Конечно! Ниже представлены реализации на Python для каждого из указанных алгоритмов:

## 1) Алгоритм Евклида (Находжение НОД)

def gcd(a, b):  
 """  
 Вычисляет наибольший общий делитель (НОД) двух чисел a и b  
 с помощью классического алгоритма Евклида.  
 """  
 while b != 0:  
 a, b = b, a % b  
 return a  
  
# Пример использования  
a = int(input("Введите число a: "))  
b = int(input("Введите число b: "))  
print(f"НОД({a}, {b}) = {gcd(a, b)}")

## 2) Бинарный алгоритм Евклида (Алгоритм Штейна)

def binary\_gcd(a, b):  
 """  
 Вычисляет НОД двух чисел a и b с помощью бинарного алгоритма Евклида.  
 """  
 if a == 0:  
 return b  
 if b == 0:  
 return a  
  
 # Ищем общие множители 2  
 shift = 0  
 while ((a | b) & 1) == 0:  
 a >>= 1  
 b >>= 1  
 shift += 1  
  
 # Делим a на 2, пока оно четное  
 while (a & 1) == 0:  
 a >>= 1  
  
 while b != 0:  
 # Делим b на 2, пока оно четное  
 while (b & 1) == 0:  
 b >>= 1  
  
 # Обмениваем значения, если необходимо  
 if a > b:  
 a, b = b, a  
  
 b = b - a  
  
 return a << shift  
  
# Пример использования  
a = int(input("Введите число a: "))  
b = int(input("Введите число b: "))  
print(f"НОД({a}, {b}) = {binary\_gcd(a, b)}")

## 3) Расширенный алгоритм Евклида

def extended\_gcd(a, b):  
 """  
 Расширенный алгоритм Евклида.  
 Возвращает кортеж (gcd, x, y), такой что ax + by = gcd(a, b)  
 """  
 if b == 0:  
 return a, 1, 0  
 else:  
 gcd, x1, y1 = extended\_gcd(b, a % b)  
 x = y1  
 y = x1 - (a // b) \* y1  
 return gcd, x, y  
  
# Пример использования  
a = int(input("Введите число a: "))  
b = int(input("Введите число b: "))  
gcd\_value, x, y = extended\_gcd(a, b)  
print(f"НОД({a}, {b}) = {gcd\_value}")  
print(f"Коэффициенты: x = {x}, y = {y}")  
print(f"Проверка: {a}\*({x}) + {b}\*({y}) = {gcd\_value}")

## 4) Расширенный бинарный алгоритм Евклида

def extended\_binary\_gcd(a, b):  
 """  
 Расширенный бинарный алгоритм Евклида.  
 Возвращает кортеж (gcd, x, y), такой что ax + by = gcd(a, b)  
 """  
 if a == 0:  
 return b, 0, 1  
 if b == 0:  
 return a, 1, 0  
  
 # Инициализация  
 g = 1  
 while ((a | b) & 1) == 0:  
 a >>= 1  
 b >>= 1  
 g <<= 1  
  
 x, lastx = 1, 0  
 y, lasty = 0, 1  
  
 while a != 0:  
 while (a & 1) == 0:  
 a >>= 1  
 if (x & 1) == 0 and (y & 1) == 0:  
 x >>= 1  
 y >>= 1  
 else:  
 x = (x + b) >> 1  
 y = (y - a\_original) >> 1  
  
 while (b & 1) == 0:  
 b >>= 1  
 if (lastx & 1) == 0 and (lasty & 1) == 0:  
 lastx >>= 1  
 lasty >>= 1  
 else:  
 lastx = (lastx + b\_original) >> 1  
 lasty = (lasty - a\_original) >> 1  
  
 if a >= b:  
 a -= b  
 x -= lastx  
 y -= lasty  
 else:  
 b -= a  
 lastx -= x  
 lasty -= y  
  
 gcd = b  
 x = lastx  
 y = lasty  
 return g \* gcd, x, y  
  
# Пример использования  
a\_original = a = int(input("Введите число a: "))  
b\_original = b = int(input("Введите число b: "))  
gcd\_value, x, y = extended\_binary\_gcd(a, b)  
print(f"НОД({a}, {b}) = {gcd\_value}")  
print(f"Коэффициенты: x = {x}, y = {y}")  
print(f"Проверка: {a}\*({x}) + {b}\*({y}) = {gcd\_value}")

**Примечание:** Расширенный бинарный алгоритм Евклида является более сложным для реализации по сравнению с классическим расширенным алгоритмом Евклида. Код выше обеспечивает вычисление НОД и коэффициентов x и y для уравнения Bezout’s identity ( ax + by = (a, b) ).

### **Общие примечания:**

* **Ввод данных:** Во всех примерах используется ввод чисел a и b с клавиатуры.
* **Проверка результатов:** В расширенных алгоритмах добавлена проверка того, что найденные коэффициенты действительно удовлетворяют уравнению ( ax + by = (a, b) ).
* **Структура кода:** Каждый алгоритм оформлен как функция для удобства повторного использования.
* **Комментарии:** В коде добавлены комментарии для пояснения шагов алгоритма.

**Если у вас есть дополнительные вопросы или требуется пояснение по какому-либо из алгоритмов, пожалуйста, дайте знать!**