

```

import time
import pyRAPL
import matplotlib.pyplot as plt
from matplotlib.backends.backend_pdf import PdfPages
import pandas as pd

from algoritmos import (
    primos_ate_n,
    fatorial,
    multiplicar_matrizes,
    ordenar_lista,
    busca_binaria,
    calcular_pi
)

# Inicializa pyRAPL
pyRAPL.setup()

def medir_consumo(nome, func, *args, repeticoes=100):
    tempos = []
    energias = []
    potencias = []

    for _ in range(repeticoes):
        meter = pyRAPL.Measurement(nome)

        inicio = time.perf_counter()
        with meter:
            func(*args)
        fim = time.perf_counter()

        # tempo em segundos
        duracao = fim - inicio
        tempos.append(duracao)

        # energia em joules
        try:
            energia_uj = meter.result.pkg[0].energy # versões novas
        except AttributeError:
            energia_uj = meter.result.pkg[0] # sua versão

        energia_j = energia_uj / 1e6
        energias.append(energia_j)

        # potência média
        potencias.append(energia_j / duracao if duracao > 0 else 0)

    return tempos, energias, potencias

if __name__ == "__main__":
    algoritmos = {
        "Primos até N": (primos_ate_n, [10000]),
        "Fatorial de N": (fatorial, [5000]),
        "Multiplicação de matrizes": (multiplicar_matrizes, [100]),
        "Ordenação de lista": (ordenar_lista, [5000]),
        "Busca binária": (busca_binaria, [ordenar_lista(5000), 2500]),
        "Cálculo de PI": (calcular_pi, [2000000]),
    }

    resultados = {}
    for nome, (func, args) in algoritmos.items():
        tempos, energias, potencias = medir_consumo(nome, func, *args, repeticoes=100)

        resultados[nome] = {
            "tempo": tempos,
            "energia": energias,
            "potencia": potencias,
        }

    nomes = list(resultados.keys())

    # Gerar PDF com todos os boxplots

```

```
with PdfPages("resultados_boxplots.pdf") as pdf:

    # Tempo
    plt.figure(figsize=(10,6))
    plt.boxplot([resultados[n]["tempo"] for n in nomes], tick_labels=nomes, show
fliers=False)
    plt.ylabel("Tempo (s)")
    plt.title("Tempo de execução - Boxplot")
    plt.xticks(rotation=30, ha="right")
    plt.tight_layout()
    pdf.savefig()
    plt.close()

    # Energia
    plt.figure(figsize=(10,6))
    plt.boxplot([resultados[n]["energia"] for n in nomes], tick_labels=nomes, sh
owfliers=False)
    plt.ylabel("Energia (J)")
    plt.title("Energia - Boxplot")
    plt.xticks(rotation=30, ha="right")
    plt.tight_layout()
    pdf.savefig()
    plt.close()

    # Potência
    plt.figure(figsize=(10,6))
    plt.boxplot([resultados[n]["potencia"] for n in nomes], tick_labels=nomes, s
howfliers=False)
    plt.ylabel("Potência (W)")
    plt.title("Potência - Boxplot")
    plt.xticks(rotation=30, ha="right")
    plt.tight_layout()
    pdf.savefig()
    plt.close()

print("¿ Relatório gerado: resultados_boxplots.pdf")

# Exportar CSV com médias, medianas e desvio padrão
dados_resumo = []
for nome in nomes:
    for metrica in ["tempo", "energia", "potencia"]:
        serie = pd.Series(resultados[nome][metrica])
        dados_resumo.append({
            "Algoritmo": nome,
            "Métrica": metrica,
            "Média": serie.mean(),
            "Mediana": serie.median(),
            "Desvio Padrão": serie.std()
        })

df_resumo = pd.DataFrame(dados_resumo)
df_resumo.to_csv("resultados_resumo.csv", index=False)
print("¿ CSV gerado: resultados_resumo.csv")
```

```
from .primos import primos_ate_n
from .fatorial import fatorial
from .matrizes import multiplicar_matrizes
from .ordenacao import ordenar_lista
from .busca import busca_binaria
from .pi import calcular_pi
```

```
def busca_binaria(lista, alvo):
    inicio, fim = 0, len(lista) - 1
    while inicio <= fim:
        meio = (inicio + fim) // 2
        if lista[meio] == alvo:
            return meio
        elif lista[meio] < alvo:
            inicio = meio + 1
        else:
            fim = meio - 1
    return -1
```

```
import math

def fatorial(n):
    return math.factorial(n)
```

```
import random

def multiplicar_matrizes(n):
    A = [[random.randint(1, 10) for _ in range(n)] for _ in range(n)]
    B = [[random.randint(1, 10) for _ in range(n)] for _ in range(n)]
    C = [[0] * n for _ in range(n)]
    for i in range(n):
        for j in range(n):
            for k in range(n):
                C[i][j] += A[i][k] * B[k][j]
    return C
```

```
import random

def ordenar_lista(n=1000):
    lista = [random.randint(1, 100000) for _ in range(n)]
    return sorted(lista)
```

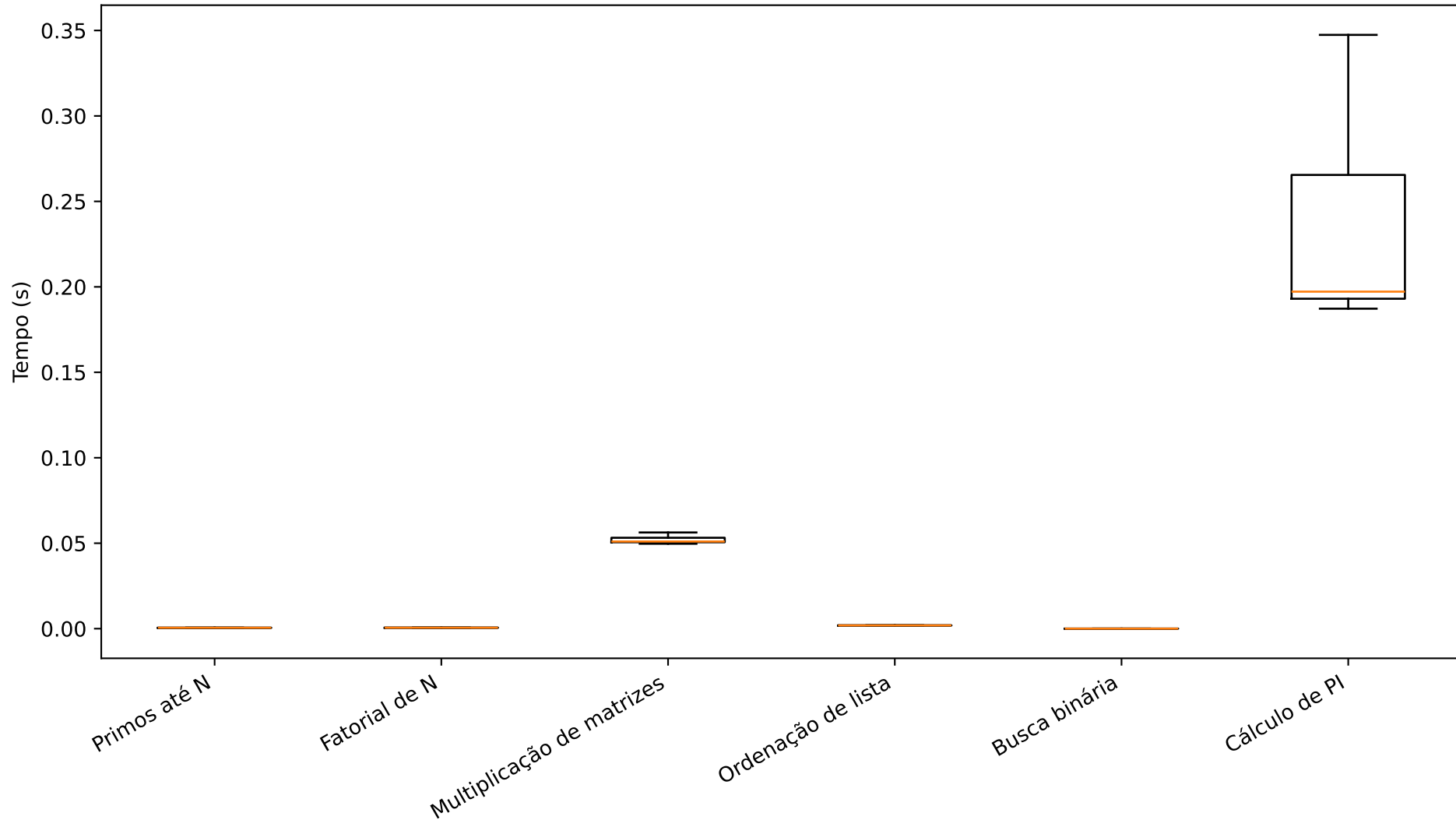
```
import random

def calcular_pi(n=1000000):
    dentro = 0
    for _ in range(n):
        x, y = random.random(), random.random()
        if x*x + y*y <= 1:
            dentro += 1
    return (4 * dentro) / n
```

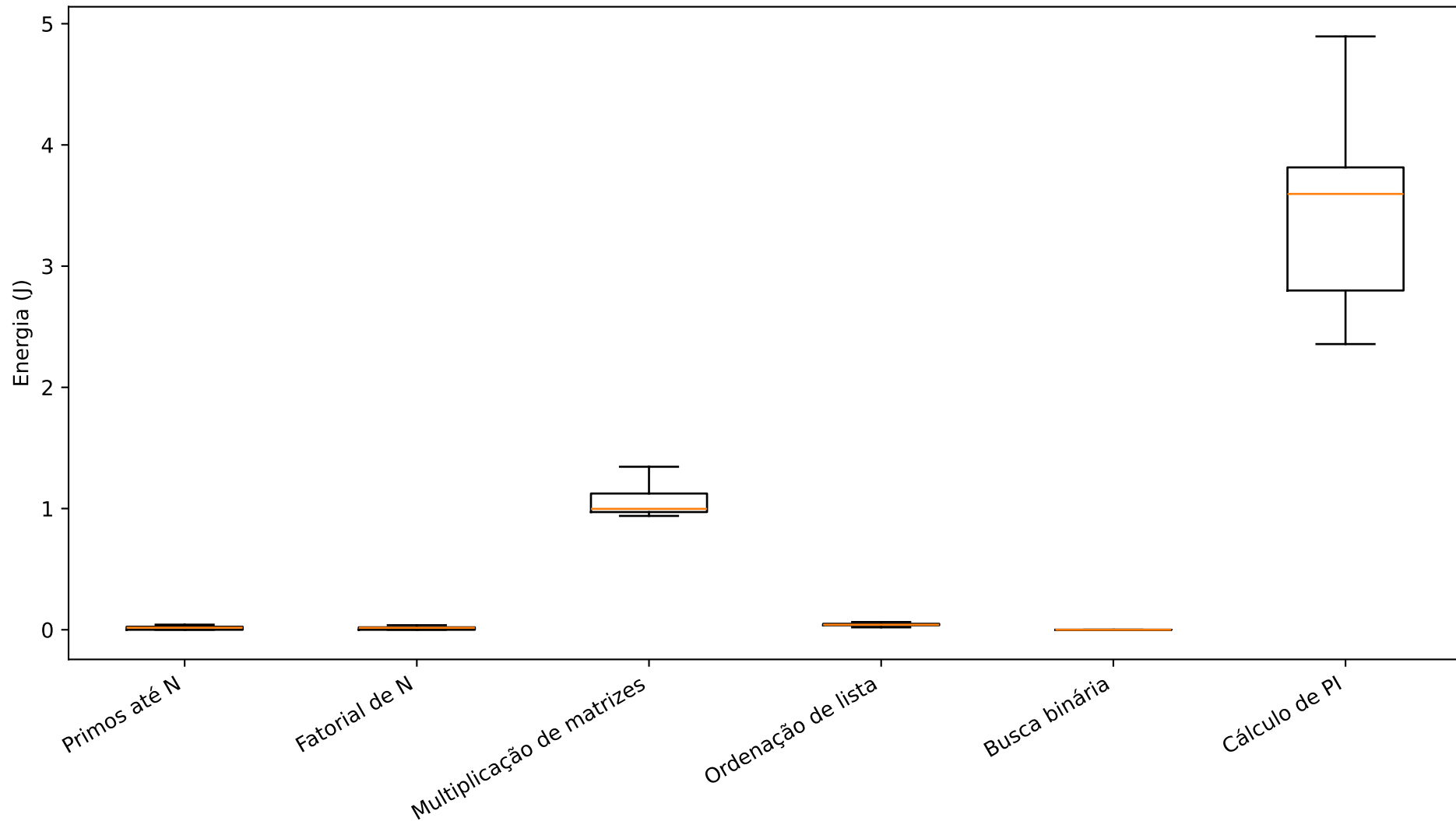


```
def primos_ate_n(n):  
    sieve = [True] * (n + 1)  
    sieve[0:2] = [False, False]  
    for i in range(2, int(n**0.5) + 1):  
        if sieve[i]:  
            for j in range(i*i, n+1, i):  
                sieve[j] = False  
    return [i for i, prime in enumerate(sieve) if prime]
```

Tempo de execução - Boxplot



Energia - Boxplot



Potência - Boxplot

