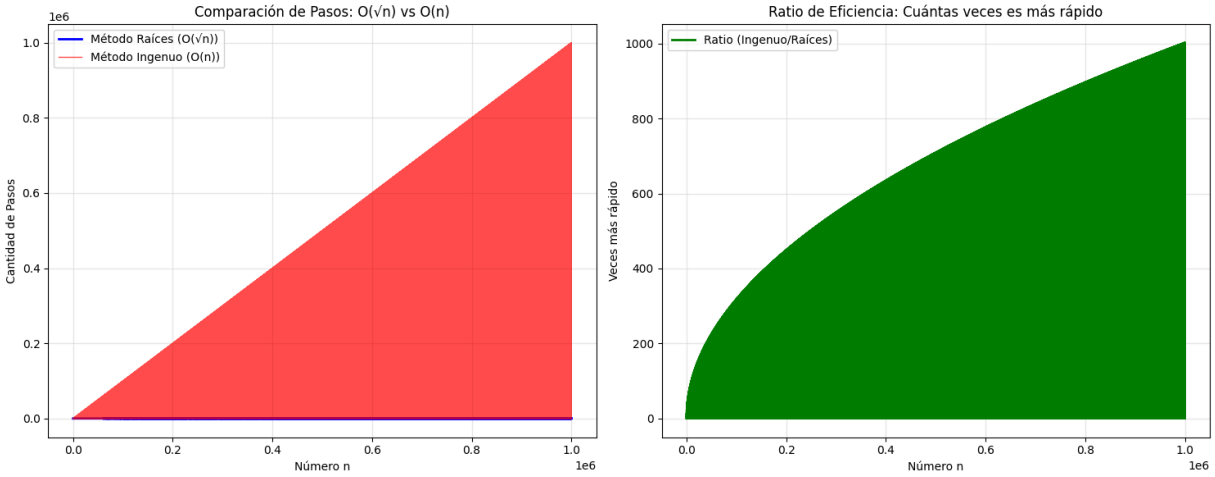
In [13]:
```
from matplotlib import pyplot as plt


def primos_raices(num):

    pasos = 0

    if num < 2:
        return False, pasos
    for i in range(2, int(num ** 0.5) + 1):
        pasos += 1
        if num % i == 0:
            return False, pasos
    return True, pasos


def primos_ingenuos(num):

    pasos = 0

    if num < 2:
        return False, pasos
    for i in range(2, num):
        pasos += 1
        if num % i == 0:
            return False, pasos
    return True, pasos


def obtener_pasos(num):

    pasos_raices = list()
    pasos_ingenuos = list()

    for i in range(2, num):
        resultado_raices = primos_raices(i)
        resultado_ingenuos = primos_ingenuos(i)

        pasos_raices.append(resultado_raices[1])
        pasos_ingenuos.append(resultado_ingenuos[1])

    return pasos_raices, pasos_ingenuos


def calcular_eficiencia(num):
    pasos_raices, pasos_ingenuos = obtener_pasos(num)
    numeros = list(range(2, num))

    total_pasos_raices = sum(pasos_raices)
    total_pasos_ingenuos = sum(pasos_ingenuos)

    eficiencia_relativa = total_pasos_ingenuos / total_pasos_raices
    mejora_porcentual = ((total_pasos_ingenuos - total_pasos_raices) / total
```

```
    print("=== ANÁLISIS DE EFICIENCIA ===")
    print(f"Total de pasos método ingenuo: {total_pasos_ingenuos:,}")
    print(f"Total de pasos método raíces:  {total_pasos_raices:,}")
    print(f"Eficiencia relativa: {eficiencia_relativa:.2f}x más rápido")
    print(f"Mejora porcentual: {mejora_porcentual:.1f}% de reducción en pasc

    return numeros, pasos_raices, pasos_ingenuos

def graficar_eficiencia(num):
    numeros, pasos_raices, pasos_ingenuos = calcular_eficiencia(num)

    # Crear figura con subgráficos
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))

    # Gráfico 1: Comparación directa
    ax1.plot(numeros, pasos_raices, label='Método Raíces (O(√n))', color='bl
    ax1.plot(numeros, pasos_ingenuos, label='Método Ingenuo (O(n))', color='
    ax1.set_title('Comparación de Pasos: O(√n) vs O(n)')
    ax1.set_xlabel('Número n')
    ax1.set_ylabel('Cantidad de Pasos')
    ax1.legend()
    ax1.grid(True, alpha=0.3)

    # Gráfico 2: Ratio de eficiencia
    ratios = [pi/pr if pr > 0 else 0 for pi, pr in zip(pasos_ingenuos, pasos
    ax2.plot(numeros, ratios, label='Ratio (Ingenuo/Raíces)', color='green',
    ax2.set_title('Ratio de Eficiencia: Cuántas veces es más rápido')
    ax2.set_xlabel('Número n')
    ax2.set_ylabel('Veces más rápido')
    ax2.legend()
    ax2.grid(True, alpha=0.3)

    plt.tight_layout()
    plt.show()

def main():
    print("Calculadora de numeros primos")
    # num = int(input("Buscar primos hasta el número:"))

    # resultado_raices = primos_raices(num)
    # resultado_ingenuos = primos_ingenuos(num)

    # if resultado_raices[0] == False:
    #     print("No es un numero primo")
    # else:
    #     print("Es un numero primo")

    graficar_eficiencia()


graficar_eficiencia(1000000)
```

```
=== ANÁLISIS DE EFICIENCIA ===
Total de pasos método ingenuo: 37,567,326,491
Total de pasos método raíces:  67,740,403
Eficiencia relativa: 554.58x más rápido
Mejora porcentual: 99.8% de reducción en pasos
```



In [ ]: