```
#cuckoo search(Traffic Signal Optimization)
import numpy as np
from scipy.special import gamma

def fitness_function(x):
    waiting_times = np.array([10 + (x[i] ** 2) / 100 for i in range(len(x))])
    total_waiting_time = np.sum(waiting_times)
    return total_waiting_time

def levy_flight(dim, beta=1.5):
    sigma_u = np.power((gamma(1 + beta) * np.sin(np.pi * beta / 2) /
                        gamma((1 + beta) / 2) * beta * (2 ** (beta - 1))), 1 / beta)
    u = np.random.normal(0, sigma_u, dim)
    v = np.random.normal(0, 1, dim)
    step = u / np.power(np.abs(v), 1 / beta)
    return step

def cuckoo_search(dim, bounds, num_nests, max_iter, p_a=0.25, Lambda=1.5):
    nests = np.random.uniform(bounds[0], bounds[1], (num_nests, dim))
    fitness = np.array([fitness_function(nest) for nest in nests])

    best_idx = np.argmin(fitness)
    best_nest = nests[best_idx]
    best_fitness = fitness[best_idx]

    for iter in range(max_iter):
        new_nests = np.copy(nests)
        for i in range(num_nests):
            step = levy_flight(dim, Lambda)
            new_nests[i] = nests[i] + step
            new_nests[i] = np.clip(new_nests[i], bounds[0], bounds[1])

        new_fitness = np.array([fitness_function(nest) for nest in new_nests])

        for i in range(num_nests):
            if new_fitness[i] < fitness[i]:
                nests[i] = new_nests[i]
                fitness[i] = new_fitness[i]

        if np.random.rand() < p_a:
            random_idx = np.random.randint(num_nests)
            nests[random_idx] = np.random.uniform(bounds[0], bounds[1], dim)
            fitness[random_idx] = fitness_function(nests[random_idx])

        current_best_idx = np.argmin(fitness)
        current_best_fitness = fitness[current_best_idx]

        if current_best_fitness < best_fitness:
            best_fitness = current_best_fitness
            best_nest = nests[current_best_idx]



    return best_nest, best_fitness

dim = 3
bounds = [10, 120]
num_nests = 20
max_iter = 100

best_solution, best_value = cuckoo_search(dim, bounds, num_nests, max_iter)

print("\n--- Best Solution ---")
print("Green Light Timings (seconds):", best_solution)
print("Best Fitness Value (Total Waiting Time):", best_value)
```

```
--- Best Solution ---
Green Light Timings (seconds): [10. 10. 10.]
Best Fitness Value (Total Waiting Time): 33.0
```