In [ ]:
```python
import random
import math
```

In [ ]:
```python
# Genetic Algorithm
def generate_individual():
    rolls = []
    remaining_length = sum(requests)
    while remaining_length > 0:
        roll_length = min(stock_length, remaining_length)
        rolls.append(roll_length)
        remaining_length -= roll_length
    while len(rolls) > less_than:  # Ensure less than "given number" rolls
        rolls.pop(random.randint(0, len(rolls)-1))
    return rolls

def crossover(parent1, parent2):
    crossover_point = random.randint(1, len(parent1) - 1)
    child1 = parent1[:crossover_point] + parent2[crossover_point:]
    child2 = parent2[:crossover_point] + parent1[crossover_point:]
    return child1, child2

def mutate(individual, mutation_rate=0.01):
    for i in range(len(individual)):
        if random.random() < mutation_rate:
            individual[i] = random.randint(1, stock_length)
    return individual

def genetic_algorithm(population_size=100, generations=100):
    population = [generate_individual() for _ in range(population_size)]
    for _ in range(generations):
        population = sorted(population, key=lambda x: fitness(x), reverse=True)
        next_generation = population[:population_size // 2]
        while len(next_generation) < population_size:
            parent1, parent2 = random.choices(population[:10], k=2)
            child1, child2 = crossover(parent1, parent2)
            next_generation.extend([mutate(child1), mutate(child2)])
        population = next_generation
    return population[0]
```

In [ ]:
```python
# Simulated Annealing
def simulated_annealing(initial_solution, max_iterations=10000, initial_temperature=100.0, cooling_rate=0.95):
    current_solution = initial_solution
    current_fitness = fitness(current_solution)
    best_solution = current_solution
    best_fitness = current_fitness

    for iteration in range(max_iterations):
        temperature = initial_temperature * math.exp(-cooling_rate * iteration)
        if temperature == 0:
            break

        neighbor = mutate(current_solution)
        neighbor_fitness = fitness(neighbor)

        if neighbor_fitness > current_fitness or random.random() < math.exp((neighbor_fitness - current_fitness) / temperature):
            current_solution = neighbor
            current_fitness = neighbor_fitness

        if current_fitness > best_fitness:
            best_solution = current_solution
            best_fitness = current_fitness

    return best_solution
```

In [ ]:
```python
# Hill Climbing
def hill_climbing(initial_solution, max_iterations=10000):
    current_solution = initial_solution
    current_fitness = fitness(current_solution)

    for iteration in range(max_iterations):
        neighbor = mutate(current_solution)
        neighbor_fitness = fitness(neighbor)

        if neighbor_fitness > current_fitness:
            current_solution = neighbor
            current_fitness = neighbor_fitness

    return current_solution
```

In [ ]:
```python
stock_length = 1000
requests = [106, 187, 914, 106, 33, 18, 402, 230, 507, 495, 609, 627, 346, 295, 312, 107, 716, 88, 106, 248, 689, 115, 106, 218, 672, 618, 117, 805, 306, 753, 414, 84, 557, 266, 409, 144, 69,
less_than = 56

# Run algorithms
genetic_solution = len(genetic_algorithm())
sa_solution = len(simulated_annealing(generate_individual()))
hc_solution = len(hill_climbing(generate_individual()))

print("Genetic Algorithm Solution:", genetic_solution)
print("Simulated Annealing Solution:", sa_solution)
print("Hill Climbing Solution:", hc_solution)
```

```
Genetic Algorithm Solution: 49
Simulated Annealing Solution: 49
Hill Climbing Solution: 49
```

In [ ]:
```python
stock_length = 5600
requests = [1520, 2150, 1880, 1520, 2150, 1820, 2150, 2050, 2140, 2140, 1710, 1820, 2150, 1380, 2140, 2150, 1820, 2050, 2100, 1380, 1880, 1880, 1520, 1930, 1710, 2140, 1880, 2050, 1710, 2150,
less_than = 80

# Run algorithms
genetic_solution = len(genetic_algorithm())
sa_solution = len(simulated_annealing(generate_individual()))
hc_solution = len(hill_climbing(generate_individual()))

print("Genetic Algorithm Solution:", genetic_solution)
print("Simulated Annealing Solution:", sa_solution)
print("Hill Climbing Solution:", hc_solution)
```

```
Genetic Algorithm Solution: 73
Simulated Annealing Solution: 73
Hill Climbing Solution: 73
```

In [ ]:
```python
stock_length = 500
requests = [6, 11, 288, 19, 18, 3, 6, 2, 1, 116, 17, 9, 2, 470, 224, 16, 3, 1, 7, 2, 25, 2, 1, 18, 5, 5, 92, 1, 162, 8, 2, 153, 161, 8, 1, 17, 9, 5, 8, 244, 8, 134, 2, 1, 88, 11, 49, 8, 3, 1,
less_than = 115

# Run algorithms
genetic_solution = len(genetic_algorithm())
sa_solution = len(simulated_annealing(generate_individual()))
hc_solution = len(hill_climbing(generate_individual()))

print("Genetic Algorithm Solution:", genetic_solution)
print("Simulated Annealing Solution:", sa_solution)
print("Hill Climbing Solution:", hc_solution)
```

```
Genetic Algorithm Solution: 92
Simulated Annealing Solution: 92
Hill Climbing Solution: 92
```

In [ ]:
```python
stock_length = 100
requests = [22, 7, 5, 3, 28, 2, 14, 5, 32, 29, 74, 24, 67, 3, 35, 1, 6, 66, 30, 70, 65, 1, 12, 47, 33, 36, 99, 54, 4, 10, 18, 11, 5, 23, 48, 30, 44, 5, 51, 13, 13, 13, 2, 11, 60, 22, 5, 13, 10
less_than = 235

# Run algorithms
genetic_solution = len(genetic_algorithm())
sa_solution = len(simulated_annealing(generate_individual()))
hc_solution = len(hill_climbing(generate_individual()))

print("Genetic Algorithm Solution:", genetic_solution)
print("Simulated Annealing Solution:", sa_solution)
print("Hill Climbing Solution:", hc_solution)
```

```
Genetic Algorithm Solution: 205
Simulated Annealing Solution: 205
Hill Climbing Solution: 205
```