Implementação dos Algoritmos de Escalonamento

```
import random
from collections import deque

class Processo:
    def __init__(self, id, chegada, duracao, prioridade=0, bilhetes=1):
        self.id = id
        self.chegada = chegada
        self.duracao = duracao
        self.tempo_restante = duracao
        self.finalizado = False
        self.inicio = None
        self.termino = None
        self.prioridade = prioridade
        self.bilhetes = bilhetes

def gerar_processos(n):
    return [Processo(i, random.randint(0, 10), random.randint(1, 10),
random.randint(1, 5), random.randint(1, 10)) for i in range(n)]

def fifo(processos):
    tempo_atual = 0
    for p in sorted(processos, key=lambda x: x.chegada):
        if tempo_atual < p.chegada:
            tempo_atual = p.chegada
        p.inicio = tempo_atual
        tempo_atual += p.duracao
        p.termino = tempo_atual
    return processos

def sjf(processos):
    processos.sort(key=lambda p: (p.chegada, p.duracao))
    tempo_atual = 0
    fila = []
    ordem_execucao = []

    while processos or fila:
        while processos and processos[0].chegada <= tempo_atual:
            fila.append(processos.pop(0))
            fila.sort(key=lambda p: p.duracao)
```

```
        if fila:
            p = fila.pop(0)
            p.inicio = tempo_atual
            tempo_atual += p.duracao
            p.termino = tempo_atual
            ordem_execucao.append(p)
        else:
            tempo_atual += 1

    return ordem_execucao

def round_robin(processos, quantum=2):
    fila = deque(sorted(processos, key=lambda x: x.chegada))
    tempo_atual = 0

    while fila:
        p = fila.popleft()
        if p.inicio is None:
            p.inicio = max(tempo_atual, p.chegada)

        exec_time = min(quantum, p.tempo_restante)
        tempo_atual += exec_time
        p.tempo_restante -= exec_time

        if p.tempo_restante > 0:
            fila.append(p)
        else:
            p.termino = tempo_atual

    return processos

def prioridade(processos):
    tempo_atual = 0
    processos.sort(key=lambda p: (p.chegada, p.prioridade))
    fila = []
    ordem_execucao = []

    while processos or fila:
        while processos and processos[0].chegada <= tempo_atual:
            fila.append(processos.pop(0))
            fila.sort(key=lambda p: p.prioridade)

        if fila:
```

```
            p = fila.pop(0)
            p.inicio = tempo_atual
            tempo_atual += p.duracao
            p.termino = tempo_atual
            ordem_execucao.append(p)
        else:
            tempo_atual += 1

    return ordem_execucao

def loteria(processos):
    tempo_atual = 0
    fila = processos[:]
    random.shuffle(fila)

    while fila:
        p = random.choices(fila, weights=[p.bilhetes for p in fila])[0]
        fila.remove(p)
        if p.inicio is None:
            p.inicio = tempo_atual
        tempo_atual += p.duracao
        p.termino = tempo_atual

    return processos

def calcular_metricas(processos):
    tempos_retorno = [(p.termino - p.chegada) for p in processos]
    tempos_espera = [(p.inicio - p.chegada) for p in processos]
    tempo_medio_retorno = sum(tempos_retorno) / len(processos)
    tempo_medio_espera = sum(tempos_espera) / len(processos)
    print(f"Tempo médio de retorno: {tempo_medio_retorno:.2f}")
    print(f"Tempo médio de espera: {tempo_medio_espera:.2f}")

processos = gerar_processos(5)
print("FIFO:")
calcular_metricas(fifo(processos[:]))
print("\nSJF:")
calcular_metricas(sjf(processos[:]))
print("\nRound Robin:")
calcular_metricas(round_robin(processos[:]))
print("\nPrioridade:")
calcular_metricas(prioridade(processos[:]))
print("\nLoteria:")
```

```
calcular_metricas(loteria(processos[:]))
```