

**PAF-Karachi Institute of Economics and Technology College of Computing and Information Sciences**

**Artificial Intelligence**

**Assignment - 2**

**IMPORTANT INSTRUCTIONS**:

Read the following Instructions carefully:

* Attempt all question using python.
* After attempting all question, add screenshot of each question’s solution with output.
* Must attempt all questions by yourself, in case of copied solutions your assignment will marked ZERO.
* Arrange questions and their subsequent parts in sequence.
* Submit on LMS and GCR before deadline: 10/5/2022.
* Hardcopy submission deadline: 11/5/2022

# Save your word document with name: [Student Name Student ID]

**[5 mark]**

1. Perform Histogram Equalization on the following image



import cv2

import numpy as np

from matplotlib import pyplot as plt

img = cv2.imread('img.png',0)

hist,bins = np.histogram(img.flatten(),256,[0,256])

cdf = hist.cumsum()

cdf\_normalized = cdf \* hist.max()/ cdf.max()

plt.plot(cdf\_normalized, color = 'b')

plt.hist(img.flatten(),256,[0,256], color = 'r')

plt.xlim([0,256])

plt.legend(('cdf','histogram'), loc = 'upper left')

plt.show()

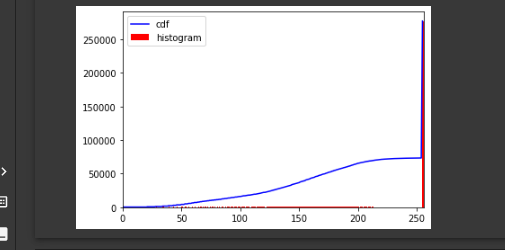
cdf\_m = np.ma.masked\_equal(cdf,0)

cdf\_m = (cdf\_m - cdf\_m.min())\*255/(cdf\_m.max()-cdf\_m.min())

cdf = np.ma.filled(cdf\_m,0).astype('uint8')

img2 = cdf[img]

**OUTPUT:**



# Save your word document with name: [Student Name Student ID]

**[5 mark]**

2. Perform Genetic algorithm for guess password test.

import random

import numpy as np

#import matplotlib.pyplot as plt

import time

def fitness(population):

fitness\_scores=[]

for chromosome in population:

matches=0

for index in range(password\_length):

if secret\_password[index]== chromosome[index]:

matches+= 1

result=[chromosome,matches]

fitness\_scores.append(result)

return fitness\_scores

def select\_parents(fitness\_scores):

parents\_list=[]

for chromosome in sorted(fitness\_scores, key=lambda x: x[1], reverse = True)[:num\_parents]:

parents\_list.append(chromosome[0])

return(parents\_list)

def breed(parent1,parent2):

child=[]

parent1=parents[0]

parent2=parents[1]

geneA=int(random.random() \* password\_length)

geneB=int(random.random() \* password\_length)

startGene=min(geneA, geneB)

endGene=max(geneA, geneB)

for i in range(0,password\_length):

if (i < startGene) or (i > endGene):

child.append(parent1[i])

else:

child.append(parent2[i])

return child

# breeding and elitism

def create\_children(parents\_pool):

children=[]

num\_new\_children = len(population) - elite\_size

for i in range(0,elite\_size):

children.append(parents\_pool[i])

for i in range(0,num\_new\_children):

parent1=parents\_pool[int(random.random() \* len(parents\_pool))]

parent2=parents\_pool[int(random.random() \* len(parents\_pool))]

children.append(breed(parent1,parent2))

return children

def mutation(children\_set):

for i in range(len(children\_set)):

if random.random() > 0.1:

continue

else:

mutated\_position = int(random.random() \* password\_length)

mutation = int(round(random.uniform(lowerbound,upper\_bound+1),0))

children\_set[i][mutated\_position] = mutation

return children\_set

password\_length=6

lowerbound=0

upper\_bound=9

population\_size=6

num\_parents=2

elite\_size=2

secret\_password=[]

for x in range(password\_length):

secret\_password.append(int(round(random.uniform(lowerbound,upper\_bound),0)))

print(secret\_password)

population=[]

for i in range(population\_size):

chromosome=[]

for x in range(password\_length):

chromosome.append(int(round(random.uniform(lowerbound,upper\_bound),0)))

population.append(chromosome)

success=[]

generations=0

t0=time.time()

while True:

fitness\_scores=fitness(population)

success.append(max([i[1] for i in fitness\_scores]))

if max([i[1] for i in fitness\_scores]) == password\_length:

print("Cracked in {} generations, and took {} seconds! \nSecret password = {} \nDiscovered password = {}".format(generations,time.time() - t0,secret\_password,[i[0] for i in fitness\_scores if i[1] == password\_length][0]))

break

parents=select\_parents(fitness\_scores)

children=create\_children(parents)

population=mutation(children)

generations+=1

**OUTPUT:**

