Ziad Arafat
FEB 10, 2017

CS 111: CS Principles
Homework III

1 P.139 Q2

1.1 a.

1. Set a to 12.
2. Set b to $a + 1$.
3. Set c to $b \times a \div 12$.
4. Print the value of c

1.2 b.

78

2 P.140 Q3

2.1 a.

1. Set F to a list [1, 1]
2. Set x to 1 and y to 2
3. Set z to 1
4. While z is less than or equal to 20:
   (a) Append the sum of F[x] and F[y] to the end of list F
   (b) Increment x, y, and z by 1
5. End of loop
6. Print the value of F[20]

2.2 b.

6765

2.3 c.

I think the the recursive algorithm is more efficient simply because it's easier to implement and because it ends up being less steps than calculating the equation. It is also much simpler to understand and rings more with the fibonacci sequence. However once we approach bigger numbers it becomes clear that the equation is much more efficient because it then involves much less steps so if we were to compute F[100] I would say the equation is better over all.

2.4 d.

Done in Python

```
F = [1, 1]
x = 0
y = 1
z = 1
while z < 20:
    F.append(F[x]+F[y])
    x += 1
    y += 1
    z += 1
for i in F:
    print(i, end=",")
```

```
Run fibbo
/home/cat/PycharmProjects/CS111H3/venv/bin/python
  /home/cat/PycharmProjects/CS111H3/fibbo.py
1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987,1597,2584,4181,6765,10946,
Process finished with exit code 0
```

2.5 e.

Here is code using the while loop algorithm

```
fibbo.py
1  F = [1, 1]
2  x = 0
3  y = 1
4  z = 1
5  while z < 50:
6      F.append(F[x]+F[y])
7      x += 1
8      y += 1
9      z += 1
10 # for i in F:
11     # print(i, end=",")
12 print(F[50-1])
13
```

code using the equation (first I used numpy then I used python's default math library)

```
fibbocalc.py
1  import math
2
3  a = (math.sqrt(5)/5)*((1+math.sqrt(5))/2)**50 \
4      - (math.sqrt(5)/5)*((1-math.sqrt(5))/2)**50
5  print(a)
6
```

And here are the results using the GNU 'time' program to measure processing time. They seem to perform about the same when calculating F[50]. However it seems to depend on which math library you use. When using numpy the equation is much slower likely due it using a more complex algorithm to get precision.

```
cat@ziad:~/PycharmProjects/CS111H3$ time ./venv/bin/python3 fibbocalc.py # numpy
12586269025.000021

real    0m0.120s
user    0m0.156s
sys     0m0.183s
cat@ziad:~/PycharmProjects/CS111H3$ time ./venv/bin/python3 fibbocalc.py # default
12586269025.000021

real    0m0.041s
user    0m0.032s
sys     0m0.008s
cat@ziad:~/PycharmProjects/CS111H3$ time ./venv/bin/python3 fibbo.py
12586269025

real    0m0.040s
user    0m0.029s
sys     0m0.011s
```

3 P.141 Q7

3.1 a.

linear/sequential search takes n comparisons in worst case and 1 comparison in best case. So the average is $\Theta(\frac{n}{2})$. $(32million \div 2) \div 12000 = 1333seconds = 12minutes\ 13seconds$

3.2 b.

The average case of binary search is $\Theta(\log_2 n)$. So $\log_2 32million \div 12000 \approx 0.002s \approx 2ms$

Done in LaTeX