Cal State University Fullerton
Computer Science
Epiphany Chua
echua@csu.fullerton.edu
Dr. Sampson Akwafuo

March 26, 2022
Algorithm Engineering

CPSC 335.2: Project 2

Note about Compiling and Running Code

Executing **exhaustive.py** will run Problem 1A. It asks for n, a non-negative integer and prints the 15th term.

Executing **golden.py** will run Problem 1B. It asks for p, a non-negative integer and n, a non-negative integer greater than p. Prints F_n using each formula, the first 20 terms of the sequence using equation 4 and equation 5, and checks the maxim.

Executing **largestsum.py** will run the 4 given sample inputs and return the largest sum sub arrays.

Executing **project2.py** will run Part 1A, 1B, and 2.

Problem 1A

Pseudocode:

```
def exhaustive_fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return exhaustive_fib(n - 1) + exhaustive_fib(n - 2)
```

The time complexity of this recursive algorithm is $O(2^n)$.

$T(n) = max(2, 3, 3 + T(n-1) + T(n-2))$

Each step involves calling the function twice:
$T(n) = T(n-1) + T(n-2) = T(n-2) + T(n-3) + T(n-3) + T(n-4) = T(n-3) + T(n-4) + T(n-4) + T(n-5) + T(n-4) + T(n-5) + T(n-5) + T(n-6) = ...$

Thus, $T(n) = 2 \times 2 \times 2 \times ... \times 2 = 2^n$ and $O(2^n)$.

The 15th term of the Fibonacci sequence is

```
610
```

Problem 1B

$$F_n = \frac{(1+\sqrt{5})^n - (1-\sqrt{5})^n}{2^n\sqrt{5}}$$

Pseudocode for Equation 3:

```
def equation3(n):
    return ((math.pow(1 + math.sqrt(5), n) - (math.pow(1 - math.sqrt(5), n))) /
    (math.pow(2, n) * math.sqrt(5)))
```

$$F_n \approx F_p \left(\frac{1+\sqrt{5}}{2}\right)^{n-p}$$

Pseudocode for Equation 4:

```
def equation4(n, p):
    fp = int(formula3(p))
    return (fp * math.pow(((1 + math.sqrt(5)) / 2), n-p))
```

$$F_{n+1} \approx F_n \left(\frac{1+\sqrt{5}}{2}\right)$$

Pseudocode for Equation 5:

```
def equation5(n):
    fn = int(equation3(n-1))
    return (fn * ((1 + math.sqrt(5)) / 2))
```

Big O Efficiency Class Analysis:
Equation 3 has a step count of 1, and has a time complexity of $O(1)$.
Equation 4 has a step count of $1+2$, and has a time complexity of $O(1)$.
Equation 5 has a step count of $1+2$, and has a time complexity of $O(1)$.

The first 20 terms of the Fibonacci sequence are:

```
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584,
4181, 6,765
```

The first 20 terms using equation 4 with p = n-1:

```
0.0, 1.618033988749895, 1.618033988749895, 3.23606797749979, 4.854101966249685,
8.090169943749475, 12.94427190999916, 21.034441853748632, 33.97871376374779,
55.01315561749642, 88.99186938124421, 144.00502499874065, 232.99689437998487,
377.0019193787255, 609.9988137587104, 987.0007331374359, 1596.9995468961463,
2584.0002800335824, 4180.999826929728, 6765.000106963311
```

The first 20 terms using equation 5:

```
0.0, 1.618033988749895, 1.618033988749895, 3.23606797749979, 4.854101966249685,
8.090169943749475, 12.94427190999916, 21.034441853748632, 33.97871376374779,
55.01315561749642, 88.99186938124421, 144.00502499874065, 232.99689437998487,
377.0019193787255, 609.9988137587104, 987.0007331374359, 1596.9995468961463,
2584.0002800335824, 4180.999826929728, 6765.000106963311
```

As expected, when $p = n-1$, the results from equation 4 and 5 are the same. The difference between the equations is seen when $p \neq n-1$. For example, the following is the first 20 terms in the sequence using equation 4 with p = 1.

```
0.0, 1.618033988749895, 2.618033988749895, 4.23606797749979, 6.854101966249686,
11.090169943749476, 17.944271909999163, 29.03444185374864, 46.978713763747805,
76.01315561749645, 122.99186938124426, 199.0050249987407, 321.996894379985,
521.0019193787257, 842.9988137587108, 1364.0007331374366, 2206.9995468961474,
3571.000280033584, 5777.999826929732, 9349.000106963316
```

Therefore, the outputs of equation 4 are more accurate and similar to equation 5 as the value of $n - p$ decreases.

The Golden Ratio:

$$\phi = \frac{f(n+1)}{f(n)} \approx 1.61803 = \frac{1+\sqrt{5}}{2}$$

Maxim: The ratio of two consecutive Fibonacci numbers approaches the Golden Ratio, as n gets bigger.

$$F_3/F_2 = \frac{1.618033988749895}{1} = 1.618033988749895$$

$$F_{30}/F_{29} = \frac{832040.0000008697}{514229.00000000047} = 1.6180339887498936$$

$$\left|\phi - \frac{F_3}{F_2}\right| = 0.0$$

$$\left|\phi - \frac{F_{30}}{F_{29}}\right| = 1.3322676295501878e - 15$$

Therefore, since the ratio of F_{30}/F_{29} is further away from the golden ratio than the ratio F_3/F_2, the maxim is not supported.

Problem 2

Pseudocode (from Project Description):

```
largest sum (V):
    b = 0, e = 1
    for i from 0 to n-1:
        for j from i+1 to n:
            if sum(V[i:j]) > sum(V[b:e]):
                b = i, e = j
    return (b, e)
```

This algorithm has a step count of $2 + n[2(n - i + 1)] + 1 = 3 + 2n^2 - 2ni + 2n = n^2$
Therefore, the Big O efficiency class is $O(n^2)$.

Example inputs with the indices and largest sum sub-array:

```
v_1 = [10, 2, -5, 1, 9, 0, -4, 2, -2]
```

$(0, 5) \rightarrow [10, 2, -5, 1, 9]$

```
v_2 = [-7, 1, 8, 2, -3, 1]
```

$(1, 4) \rightarrow [1, 8, 2]$

```
v_3 = [9, 7, 2, 16, -22, 11]
```

$(0, 4) \rightarrow [9, 7, 2, 16]$

```
v_4 = [6, 1, 9, -33, 7, 2, 9, 1, -3, 8, -2, 9, 12, -4]
```

$(4, 13) \rightarrow [7, 2, 9, 1, -3, 8, -2, 9, 12]$