NAME **ADNAN**

ID **13507**

FINAL TERM ASSIGNMENT.

Q1.

ANS.

GENETIC ALGORITHM USING PYTHON:

- First of all we define genetic Algorithm is a heuristic search method and It is used for finding optimized solutions to search problems based on the theory of natural selection and **evolutionary** biology.
- Now we taking example of Genetic Algorithm…

Crossover Operator: This represents mating between individuals. Two individuals are selected using selection operator and crossover sites are chosen randomly. Then the genes at these crossover sites are exchanged thus creating a completely new individual (offspring). For example –

PARENT1…

A	**B**	**C**	**D**	**E**	F	G	H

PARENT 2…

F	**G**	**H**	A	D	B	E	**A**

Offspring…

F	G	H	**B**	**C**	**D**	**E**	A

Mutation Operator: The key idea is to insert random genes in offspring to maintain the diversity in population to avoid the premature convergence. For example –

Before Mutation…

F	G	H	B	C	D	E	A

After Mutation…

F	**G**	M	**B**	**C**	**D**	**E**	**N**

Example problem and solution using Genetic Algorithms in python:

Python3 program to create target string, starting from

random string using Genetic Algorithm

import random

Number of individuals in each generation

POPULATION_SIZE = 100

Valid genes

GENES = '''abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOP

QRSTUVWXYZ 1234567890, .-;:_!"#%&/()=?@${[]}'''

Target string to be generated

TARGET = "I love GeeksforGeeks"

class Individual(object):

'''

Class representing individual in population

'''

def __init__(self, chromosome):

self.chromosome = chromosome

self.fitness = self.cal_fitness()

@classmethod

def mutated_genes(self):

'''

create random genes for mutation

'''

global GENES

gene = random.choice(GENES)

return gene

@classmethod

def create_gnome(self):

'''

create chromosome or string of genes

'''

global TARGET

gnome_len = len(TARGET)

return [self.mutated_genes() for _ in range(gnome_len)]

def mate(self, par2): '''

Perform mating and produce new offspring "

chromosome for offspring

child_chromosome = []

```
for gp1, gp2 in zip(self.chromosome, par2.chromosome):

# random probability

prob = random.random()

# if prob is less than 0.45, insert gene

# from parent 1

if prob < 0.45:

child_chromosome.append(gp1)

# if prob is between 0.45 and 0.90, insert

# gene from parent 2

elif prob < 0.90:

child_chromosome.append(gp2)

# otherwise insert random gene(mutate),

# for maintaining diversity

else:

child_chromosome.append(self.mutated_genes())

# create new Individual(offspring) using

# generated chromosome for offspring

return Individual(child_chromosome)

def cal_fitness(self):

Calculate fittness score, it is the number of

characters in string which differ from target

string.

'''

global TARGET

fitness = 0
```

```
for gs, gt in zip(self.chromosome, TARGET):

if gs != gt: fitness+= 1

return fitness

# Driver code

def main():

global POPULATION_SIZE

#current generation

generation = 1

found = False

population = []

# create initial population

for _ in range(POPULATION_SIZE):

gnome = Individual.create_gnome()

population.append(Individual(gnome))

while not found:

# sort the population in increasing order of fitness score

population = sorted(population, key = lambda x:x.fitness”

# if the individual having lowest fitness score ie.

# 0 then we know that we have reached to the target

# and break the loop

if population[0].fitness <= 0:

found = True


# Otherwise generate new offsprings for new generation

new_generation = []
```

```
# Perform Elitism, that mean 10% of fittest population
# goes to the next generation
s = int((10*POPULATION_SIZE)/100)
new_generation.extend(population[:s])

# From 50% of fittest population, Individuals
# will mate to produce offspring
s = int((90*POPULATION_SIZE)/100)
for _ in range(s):
parent1 = random.choice(population[:50])
parent2 = random.choice(population[:50])
child = parent1.mate(parent2)
new_generation.append(child)
population = new_generation
print("Generation: {}\tString: {}\tFitness: {}".\
format(generation,
"".join(population[0].chromosome),
population[0].fitness))

generation += 1


print("Generation: {}\tString: {}\tFitness: {}".\
format(generation,
```

"".join(population[0].chromosome),

population[0].fitness))

if __name__ == '__main__':

main ()

so the output will be

Output:

```
Generation: 1    String: tO{"-?=jH[k8=B4]Oe@}    Fitness: 18
Generation: 2    String: tO{"-?=jH[k8=B4]Oe@}    Fitness: 18
Generation: 3    String: .#lRWf9k_Ifslw #O$k_    Fitness: 17
Generation: 4    String: .-1Rq?9mHqk3Wo]3rek_    Fitness: 16
Generation: 5    String: .-1Rq?9mHqk3Wo]3rek_    Fitness: 16
Generation: 6    String: A#ldW) #lIkslw cVek)    Fitness: 14
Generation: 7    String: A#ldW) #lIkslw cVek)    Fitness: 14
Generation: 8    String: (, o x _x%Rs=, 6Peek3   Fitness: 13
                         .
                         .
                         .
Generation: 29    String: I lope Geeks#o, Geeks   Fitness: 3
Generation: 30    String: I loMe GeeksfoBGeeks    Fitness: 2
Generation: 31    String: I love Geeksfo0Geeks    Fitness: 1
Generation: 32    String: I love Geeksfo0Geeks    Fitness: 1
Generation: 33    String: I love Geeksfo0Geeks    Fitness: 1
Generation: 34    String: I love GeeksforGeeks    Fitness: 0
```

Q3.

ANS.

Here we have some data of KNN Algorithm example:

Name	Acid Durability	Strength	class
Type1	7	7	Bad
Type2	7	4	Bad
Type3	3	4	Good
Type4	1	4	Good

Now here we include Test where Data Durability =3, strength=7 without any survey how we say its include in which class" Bad or Good".

1. Determine parameter where k= number of nearest neighbors.

   Suppose where k=3

   So calculate the distance between the query- instance and all the training data.

Name	Acid Durability	Strength	Square Distance to Query instance (3,7)
Type1	7	7	(7-3)2+(7-7)2=16
Type2	7	4	(7-3)2+(4-7)2=25
Type3	3	4	(3-3)2+(4-7)2=9
Type4	1	4	(1-3)2+(4-7)2=13

Now sort the distance and find the nearest neighbors on the based on the k-th minimum distance

Acid strength Durability (seconds)	Strength	Square distance Rank to query minimum (kg/square instance distance meter)(3,7)	It is include nearest neighbors
7	7	(7-3)2+(7-7)2=16 3	Yes
7	4	(7-3)2+)(4-7)2=25 4	No
3	4	(3-3)2+(4-7)2= 9 1	Yes
1	4	(1-3)2+(4-7)2 = 13 2	Yes

Gather than category y of the nearest neighbors in the second row last column that the category of nearest (y) is not include because the rank of tis data is more than (k=3).

Name	Acid Durability	Strength	Square distance rank to query minimum in 3 (kg/square instance distance meter)(3,7)	Is it included Nearest neighbors ?	Y=category of nearest neighbors
Type1	7	7	(7-3)2+(7-7)2=16 3	Yes	Bad
Type2	7	4	(7-3)2+)(4-7)2=25 4	No	-
Type3	3	4	(3-3)2+(4-7)2= 9 1	Yes	Good

Type4	1	4	(1-3)2+(4-7)2 = 13 2	yes	Good

So we have 2 good and 1 bad, since 2>1 then we conclude that a new Test Data with Acid durability =3 and strength =7 is **include in class Good.**

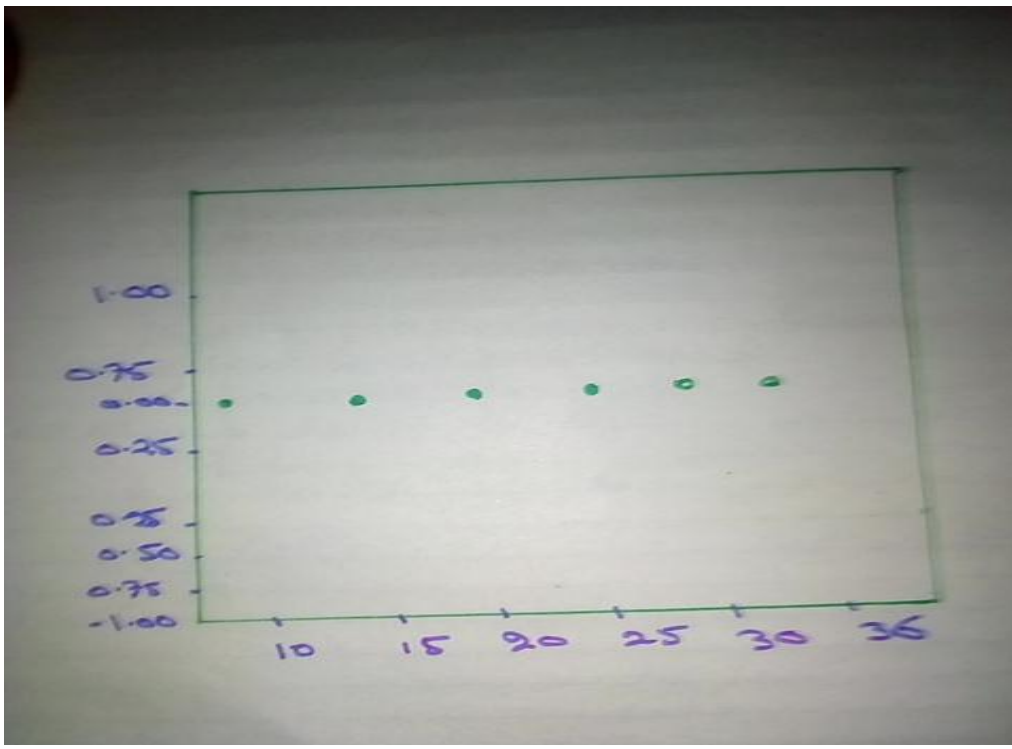
Q4.

ANS.

hierarchical Clustering:

- Dimensional data set { 8,12,22,30,36)

Lets first we visualize the above data set data

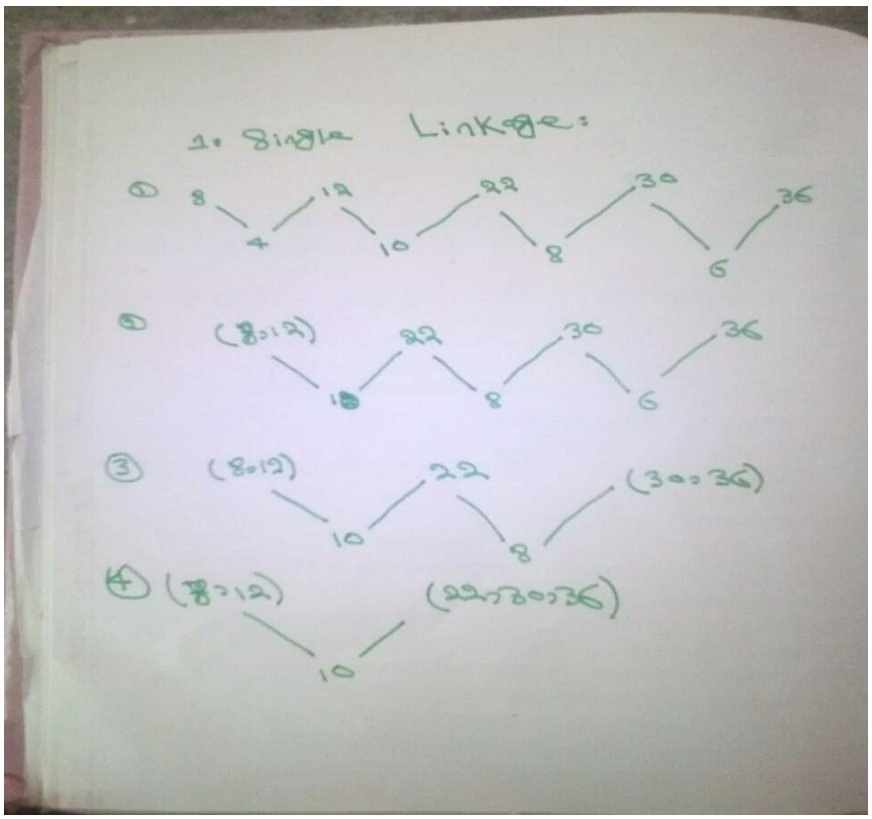


1. The first two points (8,12) are close to each other and should be in the same cluster.
2. Also the last two points (30,36) are close to each other and should be in the same cluster.
3. Cluster of the center point (22) is not easy to conclude.

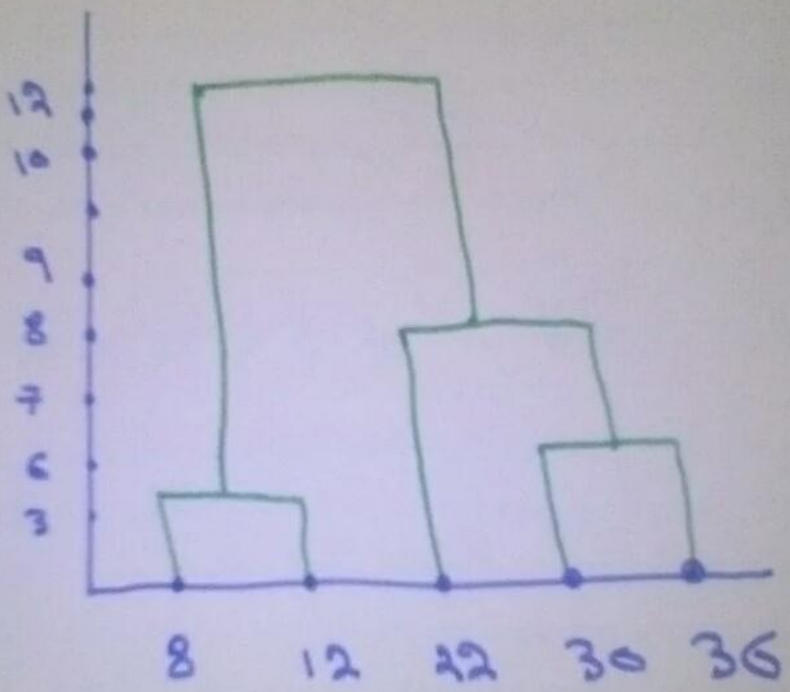
Now solved the above example in using both type of agglomerative hierarchical clustering method.

1. Single Linkage:

- In single link hierarchical clustering we merge in each step the two clusters whose two closest members have the smallest distance.



(b)



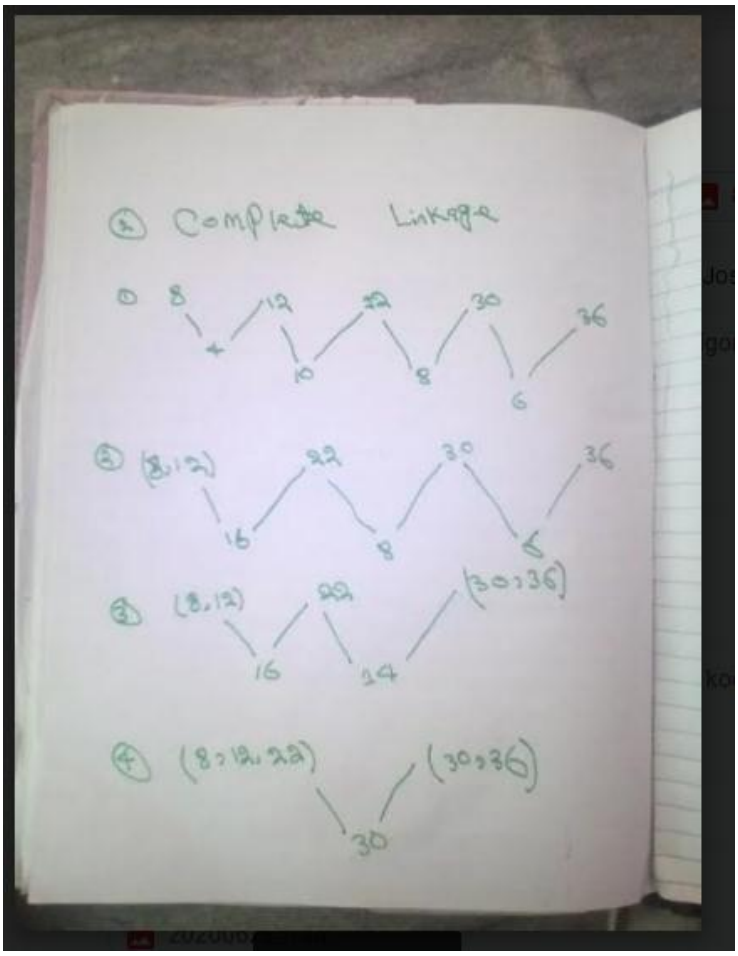
(Dendogram)

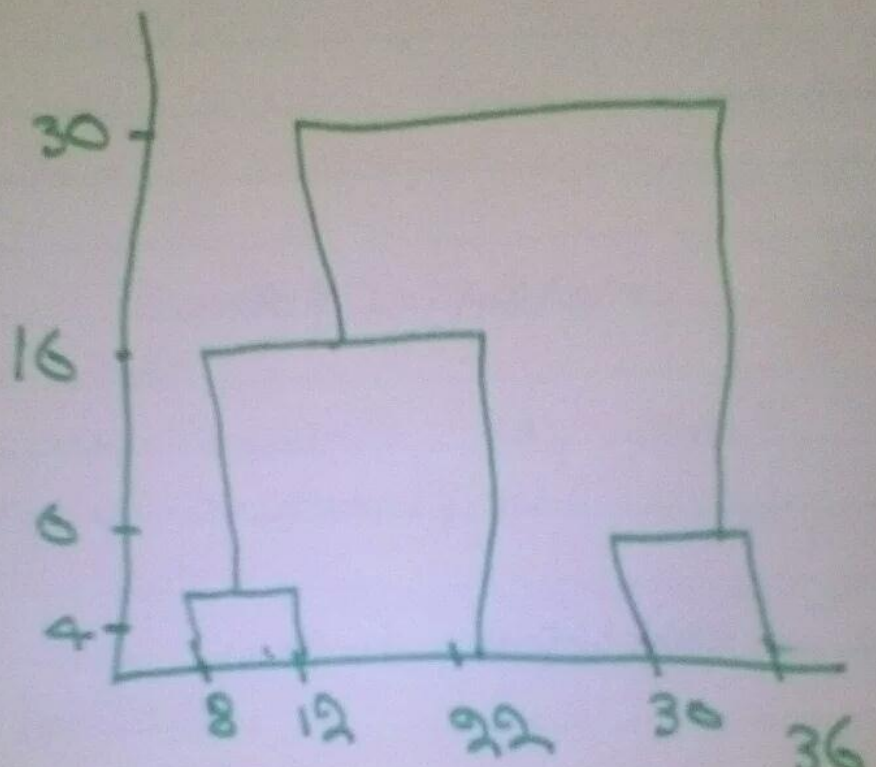
Using single linkage two clusters are formed :

Cluster 1 : (8,12)

Cluster 2 : (20,30,36)

2. **Complete Linkage:** In complete link hierarchical clustering, we merge in the members of the clusters in each step, which provide the smallest maximum pairwise distance.





(Dendogram)

----- THE END-----

Using complete linkage two clusters are formed :

Cluster 1 : (8,12,22)

Cluster 2 : (30,36)