S Abhishek AM.EN.U4CSE19147

0 - Pre Lab - Paranthesis Matching

Add a code cell and implement the parenthesis matching algorithm. Test your code

In [6]:

```
def check(x):
    c = 0
    for i in x:
        if i == "(":
            c += 1
        elif i == ")":
            c -= 1
        if c < 0:
            return False

    if c == 0:
      return True
    else:
      return False

if __name__ == '__main__':
  x = input("Enter the String : ")

  if check(x) == True:
    print("Yay! Balanced :)")
  else:
    print("No! Not Balanced :(")
```

```
Enter the String : ((((()))))
Yay! Balanced :)
```

1 - Fibonnaci Series

Write a program to compute the nth Fibonacci number using,

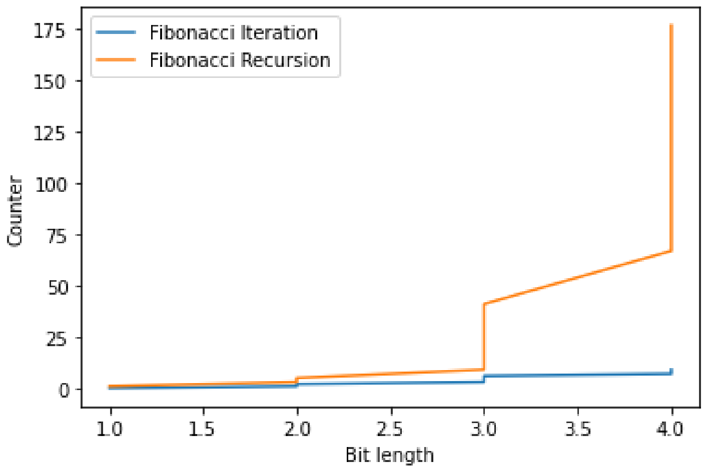
1. **Iterative Fibonacci Algorithm**
2. **Recursive Fibonacci Algorithm**

In [15]:

```
import matplotlib.pyplot as p

count = 0

def Iterative(n): # Iterative Algorithm
  x = 0
  y = 1

  if n == 0 or n == 1:
    return n

  for i in range(0,n-1):
    z = x + y
    x = y
    y = z

  return y
```

```
def Recursive(n): # Recursive Algorithm
    global count
    count += 1
    if n == 0 or n == 1:
        return n
    return Recursive(n-1) + Recursive(n-2)


if __name__ == '__main__':

  nos = [1,2,3,4,5,6,7,8,9,10] # List of 10 Numbers

  I_Count = [] # List to store the Counter of Iterative Algorithm
  R_Count = [] # List to store the Counter of Recursive Algorithm

  bit = [] # List to store the Bit Length

  print("Fibonacci Series : ",end="")

  for i in nos:

    print(Iterative(i),end = " ")

    I_Count.append(i-1) # Append the Counter of Iterative Algorithm to the List

    count = 0 # Reset Count each time
    Recursive(i) # Call the Recursive Function

    R_Count.append(count) # Append the Counter of Recursive Algorithm to the List
    bit.append(i.bit_length()) # Append Bit Length to the List

print("\n")

# Graph Plotting

p.plot(bit, I_Count, label="Fibonacci Iteration") # Plotting X and Y co-ordinates
p.plot(bit, R_Count, label = "Fibonacci Recursion") # Plotting X and Y co-ordinates

p.xlabel("Bit length") # Labelling X Axis
p.ylabel("Counter") # Labelling Y Axis

p.legend() # Area describing the elements of the Graph
p.show() # Display the Graph
```

```
Fibonacci Series : 1 1 2 3 5 8 13 21 34 55
```



2 - GCD of 2 Numbers

Write a program to find the Greatest Common Divisor (GCD) of two numbers using,

1. **Euclid's algorithm**
2. **Euclidean algorithm**
3. **Binary GCD algorithm**

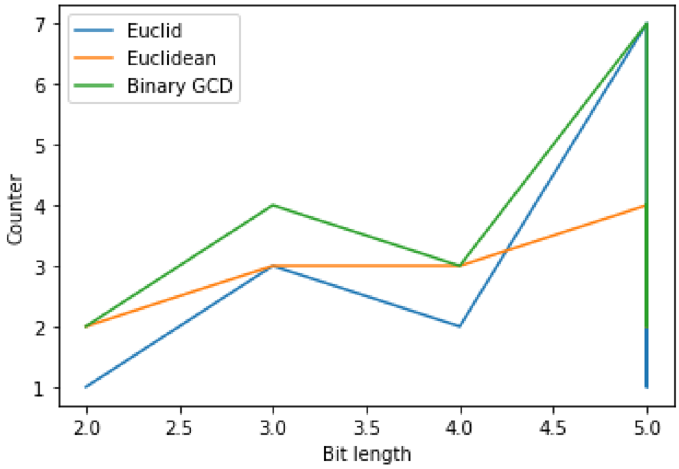
Count the number of iterations/recursions for each input. Plot the operation count against the bit length of largest among the two input numbers.

In [8]:

```
import matplotlib.pyplot as p

count = 0

def Euclid(x,y): # Euclid's Algorithm
  global count
  while(x != y):
    count += 1
    if (x > y):
      x = x - y
    else:
      y = y - x;

  return x


def Euclidean(x,y): # Euclidean Algorithm
    global count
    count += 1
    if x == 0:
        return y

    return Euclidean(y%x, x)


def Binary(x,y): # Binary GCD Algorithm
    global count
    count += 1

    if x == y: # Gcd(n, n) = n
      return x

    if x == 0: # Gcd(0, n) = n
      return y

    if y == 0: # Gcd(n, 0) = n
      return x

    if x%2 == 0 and y%2 == 0: # Both x and y are even
      return 2*Binary(x/2,y/2)

    if x%2 == 0 and y%2 == 1: # x is Even and y is Odd
      return Binary(x/2,y)

    if x%2 == 1 and y%2 == 0: # x is Odd and y is Even
      return Binary(x,y/2)

    if x > y:
      return Binary((x - y)/2, y);

    if x < y:
      return Binary((y - x)/2, x);


if __name__ == '__main__':

  Euclid_C = []
  Euclidean_C = []
  Binary_C = []
  bit = []
```

```
count = 0
Euclid(1,2)
Euclid_C.append(count)

count = 0
Euclidean(1,2)
Euclidean_C.append(count)

count = 0
Binary(1,2)
Binary_C.append(count)

bit.append((2).bit_length())


count = 0
Euclid(3,4)
Euclid_C.append(count)

count = 0
Euclidean(3,4)
Euclidean_C.append(count)

count = 0
Binary(3,4)
Binary_C.append(count)

bit.append((4).bit_length())


count = 0
Euclid(6,9)
Euclid_C.append(count)

count = 0
Euclidean(6,9)
Euclidean_C.append(count)

count = 0
Binary(6,9)
Binary_C.append(count)

bit.append((9).bit_length())


count = 0
Euclid(10,23)
Euclid_C.append(count)

count = 0
Euclidean(10,23)
Euclidean_C.append(count)

count = 0
Binary(10,23)
Binary_C.append(count)

bit.append((23).bit_length())


count = 0
Euclid(11,22)
Euclid_C.append(count)

count = 0
Euclidean(11,22)
Euclidean_C.append(count)

count = 0
Binary(11,22)
```

```
Binary_C.append(count)

bit.append((22).bit_length())


count = 0
Euclid(10,25)
Euclid_C.append(count)

count = 0
Euclidean(10,25)
Euclidean_C.append(count)

count = 0
Binary(10,25)
Binary_C.append(count)

bit.append((25).bit_length())


# Graph Plotting

p.plot(bit, Euclid_C, label="Euclid") # Plotting X and Y co-ordinates
p.plot(bit, Euclidean_C, label = "Euclidean") # Plotting X and Y co-ordinates
p.plot(bit, Binary_C, label = "Binary GCD") # Plotting X and Y co-ordinates

p.xlabel("Bit length") # Labelling X Axis
p.ylabel("Counter") # Labelling Y Axis

p.legend() # Area describing the elements of the Graph
p.show() # Display the Graph
```



3 - Worst & Best Case of 3 GCD

For each of the three gcd algorithms, identify the best case and worst case inputs and find the running times in each case. Write your answers in a text cell.

Euclidean algorithm

- **Best case is when y = 0**
- **Worst case is when two inputs are consecutive Fibonacci numbers**
- **Running time is O(log(min(x, y))**

Euclid algorithm

- **Best case is when x = 0 (or) y = 0 (or) x = y**
- **Worst case is when one of the inputs is 1**
- **Running time is O(x + y)**

Binary GCD algorithm

- Best case is when x or y is 0 or x = y
- Running time is O(log n) and for very large integers, O((log n)^2)

4 - Determine if N is Prime

Write a program to determine if a positive integer, N, is prime. Write the answers for the following questions in a text cell.

a. In terms of N, what is the worst-case running time of your program?

b. Let B equal the number of bits in the binary representation of N. What is the value of B? (as a function of N)

c. In terms of B, what is the worst-case running time of your program?

In [11]:

```
def prime(x):

  if x > 1:

    for i in range(2, int(x/2)+1):
        if (x % i) == 0:
            return False
            break
    return True

  else:

    return False


if __name__ == '__main__':

  n = int(input("Enter the N : "))

  if prime(n) == True:
    print("It's a Prime! :)")
  else:
    print("It's not a Prime! :(")
```

```
Enter the N : 37
It's a Prime! :)
```

a. In terms of N, the worst case running time is O(N)

b. If B equal the number of bits in the binary representation of N then the value of B as a function of N is log(N)

c. In terms of B, the worst-case running time is 2^B

5 - Sieve of Eratosthenes

Write a program to compute all primes up to and including n. (The Sieve of Eratosthenes method)

In [10]:

```
# Sieve of Eratosthenes Method for computing Primes

def prime(x):
  not_prime = []
  prime = []
  for i in range (2,x+1):
    if i not in not_prime:
      prime.append(i)
```

```
      for j in range(i*i,x+1,i):
        not_prime.append(j)

  return prime

if __name__ == '__main__':
  n = int(input("Enter the N : "))
  print("Primes till",n,':',prime(n))
```

```
Enter the N : 100
Primes till 100 : [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 6
7, 71, 73, 79, 83, 89, 97]
```