```python
import random
import time
import pandas as pd
import matplotlib.pyplot as plt

# Função iterativa para calcular (a^n) mod p em tempo O(log n).
# Essa operação é fundamental para testar a primalidade com base no Pequeno Teorema de Fermat.
def potencia(a, n, p):
    resultado = 1
    a = a % p  # Atualiza 'a' para o seu valor módulo 'p', se necessário.

    while n > 0:
        if n % 2:
            # Se n for ímpar, multiplica 'resultado' por 'a' e reduz 'n'
            resultado = (resultado * a) % p
            n = n - 1
        else:
            # Se n for par, eleva 'a' ao quadrado e divide 'n' por 2
            a = (a ** 2) % p
            n = n // 2

    return resultado % p

# Função para testar se um número n é primo utilizando o Pequeno Teorema de Fermat.
# Parâmetros:
# - n: número a ser testado.
# - k: número de iterações para aumentar a confiabilidade do teste.
def eh_primo(n, k):
    # Casos triviais: 1 e 4 são compostos; 2 e 3 são primos.
    if n == 1 or n == 4:
        return False
    elif n == 2 or n == 3:
        return True
    else:
        # Repete o teste 'k' vezes para reduzir a probabilidade de erro.
        for _ in range(k):
            # Escolhe aleatoriamente um inteiro 'a' no intervalo [2, n-2].
            a = random.randint(2, n - 2)
            # Se 'a^(n-1) mod n' for diferente de 1, então 'n' é composto.
            if potencia(a, n - 1, n) != 1:
                return False
        # Se passou em todos os testes, retorna True (provavelmente primo).
        return True

# Função para gerar um número primo com aproximadamente 'bits' bits de tamanho.
# Utiliza o teste de Fermat para validar a primalidade.
def gerar_primo(bits, k=5):
    inicio = time.time()  # Marca o tempo inicial de execução.
    while True:
        # Gera um número aleatório de 'bits' bits e garante que seja ímpar (bit menos significativo igual a 1).
        candidato = random.getrandbits(bits) | 1

        # Testa se o número gerado é primo.
        if eh_primo(candidato, k):
            tempo = time.time() - inicio  # Calcula o tempo decorrido.
            return candidato, tempo

# Lista com os tamanhos de bits desejados para geração dos números primos.
```

```python
tamanhos_bits = [40, 56, 80, 128, 168, 224, 256, 512, 1024, 2048, 4096]

# Lista para armazenar os resultados da geração de primos.
resultados = []

# Loop para gerar primos para cada tamanho especificado.
for bits in tamanhos_bits:
    print(f"Gerando número primo de {bits} bits...")
    primo, tempo = gerar_primo(bits)
    resultados.append({
        "Algoritmo": "Fermat",
        "Tamanho do Número (bits)": bits,
        "Número Primo Gerado": primo,
        "Tempo para Gerar (s)": tempo
    })

# Criação da tabela de resultados utilizando a biblioteca pandas.
tabela = pd.DataFrame(resultados)
print("\nTabela de Números Primos Gerados:")
print(tabela)

# Geração de gráfico utilizando a biblioteca matplotlib.
# O gráfico ilustra o tempo de geração em função do tamanho do número (em escala logarítmica).
plt.figure(figsize=(10, 6))
plt.plot(tabela["Tamanho do Número (bits)"], tabela["Tempo para Gerar (s)"], marker='o')
plt.title("Tempo para gerar números primos usando Teste de Fermat")
plt.xlabel("Tamanho do Número (bits)")
plt.ylabel("Tempo (segundos)")
plt.grid(True)
plt.xscale('log')
plt.yscale('log')
plt.show()
# ---------------------------------------------------------------------------
# Pequeno Teorema de Fermat:
# Se n é um número primo, então para todo a, com 1 < a < n-1,
# temos que:
#     a^(n-1) ≡ 1 (mod n)
# ou seja,
#     a^(n-1) % n = 1

# Exemplos:
# - Como 5 é primo, temos que:
#       2^4 ≡ 1 (mod 5),
#       3^4 ≡ 1 (mod 5),
#       4^4 ≡ 1 (mod 5).

# - Como 7 é primo, temos que:
#       2^6 ≡ 1 (mod 7),
#       3^6 ≡ 1 (mod 7),
#       4^6 ≡ 1 (mod 7),
#       5^6 ≡ 1 (mod 7),
#       6^6 ≡ 1 (mod 7).

# Procedimento para testar a primalidade usando o teste de Fermat:
# 1) Repetir k vezes:
#    a) Escolher aleatoriamente um número a no intervalo [2, n-2].
#    b) Se o máximo divisor comum (mdc) de (a, n) for diferente de 1, retornar falso.
```

```
#    c) Se a^(n-1) não for congruente a 1 módulo n, retornar falso.
# 2) Se passar por todos os testes, retornar verdadeiro (provavelmente primo).

### Observações ###:
# - Um valor mais alto de k reduz a probabilidade de erro em números compostos.
# - Para números primos verdadeiros, o teste sempre retorna verdadeiro.
# ---------------------------------------------------------------------------

# This code is contributed by Aanchal Tiwari https://www.geeksforgeeks.org/fermat-method-of-primality-test/

```