In [2]:
```
import sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import cauchy
sys.path.insert(0, './utils')
from utils.BenchmarkFunctions import Ackley, DeJong3, DeJong5, Michalewicz,
sys.path.insert(0, './algos')
from algos import *
sys.path.insert(0, './ge')
from ge import *
```

Housekeeping

In [6]:
```
numDimensions = 20
populationSize = 40
numGenerations = 1000
sphere = Sphere(numDimensions=numDimensions)
schwefel= Schwefel(numDimensions=numDimensions)
#rastrigin = Rastrigin(numDimensions=numDimensions)
```

Implementação do PSO

PSO é um algoritmo relativamente simples. Existe uma população de agentes (particulas) que passa por um processo interativo (gerações) em que, a cada geração, todas as particulas atulizam suas posições (update):

In [7]:
```
def PSO(agentImpl, fitnessFunction, numGenerations, ndims, populationSize, c
    bounds = fitnessFunction.bounds()
    generations = []
    particles = [agentImpl(fitnessFunction, bounds.min, bounds.max, ndims, c
    best = min(particles, key = lambda p : p.fitness)
    for _ in range(numGenerations):
        for p in particles:
            p.update(best)
        best = min(particles, key = lambda p : p.fitness)
        generations.append(best.fitness)
    return {
            "best": best,
            "generations": generations
    }
```

Seguindo esta estrutura base, a implementação da particula pode variar consoante a estratégia utilizada para a atualização da posição no espaço de procura....

PSO Canônico

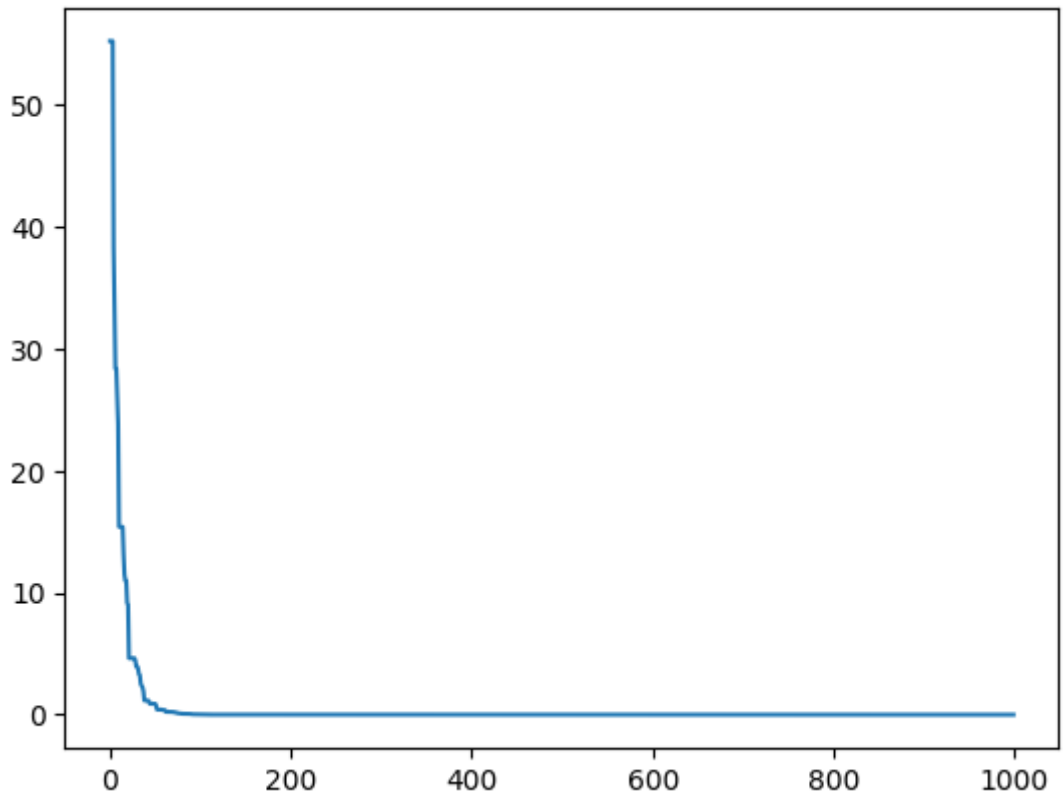
In [8]:
```
class Particle:
    def __init__(self,
                 fitnessFunction,
                 lowerBound,
                 upperBound,
```

```
                nDims,
                options
                ):
        self.fitnessFunction = fitnessFunction
        self.minval = lowerBound
        self.maxval = upperBound
        self.ndims = nDims
        self.inertiaFactor = options["inertiaFactor"] if "inertiaFactor" in
        self.selfConfidence = options["selfConfidence"] if "selfConfidence"
        self.swarmConfidence = options["swarmConfidence"] if "swarmConfidenc
        self.speed = np.array([0.0] * self.ndims)
        self.pos = np.random.uniform(low=self.minval, high = self.maxval, si
        self.best = self.pos
        self.fitness = fitnessFunction.evaluate(self.pos)

    def update(self, best):
        self.speed = (
             self.inertiaFactor * self.speed +
             self.selfConfidence * np.random.rand(self.ndims) * (self.best -
             self.swarmConfidence * np.random.rand(self.ndims) * (best.best
        )
        self.pos = np.clip(self.pos + self.speed, self.minval, self.maxval)
        fit = self.fitnessFunction.evaluate(self.pos)
        if(fit < self.fitness):
            self.fitness = fit
            self.best = self.pos
```

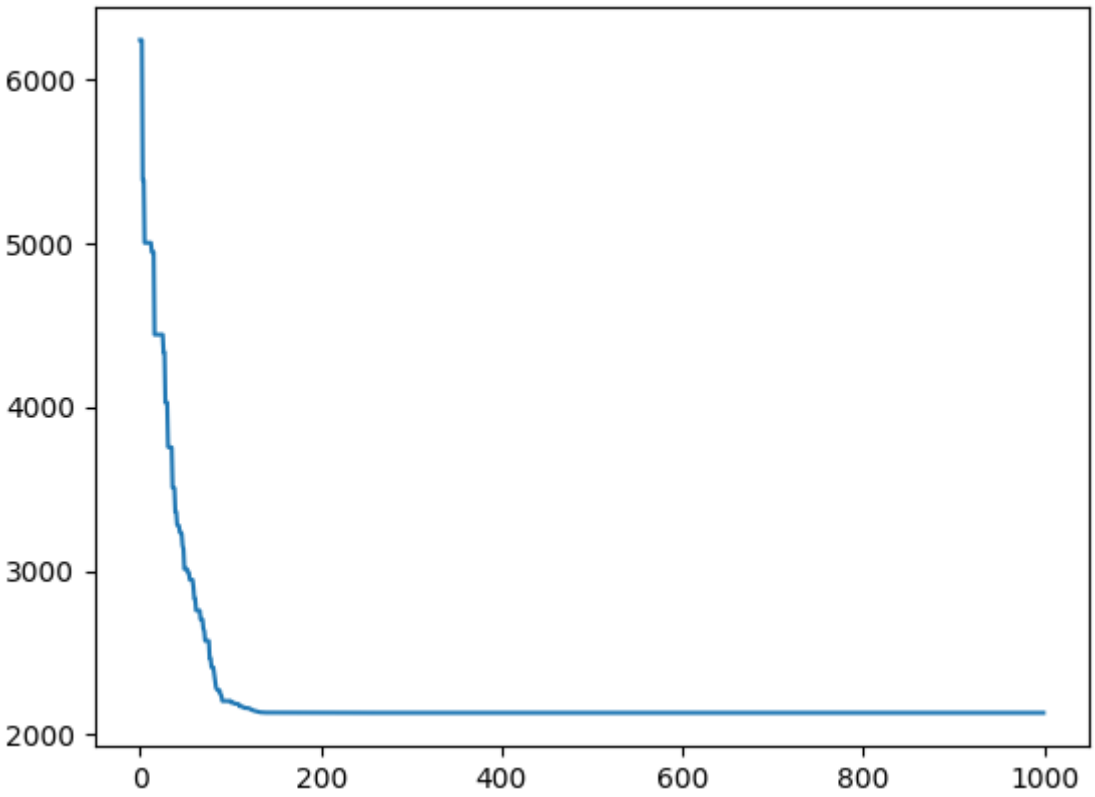
Resultados do PSO Canônico para o Sphere

In [10]:
```
sphereCPSO = PSO(Particle, sphere, numGenerations, numDimensions, population
plt.plot(sphereCPSO["generations"])
plt.show()
```



Resultados do PSO Canônico para o Schwefel

In [11]:
```
schwefelCPSO = PSO(Particle, schwefel, numGenerations, numDimensions, popula
plt.plot(schwefelCPSO["generations"])
plt.show()
```



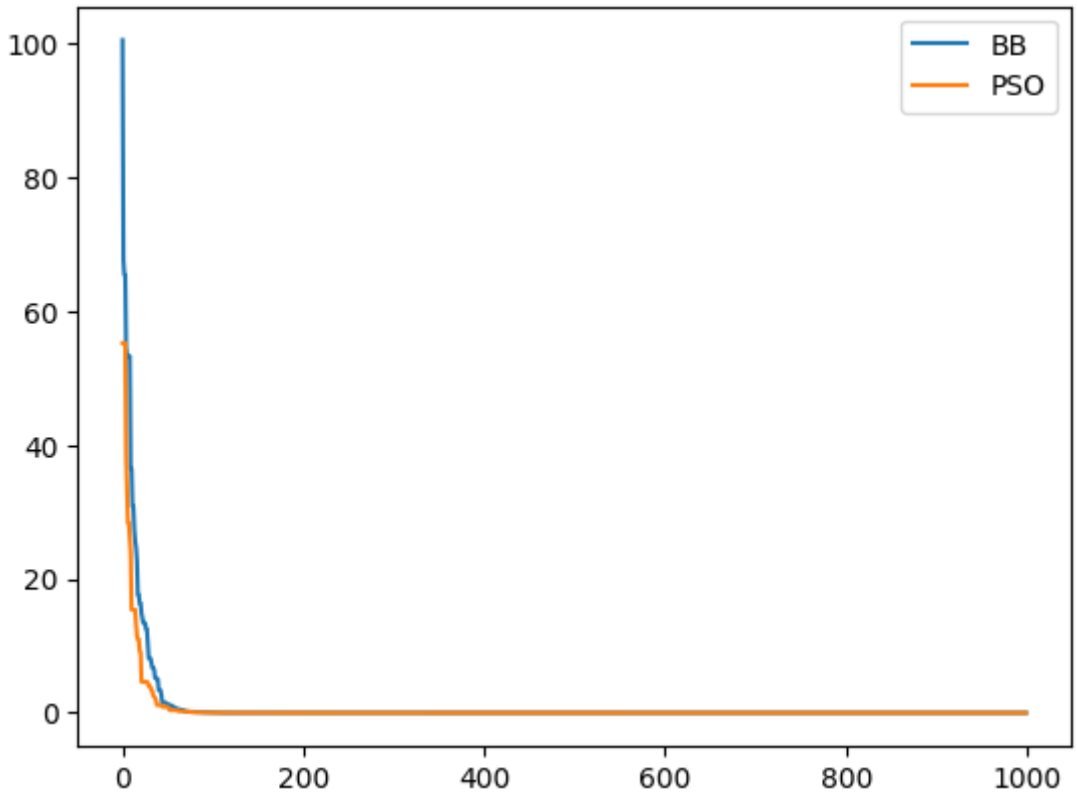
Barebones PSO

In [12]:
```
class BBParticle:
    def __init__(self, fitnessFunction, lowerBound, upperBound, nDims, optio
        self.fitnessFunction = fitnessFunction
        self.minval = lowerBound
        self.maxval = upperBound
        self.ndims = nDims
        self.pos = np.random.uniform(low=self.minval, high = self.maxval, si
        self.best = self.pos
        self.fitness = fitnessFunction.evaluate(self.pos)

    def update(self, best):
        posMean = np.add(self.best, best.best) / 2
        posSd = np.abs(np.subtract(self.best, best.best))
        self.pos = np.clip(np.random.normal(posMean, posSd)  , self.minval,
        fit = self.fitnessFunction.evaluate(self.pos)
        if(fit < self.fitness):
            self.fitness = fit
            self.best = self.pos
```

Resultados do BB vs PSO para o Sphere

In [13]:
```
sphereBB = PSO(BBParticle, sphere, numGenerations, numDimensions, population
plt.plot(sphereBB["generations"], label="BB")
plt.plot(sphereCPSO["generations"], label="PSO")
plt.legend(["BB", "PSO"], loc='upper right')
plt.show()
```

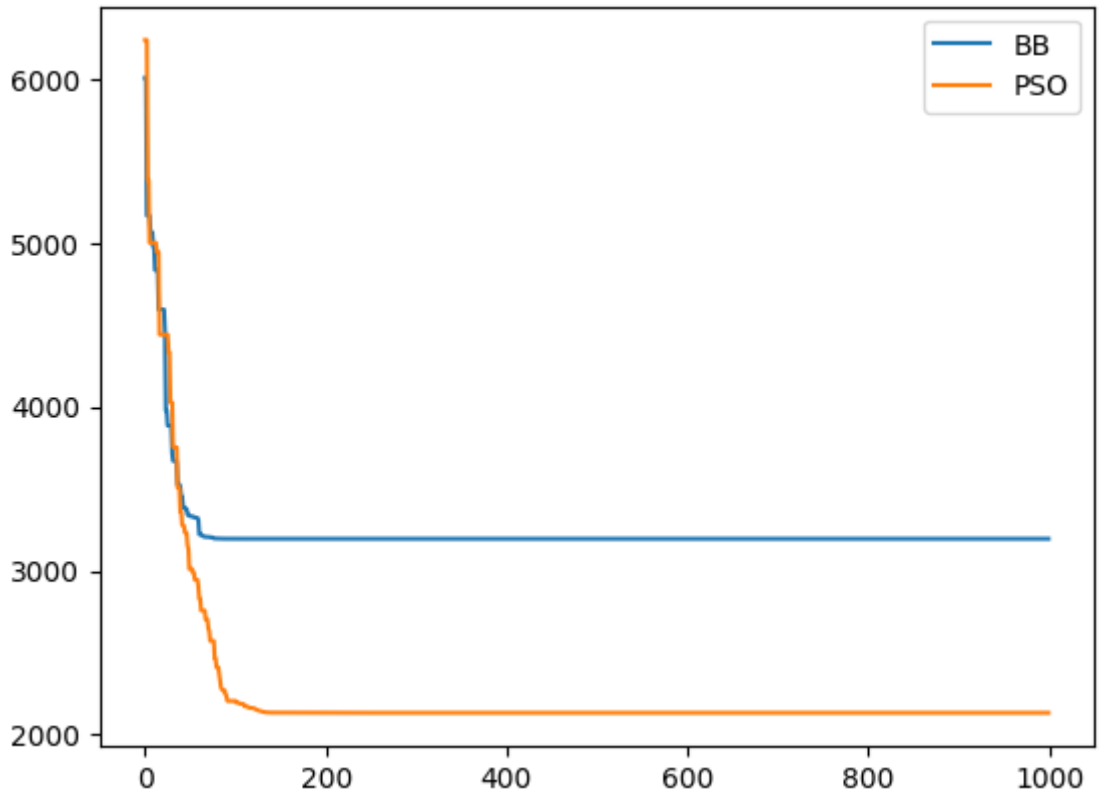


In [17]:
```
df = pd.DataFrame({
  "PSO": [sphereCPSO["best"].fitness],
  "BB": [sphereBB["best"].fitness],
})
print(df)
```

```
            PSO            BB
0  1.793353e-31  2.732798e-33
```

Resultados do BB vs PSO para o Schwefel

In [18]:
```
schwefelBB = PSO(BBParticle, schwefel, numGenerations, numDimensions, popula
plt.plot(schwefelBB["generations"], label="BB")
plt.plot(schwefelCPSO["generations"], label="PSO")
plt.legend(["BB", "PSO"], loc='upper right')
plt.show()
```



In [19]:
```
df = pd.DataFrame({
  "PSO": [schwefelCPSO["best"].fitness],
  "BB": [schwefelBB["best"].fitness],
})
print(df)
```

```
           PSO           BB
0  2133.551162  3194.785944
```

In [20]:
```
grammar = """
<expr> ::= <canonicalPso>

<canonicalPso> ::= PSO(Particle, fitnessFunc, numGenerations, numDimensions,

<inertiaFactor> ::= 0.<int><int> | 1

<selfConfidence> ::= 0.<int><int> | 1.<int><int> | 2

<swarmConfidence> ::= 0.<int><int> | 1.<int><int> | 2

<populationSize> ::= 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100

<int> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
"""
```

In [22]:
```
def siSphere(phenotype):
    fitnessFunc = sphere
    res = eval(phenotype)
    return res["best"].fitness

optPSOSphere = GE(
    grammar=grammar,
    fitnessFunction=siSphere,
    populationSize=50,
    numGenerations=50,
    mutationProbability=0.1,
```

```
    crossoverProbability=0.5,
    genotypeLength=10
).evolve("<expr>")

df = pd.DataFrame({
  "PSO": [sphereCPSO["best"].fitness],
  "BB": [sphereBB["best"].fitness],
  "PSO OPT": [optPSOSphere[0].fitness],
})
print(df)
```

```
            PSO            BB       PSO OPT
0  1.793353e-31  2.732798e-33  1.967296e-60
```

In [23]:
```
print(optPSOSphere[0].phenotype)
```

```
PSO(Particle, fitnessFunc, numGenerations, numDimensions, 90,  {"inertiaFac
tor":0.77, "selfConfidence":1.22, "swarmConfidence":1.00})
```

In [30]:
```
def siSchwefel(phenotype):
    fitnessFunc = schwefel
    res = eval(phenotype)
    return res["best"].fitness

optPSOSchwefel = GE(
    grammar=grammar,
    fitnessFunction=siSchwefel,
    populationSize=50,
    numGenerations=50,
    mutationProbability=0.1,
    crossoverProbability=0.5,
    genotypeLength=10
).evolve("<expr>")

df = pd.DataFrame({
  "PSO": [schwefelCPSO["best"].fitness],
  "BB": [schwefelBB["best"].fitness],
  "PSO OPT": [optPSOSchwefel[0].fitness],
})
print(df)
```

```
           PSO           BB      PSO OPT
0  2133.551162  3194.785944  1817.582669
```

In [31]:
```
print(optPSOSchwefel[0].phenotype)
```

```
PSO(Particle, fitnessFunc, numGenerations, numDimensions, 60,  {"inertiaFac
tor":0.11, "selfConfidence":2, "swarmConfidence":2})
```

In [46]:
```
autoGrammar = """
<expr> ::=  <bbPso> | <canonicalPso>

<bbPso> ::= PSO(BBParticle, fitnessFunc, numGenerations, numDimensions, <pop

<canonicalPso> ::= PSO(Particle, fitnessFunc, numGenerations, numDimensions,

<inertiaFactor> ::= 0.<int><int> | 1

<selfConfidence> ::= 0.<int><int> | 1.<int><int> | 2

<swarmConfidence> ::= 0.<int><int> | 1.<int><int> | 2

<populationSize> ::= 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100
```

```
<int> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
"""
```

In [50]:
```
optPSOBBSchwefel = GE(
    grammar=autoGrammar,
    fitnessFunction=siSchwefel,
    populationSize=50,
    numGenerations=50,
    mutationProbability=0.1,
    crossoverProbability=0.5,
    genotypeLength=10
).evolve("<expr>")

df = pd.DataFrame({
  "PSO": [schwefelCPSO["best"].fitness],
  "BB": [schwefelBB["best"].fitness],
  "AUTO": [optPSOBBSchwefel[0].fitness],
})
print(df)
```

```
           PSO           BB         AUTO
0  2133.551162  3194.785944  1190.451839
```

In [51]:
```
print(optPSOBBSchwefel[0].phenotype)
```

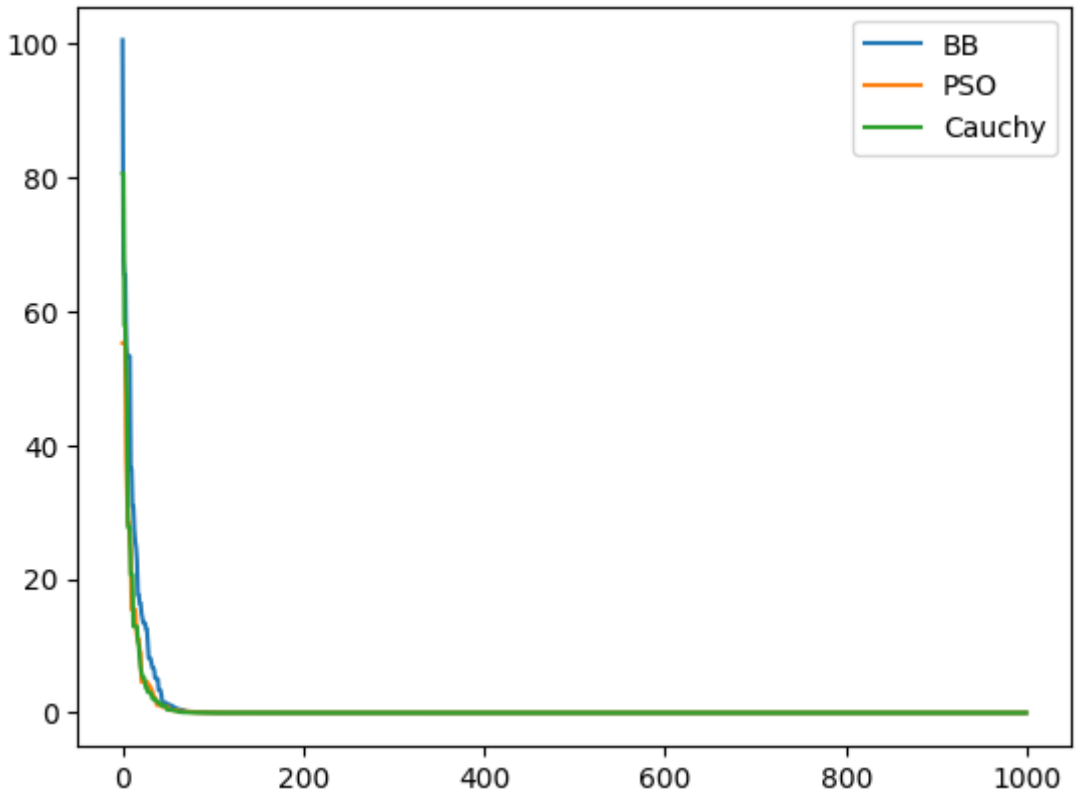
```
PSO(BBParticle, fitnessFunc, numGenerations, numDimensions, 90)
```

Cauchy PSO

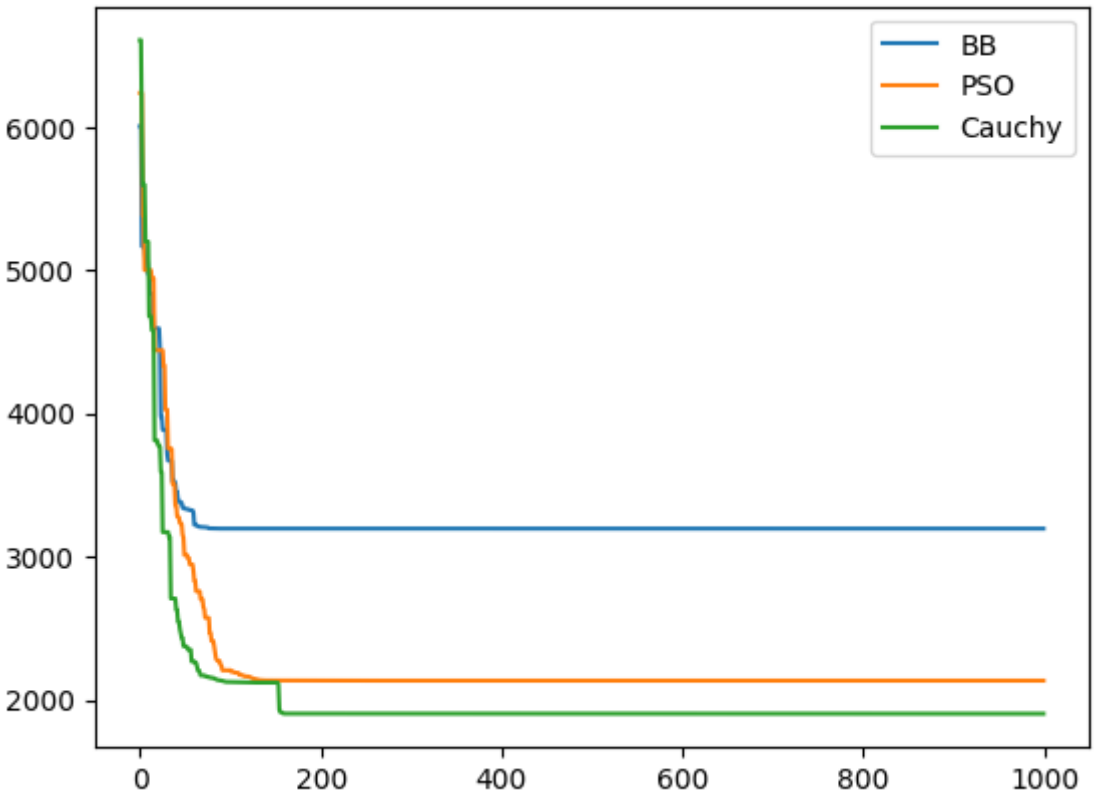
In [52]:
```
class CauchyParticle:
    def __init__(self, fitnessFunction, lowerBound, upperBound, nDims, optic
        self.fitnessFunction = fitnessFunction
        self.minval = lowerBound
        self.maxval = upperBound
        self.ndims = nDims
        self.pos = np.random.uniform(low=self.minval, high = self.maxval, si
        self.best = self.pos
        self.fitness = fitnessFunction.evaluate(self.pos)
        self.coolingFactor = options["coolingFactor"] if "coolingFactor" in

    def update(self, best):
        posMean = np.add(self.best, best.best) / 2
        posSd = np.abs(np.subtract(self.best, best.best))
        pos = cauchy.rvs(loc=posMean, scale=posSd*self.coolingFactor, size=s
        self.pos = np.clip(pos, self.minval, self.maxval)
        fit = self.fitnessFunction.evaluate(self.pos)
        if(fit < self.fitness):
            self.fitness = fit
            self.best = self.pos
```

In [55]:
```
sphereCauchy = PSO(CauchyParticle, sphere, numGenerations, numDimensions, po
plt.plot(sphereBB["generations"], label="BB")
plt.plot(sphereCPSO["generations"], label="PSO")
plt.plot(sphereCauchy["generations"], label="Cauchy")
plt.legend(["BB", "PSO", "Cauchy"], loc='upper right')
plt.show()
```



In [56]:
```
schwefelCauchy = PSO(CauchyParticle, schwefel, numGenerations, numDimensions
plt.plot(schwefelBB["generations"], label="BB")
plt.plot(schwefelCPSO["generations"], label="PSO")
plt.plot(schwefelCauchy["generations"], label="Cauchy")
plt.legend(["BB", "PSO", "Cauchy"], loc='upper right')
plt.show()
```



In [57]:
```
autoGrammar = """
<expr> ::=  <bbPso> | <cauchyPso> | <canonicalPso>

<bbPso> ::= PSO(BBParticle, fitnessFunc, numGenerations, numDimensions, <pop
```

```
<cauchyPso> ::= PSO(CauchyParticle, fitnessFunc, numGenerations, numDimensio

<canonicalPso> ::= PSO(Particle, fitnessFunc, numGenerations, numDimensions,

<inertiaFactor> ::= 0.<int><int> | 1

<selfConfidence> ::= 0.<int><int> | 1.<int><int> | 2

<swarmConfidence> ::= 0.<int><int> | 1.<int><int> | 2

<populationSize> ::= 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100

<int> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
"""
```

In [58]:
```
optAutoPSOSchwefel = GE(
    grammar=autoGrammar,
    fitnessFunction=siSchwefel,
    populationSize=50,
    numGenerations=50,
    mutationProbability=0.1,
    crossoverProbability=0.5,
    genotypeLength=10
).evolve("<expr>")

df = pd.DataFrame({
  "PSO": [schwefelCPSO["best"].fitness],
  "BB": [schwefelBB["best"].fitness],
  "Cauchy": [schwefelCauchy["best"].fitness],
  "AUTO W/ BB": [optPSOBBSchwefel[0].fitness],
  "AUTO ALL": [optAutoPSOSchwefel[0].fitness],
})
print(df)
```

```
           PSO           BB       Cauchy   AUTO W/ BB     AUTO ALL
0  2133.551162  3194.785944  1902.598906  1190.451839  1068.979385
```

In [59]:
```
print(optAutoPSOSchwefel[0].phenotype)
```

```
PSO(CauchyParticle, fitnessFunc, numGenerations, numDimensions, 10,  {"cool
ingFactor":0.44})
```

In [ ]: