**LABTASK 7**

**NAME: SARIM AMIR**

**SID: 63686**

import numpy as np

import datetime

# generate new gen

def create\_gen(\_target):

random\_number = np.random.randint(32, 126, size=\_target)

gen = ''.join([chr(i) for i in random\_number])

return gen

# calculate fitness of gen

def calculate\_fitness(gen, target, \_target):

fitness = 0

for i in range (\_target):

if gen[i:i+1] == target[i:i+1]:

fitness += 1

fitness = fitness / \_target \* 100

return fitness

# create population

def create\_population(target, max\_population, \_target):

New\_Population = {}

for i in range(max\_population):

gen = create\_gen(\_target)

genfitness = calculate\_fitness(gen, target, \_target)

New\_Population[gen] = genfitness

return New\_Population

# selection process

def selection(New\_Population):

pop = dict(New\_Population)

parent = {}

for i in range(2):

gen = max(pop, key=pop.get)

genfitness = pop[gen]

parent[gen] = genfitness

if i == 0:

del pop[gen]

return parent

# crossover

def crossover(parent, target, \_target):

child = {}

cp = round(len(list(parent)[0])/2)

for i in range(2):

gen = list(parent)[i][:cp] + list(parent)[1-i][cp:]

genfitness = calculate\_fitness(gen, target, \_target)

child[gen] = genfitness

return child

# mutation

def mutation(child, target, mutation\_rate, \_target):

mutant = {}

for i in range(len(child)):

data = list(list(child)[i])

for j in range(len(data)):

if np.random.rand(1) <= mutation\_rate:

ch = chr(np.random.randint(32, 126))

data[j] = ch

gen = ''.join(data)

genfitness = calculate\_fitness(gen, target, \_target)

mutant[gen] = genfitness

return mutant

# create new population with new best gen

def regeneration(mutant, New\_Population):

for i in range(len(mutant)):

bad\_gen = min(New\_Population, key=New\_Population.get)

del New\_Population[bad\_gen]

New\_Population.update(mutant)

return New\_Population

# get best gen in a population

def bestgen(parent):

gen = max(parent, key=parent.get)

return gen

# get best fitness in a population

def bestfitness(parent):

fitness = parent[max(parent, key=parent.get)]

return fitness

# display function

def display(parent):

timeDiff=datetime.datetime.now()-startTime

print('{}\t{}%\t{}'.format(bestgen(parent), round(bestfitness(parent), 2), timeDiff))

print("Enter Target Number from the following genes {1,2,3,4,5,6,7,8,9,0}")

target = input("Enter Number: ")

max\_population = int(input("Enter Population Size: "))

mutation\_rate = 0.2

print('Target Word :', target)

print('Max Population :', max\_population)

print('Mutation Rate :', mutation\_rate)

\_target = len (target)

startTime=datetime.datetime.now()

New\_Population = create\_population(target, int(max\_population), \_target)

parent = selection(New\_Population)

print('==============================================')

print("POPULATION")

print('==============================================')

print(New\_Population )

child = crossover(parent, target, \_target)

mutant = mutation(child, target, float(mutation\_rate), \_target)

print('==============================================')

print("SELECTION ")

print('==============================================')

print(parent)

print('==============================================')

print("MUTATION ")

print('==============================================')

print(mutant )

print('==============================================')

print("REGENERATION")

print('==============================================')

print('{}\t{}\t{}'.format('The Best','Fitness','Time'))

print('==============================================')

while 1:

child = crossover(parent, target, \_target)

mutant = mutation(child, target, float(mutation\_rate), \_target)

if bestfitness(parent) >= bestfitness(mutant):

continue

New\_Population = regeneration(mutant, New\_Population)

parent = selection(New\_Population)

display(parent)

if bestfitness(mutant) >= 100:

break

**OUTPUT:**

