Matthew Horowitz

A20377155

CS 430- Section 5 (Online)

Girish Rajani-Bathija

A20503736

CS 430- Section 4 (Live)

Bhavesh Rajesh Talreja

A20516822

CS 430-Section 4 (Live)

Muhammad Sheheryar Qureshi

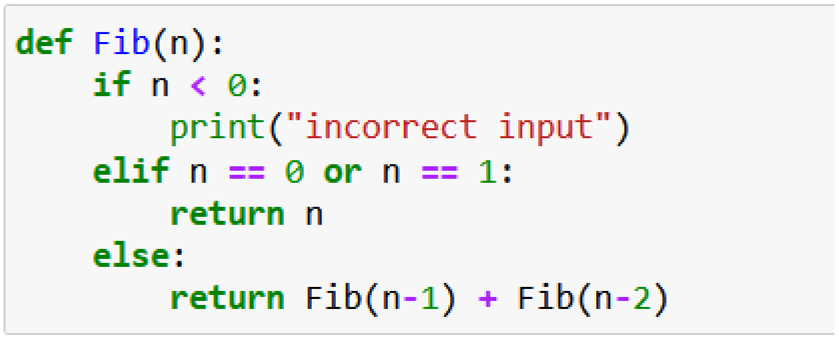
A20517229

CS 430-Section 4 (Live)

Homework Three

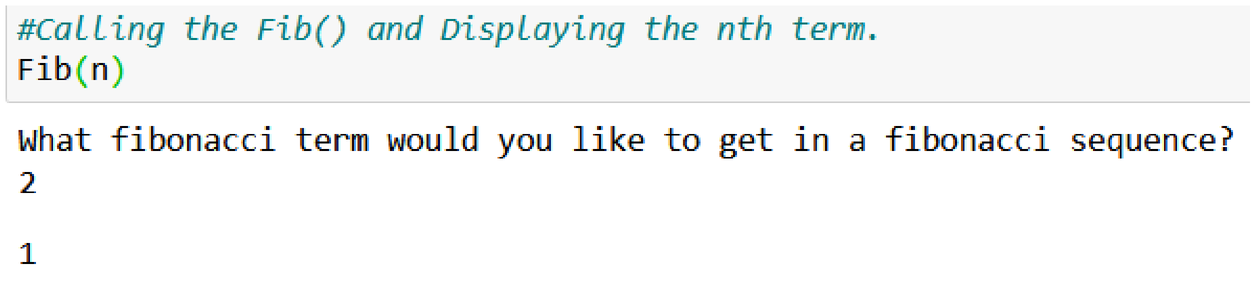
(a) Calculate recursively F(n) by applying the top-down procedure Fib( ) which calls itself. Use your favorite programming language to write Fib( ). Test Fib( ) for different numerical values of n to make sure that Fib(n) correctly calculates F(n). You can use the formula for F(n) given in item (f) below to check your numerical results

Below is the code for the method Fib() used to recursively call the Fibonacci sequence number (F(n)) for the nth term:



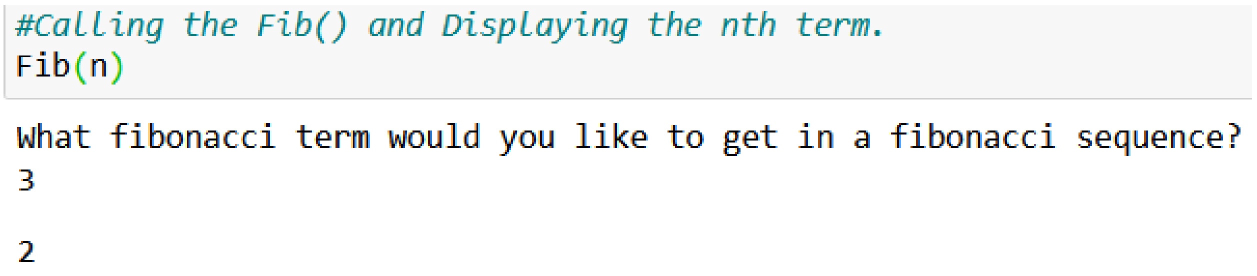
```
def Fib(n):
    if n < 0:
        print("incorrect input")
    elif n == 0 or n == 1:
        return n
    else:
        return Fib(n-1) + Fib(n-2)
```

Below are a few outputs that show that the method runs correctly, cross verified against the direct formula:



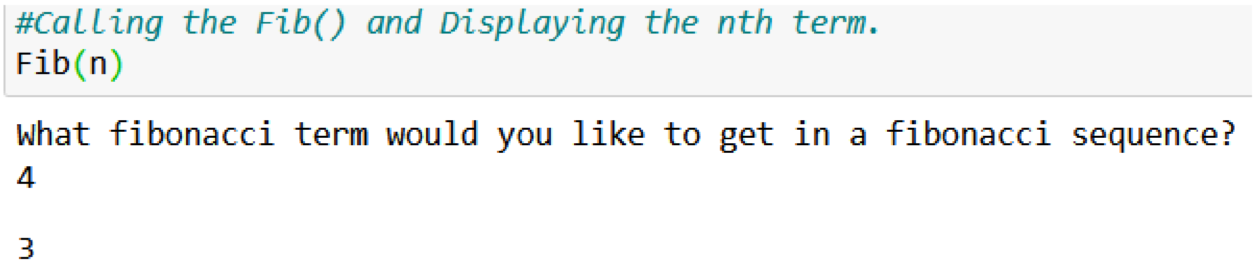
```
#Calling the Fib() and Displaying the nth term.
Fib(n)
```

```
What fibonacci term would you like to get in a fibonacci sequence?
2

1
```



```
#Calling the Fib() and Displaying the nth term.
Fib(n)
```

```
What fibonacci term would you like to get in a fibonacci sequence?
3

2
```



```
#Calling the Fib() and Displaying the nth term.
Fib(n)
```

```
What fibonacci term would you like to get in a fibonacci sequence?
4

3
```

```
#Calling the Fib() and Displaying the nth term.
Fib(n)
```

```
What fibonacci term would you like to get in a fibonacci sequence?
5

5
```

```
#Calling the Fib() and Displaying the nth term.
Fib(n)
```

```
What fibonacci term would you like to get in a fibonacci sequence?
6

8
```

```
#Calling the Fib() and Displaying the nth term.
Fib(n)
```

```
What fibonacci term would you like to get in a fibonacci sequence?
7

13
```

```
#Calling the Fib() and Displaying the nth term.
Fib(n)
```

```
What fibonacci term would you like to get in a fibonacci sequence?
8

21
```

Below are the outputs for F(n) by calculating the closed form of the Fibonacci Sequence:

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

n	F(n)
2	1
3	2
4	3
5	5
6	8
7	13
8	21

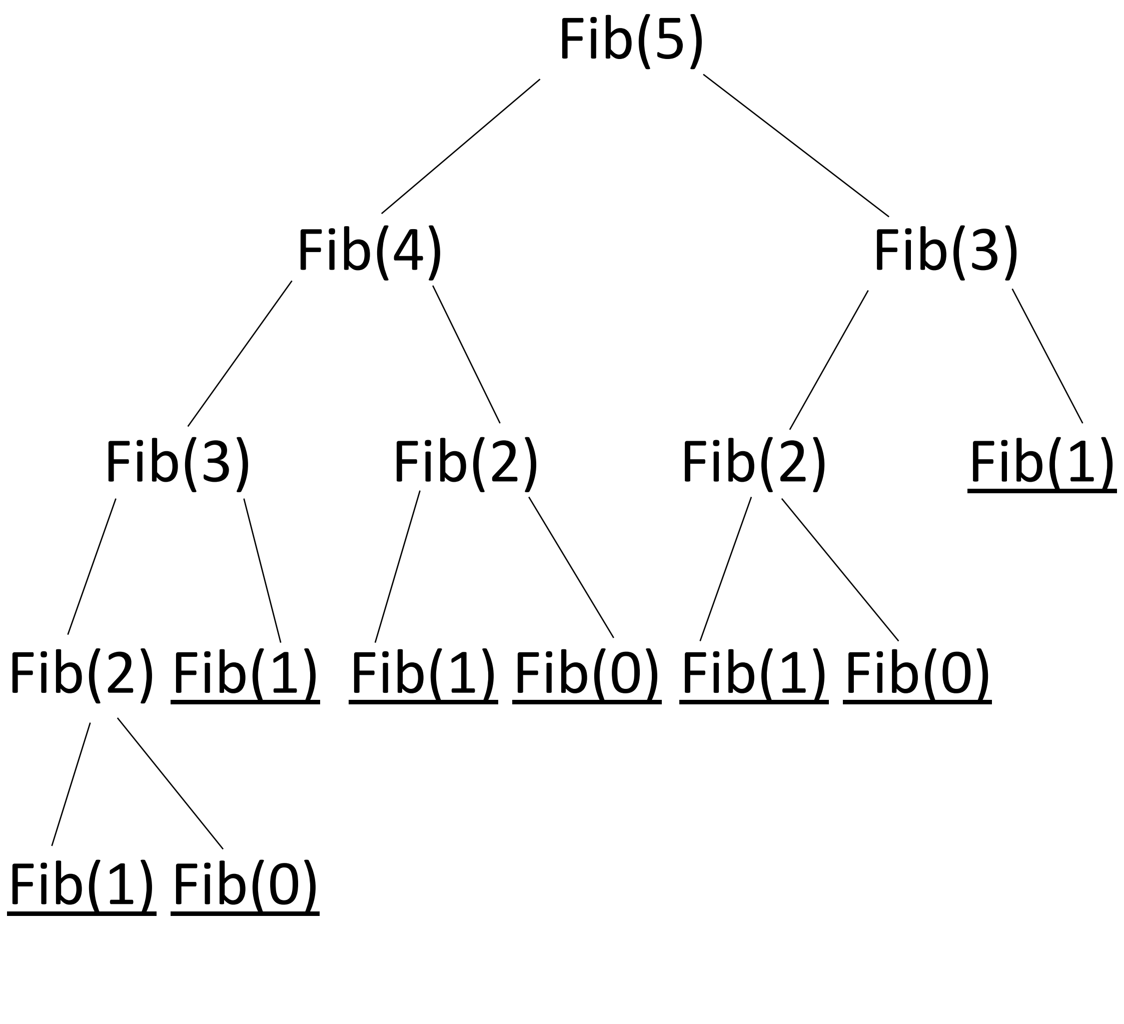
As is noted, the program runs correctly.

The entire Python code required to calculate the Fibonacci sequence can be found in the Python script file submitted with this report.

(b) Draw the recursion tree for Fib(5). Let's denote by T(n) the total number of calls to Fib( ) needed to calculate F(n) (note that T(n) includes the original call to Fib(n) at the root). What is the value of T(5)?

Below is the tree for Fib(5), showing the number of recursive calls made T(n) , including the root, to calculate the 5th term of Fibonacci series, where F(5) = 5:

Recursion tree T(5):

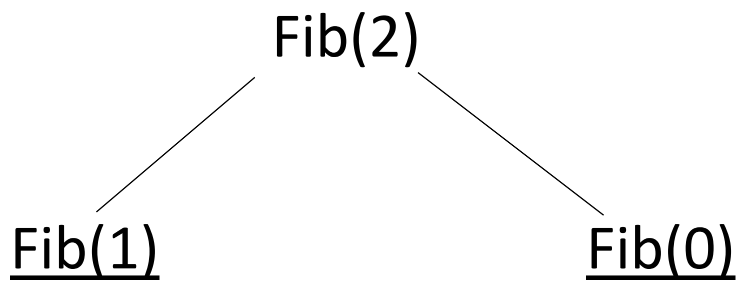


Having drawn the tree, it takes **<u>15 calls</u>** (including the root) to calculate F(5) recursively. As such, **T(5)= 15**.

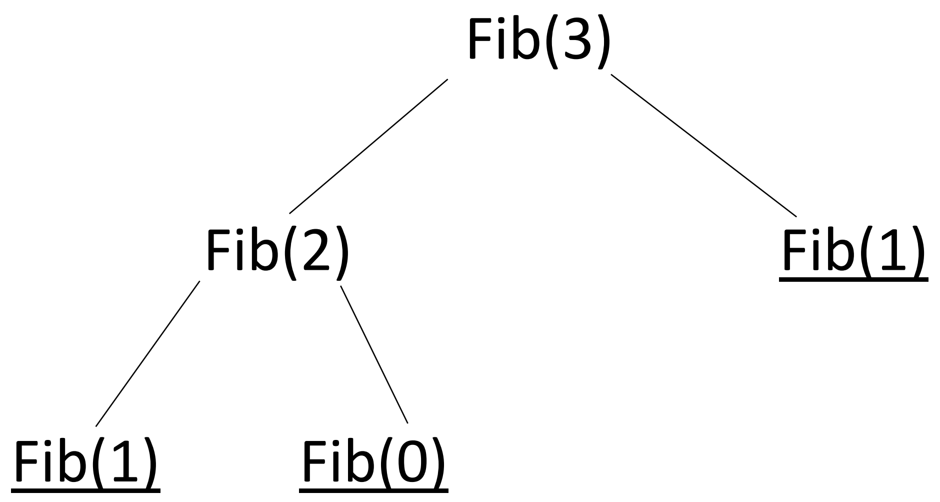
(c) What is the recurrence for T(n)? What are the initial conditions of this recurrence?

In order to calculate the recurrence for T(n), the recursion trees for the n=2 to 8 Fibonacci sequence terms were drawn. Below are those trees:

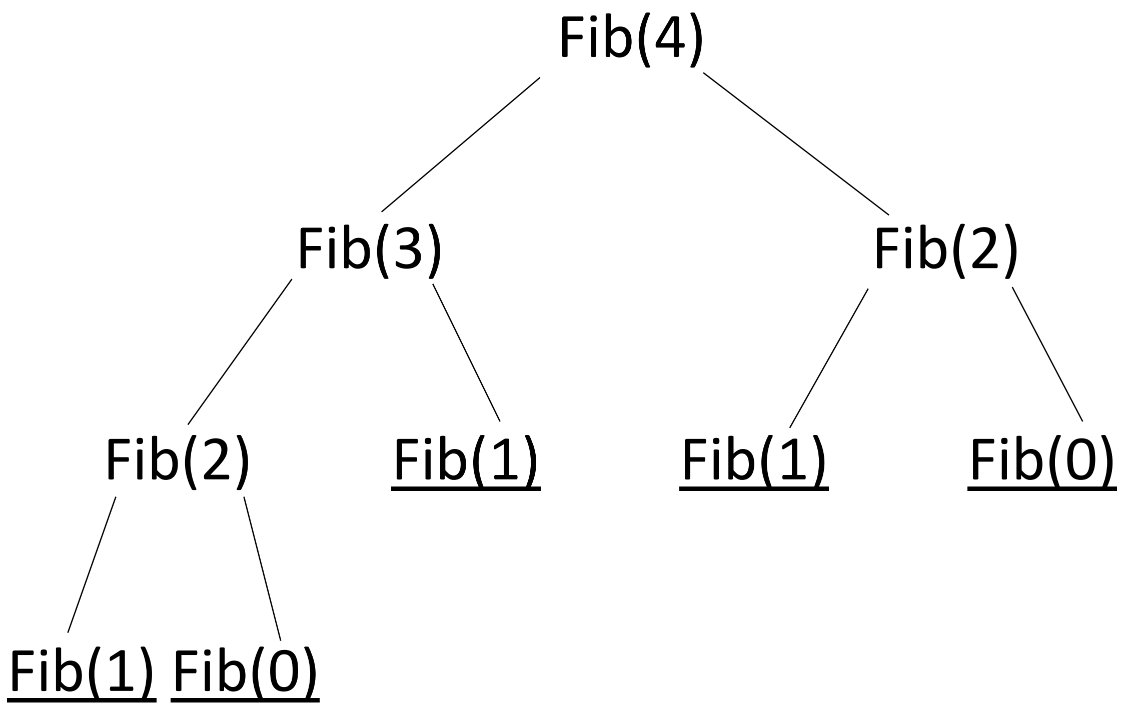
Recursion tree T(2):



Recursion tree T(3):

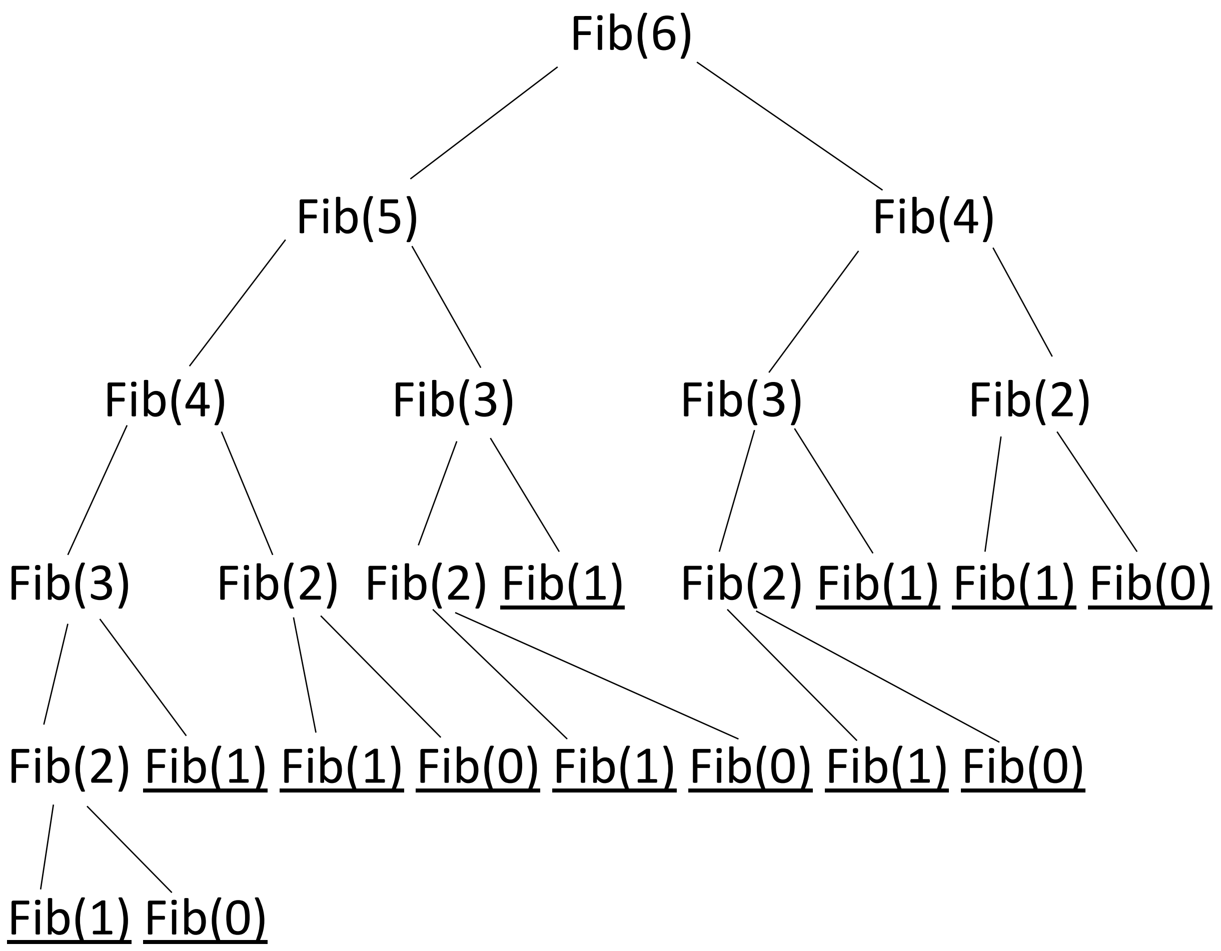


Recursion tree T(4):

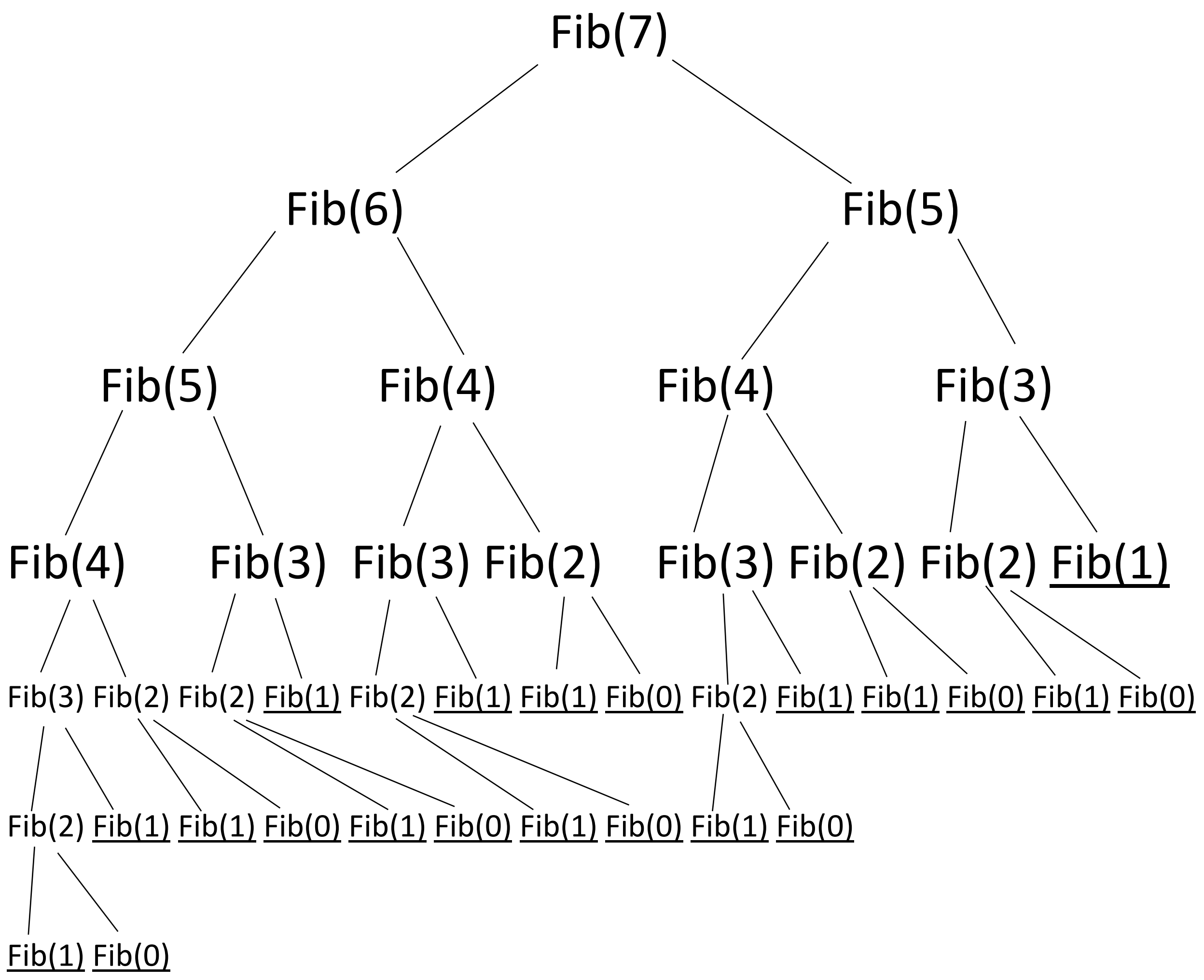


Recursion Tree F(5): see Section B.

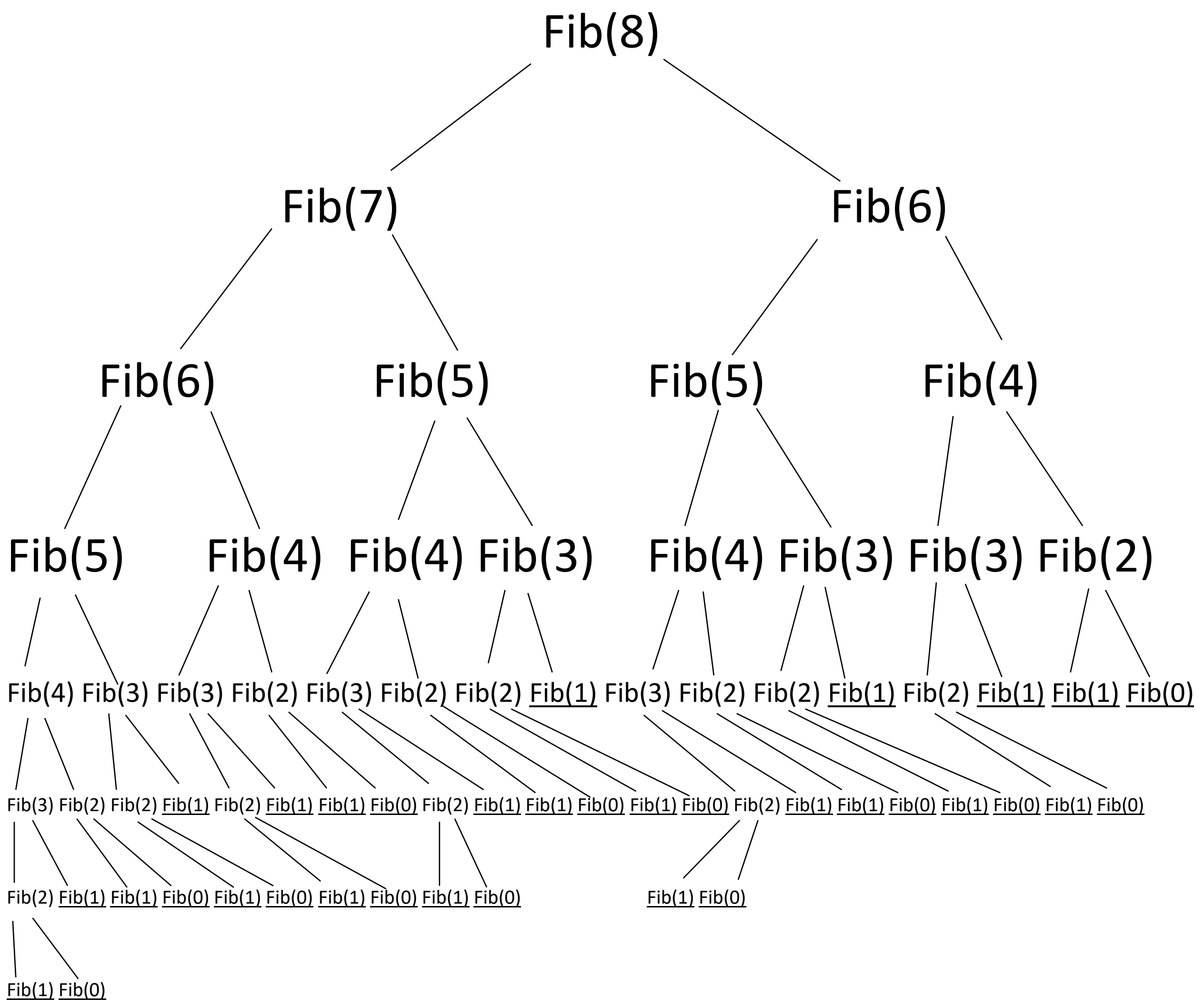
Recursion tree T(6):



Recursion tree T(7):



Recursion tree T(8):


Fib(8)
Fib(7) Fib(6)
Fib(6) Fib(5) Fib(5) Fib(4)
Fib(5) Fib(4) Fib(4) Fib(3) Fib(4) Fib(3) Fib(3) Fib(2)
Fib(4) Fib(3) Fib(3) Fib(2) Fib(3) Fib(2) Fib(2) Fib(1) Fib(3) Fib(2) Fib(2) Fib(1) Fib(2) Fib(1) Fib(1) Fib(0)
Fib(3) Fib(2) Fib(2) Fib(1) Fib(2) Fib(1) Fib(1) Fib(0) Fib(2) Fib(1) Fib(1) Fib(0) Fib(1) Fib(0) Fib(2) Fib(1) Fib(1) Fib(0) Fib(1) Fib(0) Fib(1) Fib(0)
Fib(2) Fib(1) Fib(1) Fib(0) Fib(1) Fib(0) Fib(1) Fib(0) Fib(1) Fib(0) Fib(1) Fib(0)
Fib(1) Fib(0)

Next, the results from the above tree diagrams, were placed into the table below, where the value of each Fibonacci term F(n) and the number of recursive calls required to calculate the F(n)—T(n)--were recorded:

n	Number of Calls	F(n)
0	1	0
1	1	1
2	3	1
3	5	2
4	9	3
5	15	5
6	25	8
7	41	13
8	67	21
9	109	34

Note: Due to space limitations, the tree for T(9) was not included.

In the above table, one notices that the number of calls it takes to recursively calculate a term in a Fibonacci sequence is always one less than twice the F(n+1) value. This pattern holds for all When $n \geq 2$, thus yielding, T(n)=2F(n+1)-1. In addition F(0)=0, requiring one call, F(1)=1, requiring one call as well, in turn yielding T(0) and T(1)=1, and as such are the base cases.

So far, it seems that the number of calls for a Fibonacci term is:

$$T(n) = \begin{cases} 1, & n = 0 \; and \; n = 1 \\ 2F(n+1) - 1, & n \geq 2 \end{cases}$$

However, this is not a recurrence, as it still requires the number of calls to be defined based on a Fibonacci value. But looking at the table above, one can see another pattern arise. It appears that the number of calls for a Fibonacci term is always the number of calls of the two previous terms summed, plus one. This yields the following equation:

$\boldsymbol{T(n) = 1 + T(n-1) + T(n-2)}$,

Equivalent to:

$\boldsymbol{T(n) = \theta(1) + T(n-1) + T(n-2)}$,

which makes sense given the Python code used to run the recursive calls. This code makes an initial call (the $\boldsymbol{\theta(1)}$ in the above formula which represents the initial call and summing the terms which have been returned), and calls into fib(n-1) and fib(n-2) meaning that the total time complexity should be the sum of the two previous time complexities plus the initial call.

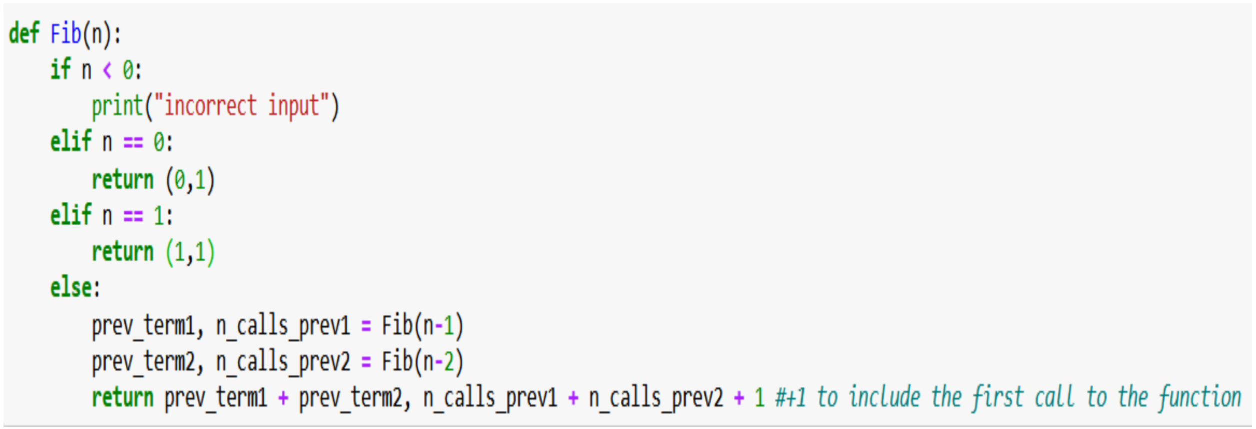
As such, the recursive time complexity is:

$$\boldsymbol{T(n)} = \begin{cases} \mathbf{1}, & \boldsymbol{n = 0\ and\ n = 1} \\ \mathbf{1} + \boldsymbol{T(n-1)} + \boldsymbol{T(n-2)}, & \boldsymbol{n \geq 2} \end{cases}$$

Where n=0 and n=1 have one call, while $n \geq 2$ follows the above pattern.

(d) Use Fib( ) to empirically verify that T(n)=2F(n+1)-1.

For this section, a counter was added to the method to keep track of the number of calls recursively made to verify that that T(n)=2F(n+1)-1 does in fact hold. This was done as it becomes too difficult to draw the recursion trees for n greater than or equal to 9. Below is the line of code showing the counter. The entire python code can be found along with this report:



```python
def Fib(n):
    if n < 0:
        print("incorrect input")
    elif n == 0:
        return (0,1)
    elif n == 1:
        return (1,1)
    else:
        prev_term1, n_calls_prev1 = Fib(n-1)
        prev_term2, n_calls_prev2 = Fib(n-2)
        return prev_term1 + prev_term2, n_calls_prev1 + n_calls_prev2 + 1 #+1 to include the first call to the function
```

After running the Python code, here are the outputs for test cases n=9 to n=22

```
Fibonacci term 9 is 34
Number of calls: 109
Fibonacci term 10 is 55
Number of calls: 177
Fibonacci term 11 is 89
Number of calls: 287
Fibonacci term 12 is 144
Number of calls: 465
Fibonacci term 13 is 233
Number of calls: 753
Fibonacci term 14 is 377
Number of calls: 1219
Fibonacci term 15 is 610
Number of calls: 1973
Fibonacci term 16 is 987
Number of calls: 3193
Fibonacci term 17 is 1597
Number of calls: 5167
Fibonacci term 18 is 2584
Number of calls: 8361
Fibonacci term 19 is 4181
Number of calls: 13529
Fibonacci term 20 is 6765
Number of calls: 21891
Fibonacci term 21 is 10946
Number of calls: 35421
Fibonacci term 22 is 17711
Number of calls: 57313
```

Cross verifying the number of calls with the Fibonacci outputs per term n+1, it does in fact appear that, $T(n)=2F(n+1)-1$ is correct.

(e) Given the recurrence for T(n) found in item (c) above, prove by induction that the formula T(n)=2F(n+1)-1 is correct.

This section will now prove the following:

$$T(n) = \begin{cases} 1, & n = 0 \; and \; n = 1 \\ 2F(n+1) - 1, & n \geq 2 \end{cases}$$

Using the recurrence formula found in section C:

$$T(n) = \begin{cases} 1, & n = 0 \; and \; n = 1 \\ 1 + T(n-1) + T(n-2), & n \geq 2 \end{cases}$$

Step One: Base Case Tests

Following the rules of mathematical induction, first, the base cases will be tested. In the above recurrence, the base cases are n=0 and n=1:

n=0, number of calls 1
  Calculation:
  T(0) = 2F(1)-1 = 2*1 -1 =1, holds
n=1, number of calls 1
  Calculation:
  T(1) = 2F(2)-1 = 2*1-1 =1, holds

Step Two: Mathematical Induction

Next, mathematical induction will be conducted to show that T(n)=2F(n+1)-1 holds for all n terms, such that T(n+1)=2F((n+1)+1)-1 holds. It should noted that 2F((n+1)+1)-1, is equivalent to 2F(n+2)-1.

a. Using the recursion found:

$$T(n) = 1 + T(n-1) + T(n-2)$$

b. Assuming the definition of T(n) holds:

$$T(n) = 2F(n+1) - 1$$

c. Per rules of sequences:

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

d. Using substitution:

$$T(n+1) = 1 + 2F(n+1) - 1 \; + \; 2F(n) - 1$$

e. Simplifying:

$$T(n+1) \;=\; 2F(n+1) \; + 2F(n) \; - 1$$

f. Using the definition of Fibonacci provided:

$$F(n) = F(n-1) + F(n-2)$$

g. And the associated phase shift:

$$F(n+2) = F(n+1) + F(n)$$

h. Multiplying the output from g by 2.

$$2F(n+2) = 2F(n+1) + 2F(n)$$

i. Plug in value from h into e:

$$T(n+1) \;=\; 2F(n+1) \; + 2F(n) \; - 1$$

j. The values

$$2F(n+1) \; + 2F(n)$$

k. Will be replaced with:

$$2F(n+2)$$

l. Therefore:

$$T(n+1) = 2F(n+2) - 1$$

m. Which is equivalent to:

$$T(n+1) = 2F((n+1)+1) - 1$$

Which means the induction holds and T(n)=2F(n+1)-1 is true for every value of n.

(f) Use the formula given in item (d) to find the asymptotic complexity T(n)=O(?) of Fib( ). The closed-form expression for the Fibonacci numbers F(n) = φ n−ψ n √5 , where φ = 1+√5 2 and ψ= 1−√5 2 , may be helpful.

To determine the asymptotic complexity of T(n), the first step is to look at the outputs found from running the recursive code. A portion of the table presented in section C has been recreated below:

n	Number of Calls
0	1
1	1
2	3
3	5
4	9
5	15
6	25
7	41
8	67
9	109

Above, it appears that for the growth in recursive call size per each increase in n is about 2^n, leading to the belief that the asymptotic complexity of T(n) is O(2^n).

But on second glance, it does appear that the growth of T(n) is actually a little bit less than 2^n. As such, it becomes wise to look at the formula of the Fibonacci sequence, and in particular the derivation of its closed form. The closed form solution for the Fibonacci sequence can be said to be:

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

where the roots **φ** and **ψ** can be defined as $\boldsymbol{\phi} = (\frac{1+\sqrt{5}}{2})$ and $\boldsymbol{\psi} = (\frac{1-\sqrt{5}}{2})$. These values come from the homogeneous characteristic equation.

The characteristic equation is an equation that allows one to find the roots of a recurrence, where the roots define the change in the sequence, thus yielding a specific term for a given sequence, its closed form.[1] Provided a recurrence meets the following pattern, the characteristic equation's roots can be used to determine the closed form:

$a_n = a_{n-1} + a_{n-2}$ [2]

Since the Fibonacci sequence does fit this pattern, we can apply to the Fibonacci sequence the homogeneous characteristic equation and solve for its roots.

The conversion can be seen by rewriting the recurrence into the form:

$$a_n + \alpha a_{n-1} + \beta a_{n-1}$$

into the homogeneous characteristic equation takes the form of:

$$x^2 + \alpha x + \beta = 0$$

where alpha and beta are solutions of the initial conditions, and then solving for its roots, to get the closed form.[3]

Applying this technique to the Fibonacci sequence yields the characteristic equation:

$x^2 - x - 1 = 0,$

whose roots are $\frac{1+\sqrt{5}}{2}$ and $\frac{1+\sqrt{5}}{2}$, after applying the quadratic formula.

What this means is that for each input of n, each term F(n) grows by $\frac{(\frac{1+\sqrt{5}}{2})^n - (\frac{1-\sqrt{5}}{2})^n}{\sqrt{5}}$, amount, for n increasing to n+1. Since it has been determined that the number of recursive calls T(n) needed to compute a Fibonacci sequence is simply 2F(n+1)-1, one can substitute the closed form of the formula into the recurrence to get a time complexity.

[1] Levin, O (2021), Discrete Mathematics: An Open Introduction, Third Edition, University of Northern Colorado, Creative Commons, Available at: http://discrete.openmathbooks.org. pp. 171-172.
Epp, S S (2020), Discrete Mathematics with Applications, Fifth Edition, Cengage: Boston. Pp. 352-363.
[2] Levin, O (2021), Discrete Mathematics: An Open Introduction, Third Edition, University of Northern Colorado, Creative Commons, Available at: http://discrete.openmathbooks.org., pp. 172.
[3] Ibid., pp. 172.

As such:

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

Using convention, the asymptotic complexity is just the dominant term, which in this case, after factoring out 2, $1/\sqrt{5}$, and the addition roots to the first power, the following is the result:

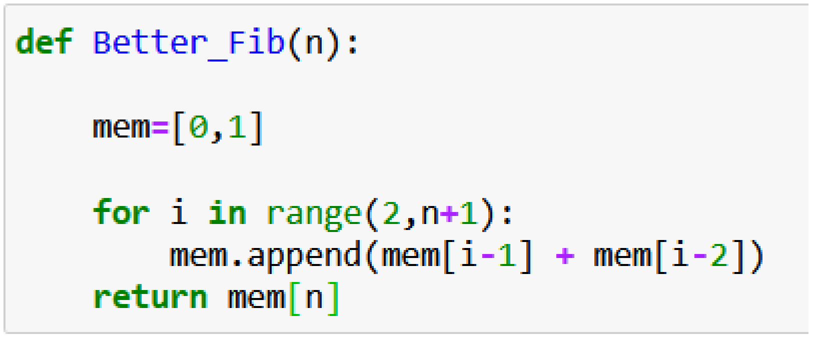
$(\frac{1+\sqrt{5}}{2})^n - (\frac{1-\sqrt{5}}{2})^n$.

Since the positive term, $(\frac{1+\sqrt{5}}{2})^n$ is dominant, relative to $-(\frac{1-\sqrt{5}}{2})^n$ term, one can say that **T(n)=O($(\frac{1+\sqrt{5}}{2})^n$), which is less than 2^n.**[4]

This makes sense as the number of recursive calls grows at a fixed rate [2F(n+1)-1], which is expressed in the terms of the next Fibonacci number. Since, Fibonacci numbers also grow at a fixed rate $\frac{(\frac{1+\sqrt{5}}{2})^n - (\frac{1-\sqrt{5}}{2})^n}{\sqrt{5}}$, it would make sense that the tight time complexity of the number of recursive calls should be the dominant term of closed form of Fibonacci sequence.

(g) Implement in your favorite programming language the bottom-up iterative procedure Better_Fib( ) with "memoization" for better performance. Test Better_Fib( ) for different numerical values of n to make sure that it correctly calculates F(n). You can use the formula for F(n) given in item (f) above to check your numerical results.

Replicated below is the method for Better_Fib() using iterative memoization. Included in report submission is the complete Python code:



```
def Better_Fib(n):

    mem=[0,1]

    for i in range(2,n+1):
        mem.append(mem[i-1] + mem[i-2])
    return mem[n]
```

[4] Corman, T.H.; Leiserson, C.E.; Rivest, R.L.; Stein, C (2009), Introduction to Algorithms, Third Edition, MIT: Cambridge, MA.

Below are a few outputs that show that the method runs correctly, cross verified against the direct formula:

```
#Calling the Better_Fib() and displaying the nth term.
Better_Fib(n)
```

```
What fibonacci term would you like to get in a fibonacci sequence?
2

1
```

```
#Calling the Better_Fib() and displaying the nth term.
Better_Fib(n)
```

```
What fibonacci term would you like to get in a fibonacci sequence?
3

2
```

```
#Calling the Better_Fib() and displaying the nth term.
Better_Fib(n)
```

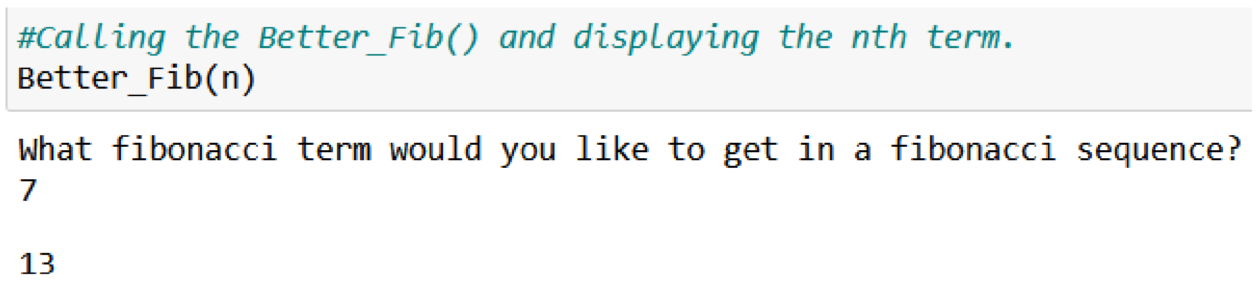
```
What fibonacci term would you like to get in a fibonacci sequence?
4

3
```

```
#Calling the Better_Fib() and displaying the nth term.
Better_Fib(n)
```

```
What fibonacci term would you like to get in a fibonacci sequence?
5

5
```

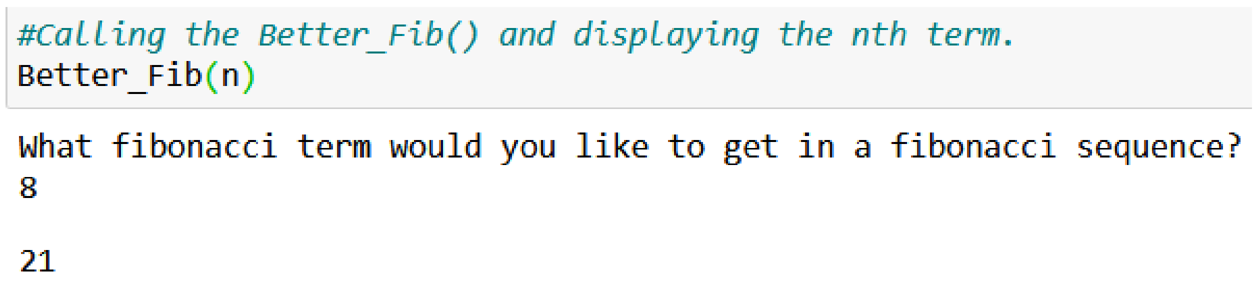
```
#Calling the Better_Fib() and displaying the nth term.
Better_Fib(n)
```

```
What fibonacci term would you like to get in a fibonacci sequence?
6

8
```



```
#Calling the Better_Fib() and displaying the nth term.
Better_Fib(n)
```

```
What fibonacci term would you like to get in a fibonacci sequence?
7

13
```



```
#Calling the Better_Fib() and displaying the nth term.
Better_Fib(n)
```

```
What fibonacci term would you like to get in a fibonacci sequence?
8

21
```

This is the closed form Fibonacci formula and its results with different values of n. This table cross referenced with the above outputs shows that the results hold.

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

n	F(n)
2	1
3	2
4	3
5	5
6	8
7	13
8	21

The entire Python code required to calculate the Fibonacci sequence can be found in the Python script file submitted with this report.

(h) What is the asymptotic complexity TBF(n)=O(?) of Better_Fib( )? Submit source codes of all programs you have used to produce your results.

The iterative memoized code simply adds the two previous terms (kth item), by accessing their values from the array and summing them, and appends them into an array for later use, for all items F(2) to F(n). As such, for each kth item, where $0 \leq k \leq n$, the time complexity is a constant. This process is repeated n times, which is a **time complexity of O(n).**

Division of Work: The team worked together to create this report.

Matt worked on proof by induction i.e., Q1(e) and time complexity of T(n) i.e., Q1(f).

Bhavesh implemented the Python code for Q1(a), Q1(g) and the counter implementation for Q1(d).

Girish worked on drawing the recursion trees in Q1(c) and time complexity of better_Fib() in Q1(h).

Sheheryar worked on calculating Fib(5) Q1(b) and empirically verify that T(n)=2F(n+1)-1 in Q1(d).

Together the team then reviewed the sections of the report and worked on final edits.

References:

Corman, T.H.; Leiserson, C.E.; Rivest, R.L.; Stein, C (2009), Introduction to Algorithms, Third Edition, MIT: Cambridge, MA.

Epp, S S (2020), Discrete Mathematics with Applications, Fifth Edition, Cengage: Boston. Pp. 352-363.

Levin, O (2021), Discrete Mathematics: An Open Introduction, Third Edition, University of Northern Colorado, Creative Commons, Available at: http://discrete.openmathbooks.org.