long      |       64 | 12200160415121876738 |        3.75e-06    |
+--------------------+----------+----------------------+--------------------+

+--------------------+----------+----------------------+----------------------------+--------------------+
| Data Type          |   n bits |                 F(n) |   Total time (100000 runs) |   Avg time per run |
+====================+==========+======================+============================+====================+
| Nibble             |        4 |                   13 |                  0.0185736 |        1.85736e-07 |
+--------------------+----------+----------------------+----------------------------+--------------------+
| Byte               |        8 |                  233 |                  0.0266018 |        2.66018e-07 |
+--------------------+----------+----------------------+----------------------------+--------------------+
| Unsigned short int |       16 |                46368 |                  0.0525207 |        5.25207e-07 |
+--------------------+----------+----------------------+----------------------------+--------------------+
| Unsigned int       |       32 |           2971215073 |                  0.123875  |        1.23875e-06 |
+--------------------+----------+----------------------+----------------------------+--------------------+
| Unsigned long      |       64 | 12200160415121876738 |                  0.254366  |        2.54366e-06 |
+--------------------+----------+----------------------+----------------------------+--------------------+
(base) gcrescenzo@EigenMac LaTeX Documents %
```

We can see that for each data type, the Fibonacci index increases by approximately a factor of 2. Also, I believe the second table gives a better measurement of the time it takes to run `largestFib()`. The iterative algorithm for computing Fibonacci numbers seems very efficient, so any variation in run-time will be incredibly small, but still measurable. For 1000 and 10000 runs, each execution of `fib.py` gave drastically different results for the average time per run. It wasn't until I increased the amount of runs to 100000 before I saw consistent results between executions.

(ii) I used `AbsoluteTiming` to measure the time it takes to find the given Fibonacci numbers.

```
In[1]:= ClearSystemCache[]
        Part[AbsoluteTiming[Fibonacci[10]], 1]
        Part[AbsoluteTiming[Fibonacci[100]], 1]
        Part[AbsoluteTiming[Fibonacci[1000]], 1]
        Part[AbsoluteTiming[Fibonacci[10000]], 1]
        Part[AbsoluteTiming[Fibonacci[100000]], 1]
        Part[AbsoluteTiming[Fibonacci[1000000]], 1]
        Part[AbsoluteTiming[Fibonacci[10000000]], 1]
        Part[AbsoluteTiming[Fibonacci[100000000]], 1]

Out[1]= 0.000002

Out[2]= 0.000027

Out[3]= 0.000027

Out[4]= 0.00003

Out[5]= 0.000172

Out[6]= 0.002766

Out[7]= 0.029873

Out[8]= 0.355072
```

I believe Mathematica is more efficient at computing larger values of Fibonacci numbers. Mathematica could be storing each previous value in my computer's memory, or have a hardcoded table of values.

(iii) Given the explicitly defined functions $T : \mathbf{Z}_+ \to \mathbf{R}$, determine an explicit formula for $T(n)$.

(A) $T(n) = nT(n-1)$

Solution. We have:

$$\begin{aligned} T(n) &= nT(n-1) \\ T(n) &= n(n-1)T(n-2) \\ T(n) &= n(n-1)(n-2)T(n-3) \\ &\vdots \end{aligned}$$

Inductively, we can see that $T(n) = n! \cdot T(0)$.

(B) $T(n) = T(\frac{n}{2}) + c$

Solution. By guessing $T(n) = A\ln(n) + B$, we can see that:

$$A\ln(n) + B = A\ln(n) - A\ln(2) + B + c.$$

We obtain the system:

$$\begin{aligned} A &= A \\ B &= -A\ln(2) + B + c. \end{aligned}$$

Solving gives $A = \frac{c}{\ln(2)}$ and B as a free variable. Moreover:

$$T(1) = \frac{c}{\ln(2)} \ln(1) + B = B.$$

Thus $T(n) = \frac{c}{\ln(2)} \ln(n) + T(1)$.

(C) $T(n) = 3T(n-1) + cn + d$

Solution. We have:

$$\begin{aligned}
&T(n) = 3T(n-1) + cn + d \\
&T(n) = 3^2 T(n-2) + c(3(n-1) + n) + d(3+1) \\
&T(n) = 3^3 T(n-3) + c(3^2(n-2) + 3^1(n-1) + 3^0 n) + d(3^2 + 3^1 + 3^0) \\
&\quad\vdots
\end{aligned}$$

Inductively, $T(n) = 3^n T(0) + c\left(\sum_{k=1}^{n-1} 3^k (n-k)\right) + d\left(\sum_{k=1}^{n-1} 3^k\right)$. Finding the closed form of each series and simplifying gives $T(n) = 3^n T(0) + \frac{c}{4}(3^{n+1} - 2n - 3) + \frac{d}{2}(3^n - 1)$.

(D) $T(n) = T(n-1) + T(n-1)$.

Solution. We have:

$$\begin{aligned}
T(n) &= 2T(n-1) \\
T(n) &= 2^2 T(n-2) \\
T(n) &= 2^3 T(n-3) \\
&\vdots
\end{aligned}$$

Inductively, $T(n) = 2^n T(0)$.