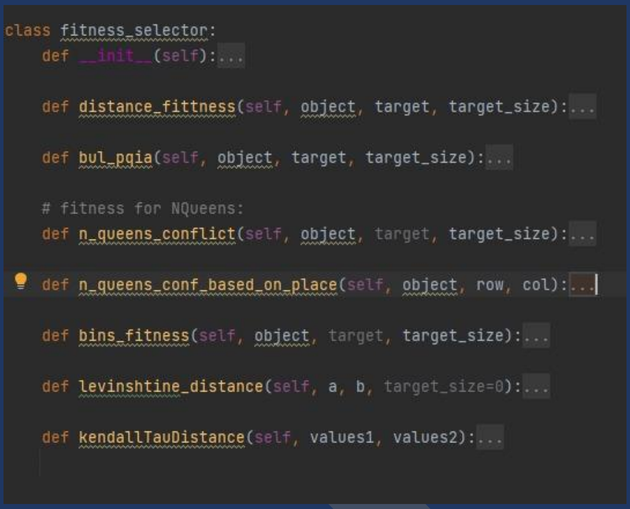
# **מעבדה 2 חלק א**

**סאמר חרעובה – 209050202**

**עבד אלרחמן אבו חוסין – 208517631**

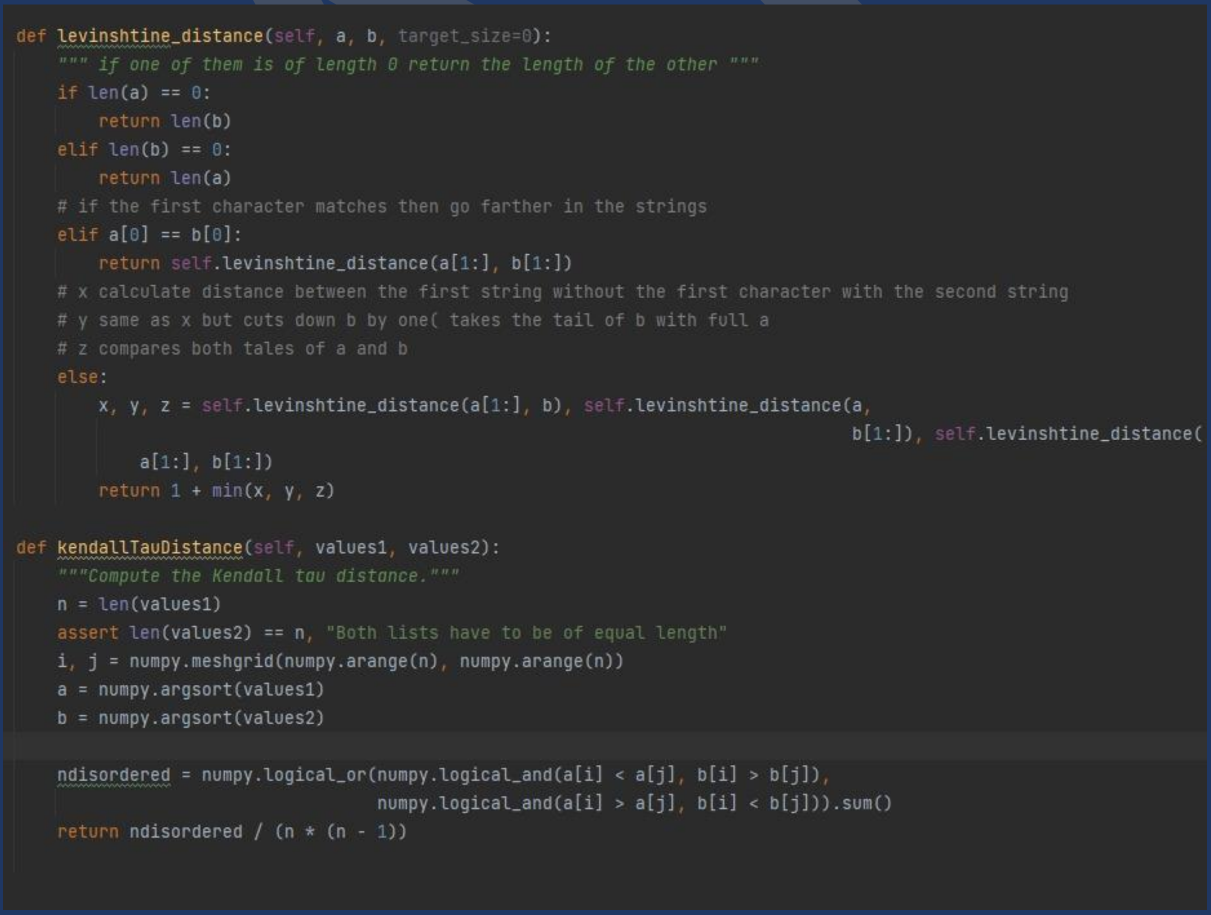
## סעיף 1:

implemented in fitness selector



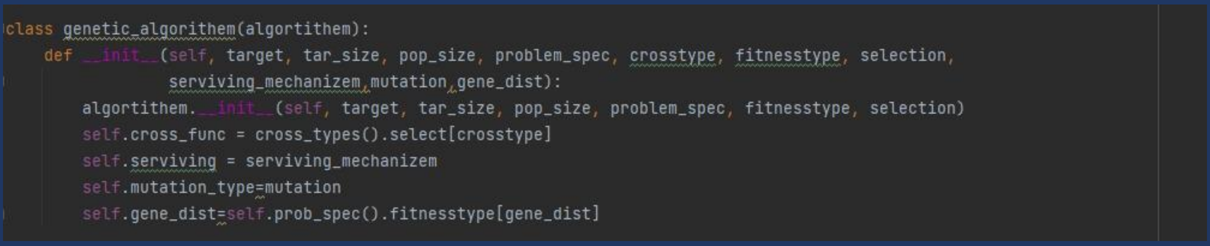
```
class fitness_selector:
    def __init__(self):...

    def distance_fittness(self, object, target, target_size):...

    def bul_pgia(self, object, target, target_size):...

    # fitness for NQueens:
    def n_queens_conflict(self, object, target, target_size):...

    def n_queens_conf_based_on_place(self, object, row, col):...

    def bins_fitness(self, object, target, target_size):...

    def levinshtine_distance(self, a, b, target_size=0):...

    def kendallTauDistance(self, values1, values2):...
```

Implementation



```
def levinshtine_distance(self, a, b, target_size=0):
    """ if one of them is of length 0 return the length of the other """
    if len(a) == 0:
        return len(b)
    elif len(b) == 0:
        return len(a)
    # if the first character matches then go farther in the strings
    elif a[0] == b[0]:
        return self.levinshtine_distance(a[1:], b[1:])
    # x calculate distance between the first string without the first character with the second string
    # y same as x but cuts down b by one( takes the tail of b with full a
    # z compares both tales of a and b
    else:
        x, y, z = self.levinshtine_distance(a[1:], b), self.levinshtine_distance(a,
                                                                                  b[1:]), self.levinshtine_distance(
            a[1:], b[1:])
        return 1 + min(x, y, z)

def kendallTauDistance(self, values1, values2):
    """Compute the Kendall tau distance."""
    n = len(values1)
    assert len(values2) == n, "Both lists have to be of equal length"
    i, j = numpy.meshgrid(numpy.arange(n), numpy.arange(n))
    a = numpy.argsort(values1)
    b = numpy.argsort(values2)

    ndisordered = numpy.logical_or(numpy.logical_and(a[i] < a[j], b[i] > b[j]),
                                   numpy.logical_and(a[i] > a[j], b[i] < b[j])).sum()
    return ndisordered / (n * (n - 1))
```

self.gene_dist is the distance function that we get from fitness type class , so that we can use it in the algorithm as we please



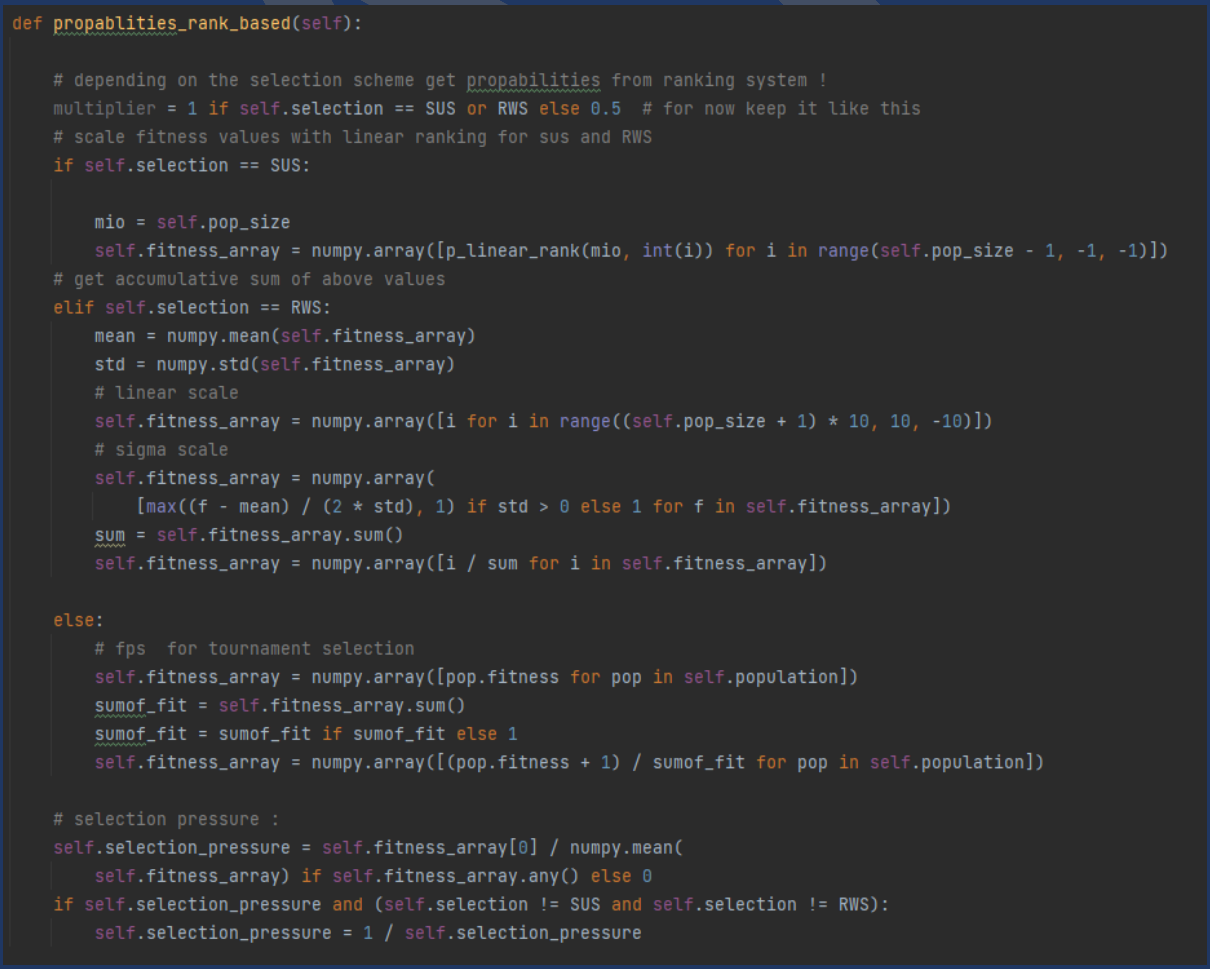
```python
class genetic_algorithem(algortithem):
    def __init__(self, target, tar_size, pop_size, problem_spec, crosstype, fitnesstype, selection,
                 serviving_mechanizem,mutation,gene_dist):
        algortithem.__init__(self, target, tar_size, pop_size, problem_spec, fitnesstype, selection)
        self.cross_func = cross_types().select[crosstype]
        self.serviving = serviving_mechanizem
        self.mutation_type=mutation
        self.gene_dist=self.prob_spec().fitnesstype[gene_dist]
```

## סעיף 2:

previously the ranking system was done in each selection function , now we changed it to be done on the population and just send a probabilities array to the selection function

selection pressure:

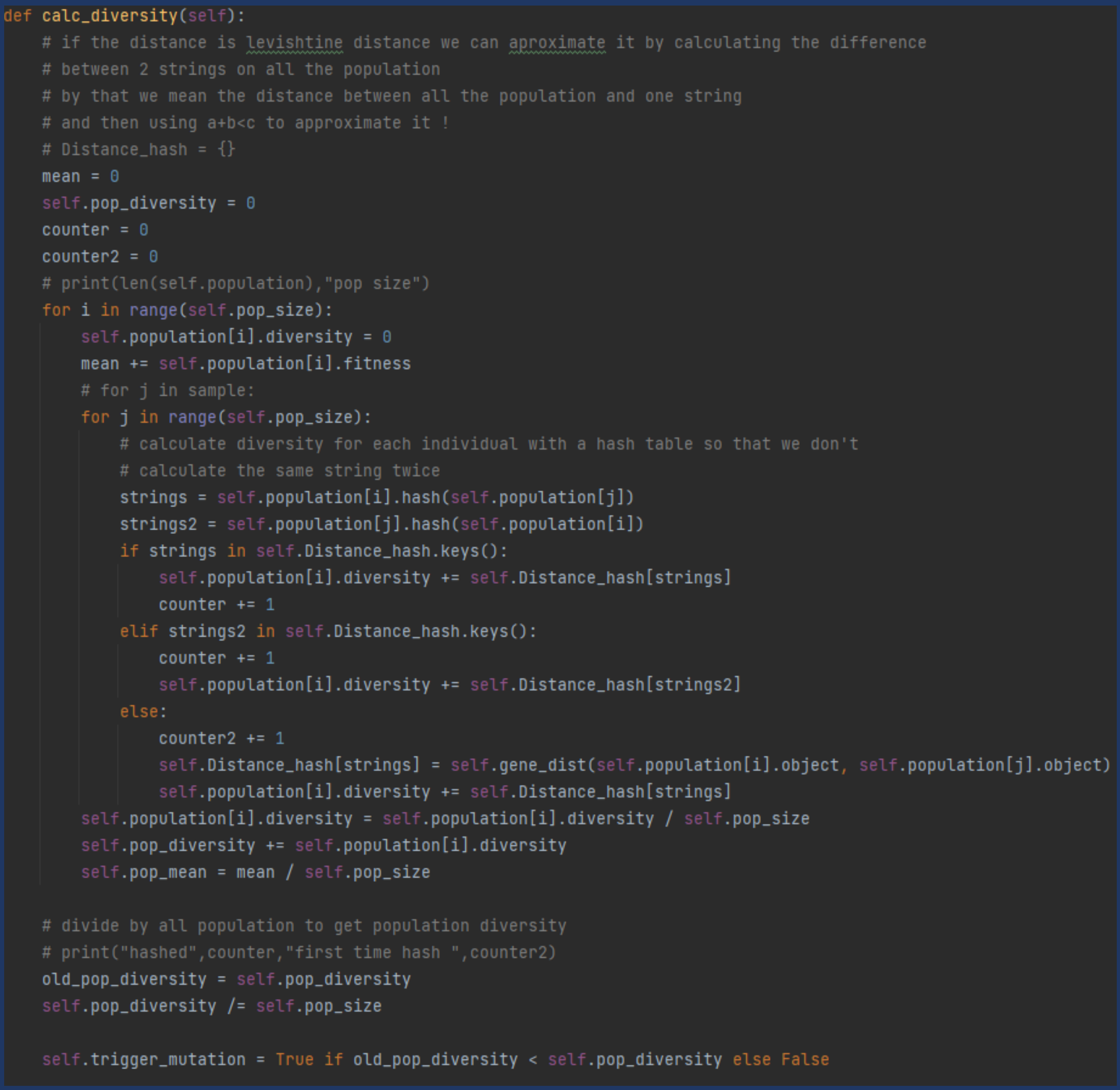
- in case of RWS and SUS and Tournment: we use the probability of selection that exists in fitness array(it's a probability array of each individual) to find the probability of selecting best individual and the average individual



```python
def propablities_rank_based(self):

    # depending on the selection scheme get propabilities from ranking system !
    multiplier = 1 if self.selection == SUS or RWS else 0.5  # for now keep it like this
    # scale fitness values with linear ranking for sus and RWS
    if self.selection == SUS:

        mio = self.pop_size
        self.fitness_array = numpy.array([p_linear_rank(mio, int(i)) for i in range(self.pop_size - 1, -1, -1)])
    # get accumulative sum of above values
    elif self.selection == RWS:
        mean = numpy.mean(self.fitness_array)
        std = numpy.std(self.fitness_array)
        # linear scale
        self.fitness_array = numpy.array([i for i in range((self.pop_size + 1) * 10, 10, -10)])
        # sigma scale
        self.fitness_array = numpy.array(
            [max((f - mean) / (2 * std), 1) if std > 0 else 1 for f in self.fitness_array])
        sum = self.fitness_array.sum()
        self.fitness_array = numpy.array([i / sum for i in self.fitness_array])

    else:
        # fps  for tournament selection
        self.fitness_array = numpy.array([pop.fitness for pop in self.population])
        sumof_fit = self.fitness_array.sum()
        sumof_fit = sumof_fit if sumof_fit else 1
        self.fitness_array = numpy.array([(pop.fitness + 1) / sumof_fit for pop in self.population])

    # selection pressure :
    self.selection_pressure = self.fitness_array[0] / numpy.mean(
        self.fitness_array) if self.fitness_array.any() else 0
    if self.selection_pressure and (self.selection != SUS and self.selection != RWS):
        self.selection_pressure = 1 / self.selection_pressure
```

## Diversity :

- to calculate diversity we calculate the distance of each individual with the full population , then average it out and store it
- then add all individual diversities and average them out to indicate the diversity of the whole population



```python
def calc_diversity(self):
    # if the distance is levishtine distance we can aproximate it by calculating the difference
    # between 2 strings on all the population
    # by that we mean the distance between all the population and one string
    # and then using a+b<c to approximate it !
    # Distance_hash = {}
    mean = 0
    self.pop_diversity = 0
    counter = 0
    counter2 = 0
    # print(len(self.population),"pop size")
    for i in range(self.pop_size):
        self.population[i].diversity = 0
        mean += self.population[i].fitness
        # for j in sample:
        for j in range(self.pop_size):
            # calculate diversity for each individual with a hash table so that we don't
            # calculate the same string twice
            strings = self.population[i].hash(self.population[j])
            strings2 = self.population[j].hash(self.population[i])
            if strings in self.Distance_hash.keys():
                self.population[i].diversity += self.Distance_hash[strings]
                counter += 1
            elif strings2 in self.Distance_hash.keys():
                counter += 1
                self.population[i].diversity += self.Distance_hash[strings2]
            else:
                counter2 += 1
                self.Distance_hash[strings] = self.gene_dist(self.population[i].object, self.population[j].object)
                self.population[i].diversity += self.Distance_hash[strings]
        self.population[i].diversity = self.population[i].diversity / self.pop_size
        self.pop_diversity += self.population[i].diversity
        self.pop_mean = mean / self.pop_size

    # divide by all population to get population diversity
    # print("hashed",counter,"first time hash ",counter2)
    old_pop_diversity = self.pop_diversity
    self.pop_diversity /= self.pop_size

    self.trigger_mutation = True if old_pop_diversity < self.pop_diversity else False
```

## סעיף 3:

### תוצאות :

| population size | exploration to exploitaion | Hyper triggered | mutation parameter=0.25 |
|---|---|---|---|
| 100 | generation:33 ,runtime: 31 | generations:28 ,runtime:32 | generation:35 ,runtime: 35.7 |
| 300 | generation:21 ,runtime: 209 | generations:19 ,runtime:182 | generation:20 ,runtime: 195 |

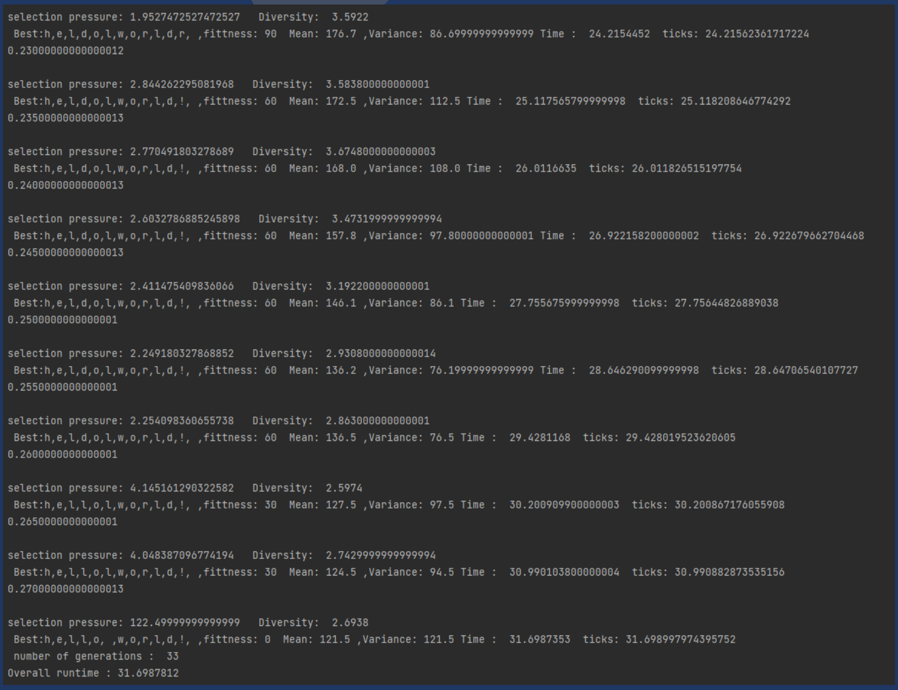
**As we have seen in previous lab ,our perfect mutation rate was 0.25 so we used it , as we have tried lower or higher , and it wasn't optimal**

**Results :**

- **when the population is small both Hyper triggered and linear exploration to exploitation both work better than a fixed mutation rate.**
- **When the population is bigger:**
  1. **we can see that the linear exploration to exploitation is worst than both , but this result does not reflect the full truth , as in bigger populations our linear function should be adjusted to the appropriate ratio between population size and the linear value that we add to the original mating probability**
  2. **Hyper triggered is faster than a fixed mutation rate. And gets out of local optima much faster**

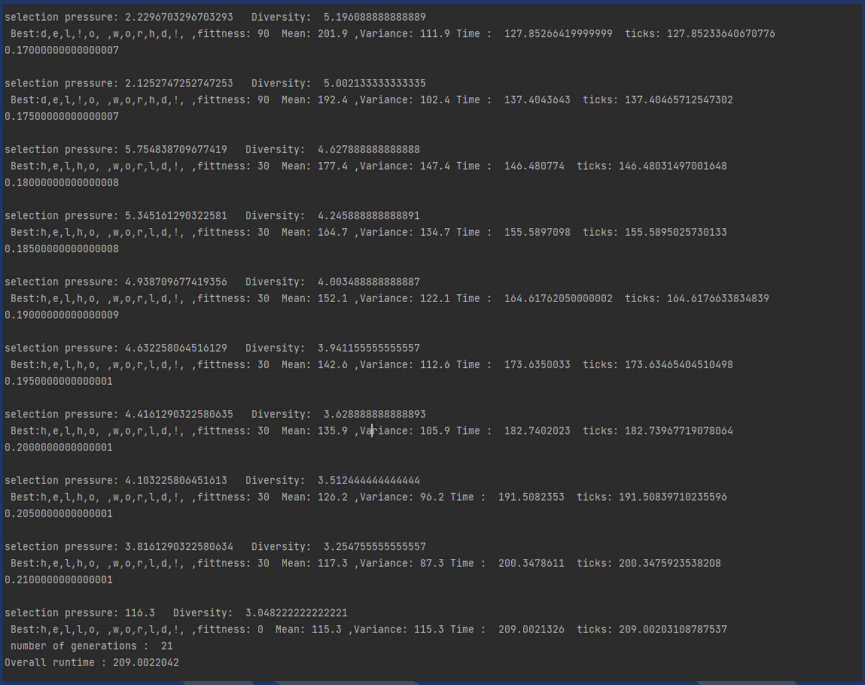
## מעבר הדרגתי מ exploration ל exploitation :

Population size=100



```
selection pressure: 1.9527472527472527   Diversity:  3.5922
 Best:h,e,l,d,o,l,w,o,r,l,d,r, ,fittness: 90  Mean: 176.7 ,Variance: 86.69999999999999 Time :  24.2154452  ticks: 24.21562361717224
0.23000000000000012

selection pressure: 2.844262295081968   Diversity:  3.5838000000000001
 Best:h,e,l,d,o,l,w,o,r,l,d,!, ,fittness: 60  Mean: 172.5 ,Variance: 112.5 Time :  25.117565799999998  ticks: 25.118208646774292
0.23500000000000013

selection pressure: 2.770491803278689   Diversity:  3.6748000000000003
 Best:h,e,l,d,o,l,w,o,r,l,d,!, ,fittness: 60  Mean: 168.0 ,Variance: 108.0 Time :  26.0116635  ticks: 26.011826515197754
0.24000000000000013

selection pressure: 2.6032786885245898   Diversity:  3.4731999999999994
 Best:h,e,l,d,o,l,w,o,r,l,d,!, ,fittness: 60  Mean: 157.8 ,Variance: 97.80000000000001 Time :  26.922158200000002  ticks: 26.922679662704468
0.24500000000000013

selection pressure: 2.411475409836066   Diversity:  3.1922000000000001
 Best:h,e,l,d,o,l,w,o,r,l,d,!, ,fittness: 60  Mean: 146.1 ,Variance: 86.1 Time :  27.755675999999998  ticks: 27.75644826889038
0.2500000000000001

selection pressure: 2.249180327868852   Diversity:  2.9308000000000014
 Best:h,e,l,d,o,l,w,o,r,l,d,!, ,fittness: 60  Mean: 136.2 ,Variance: 76.19999999999999 Time :  28.646290099999998  ticks: 28.64706540107727
0.2550000000000001

selection pressure: 2.254098360655738   Diversity:  2.8630000000000001
 Best:h,e,l,d,o,l,w,o,r,l,d,!, ,fittness: 60  Mean: 136.5 ,Variance: 76.5 Time :  29.4281168  ticks: 29.428019523620605
0.2600000000000001

selection pressure: 4.145161290322582   Diversity:  2.5974
 Best:h,e,l,l,o,l,w,o,r,l,d,!, ,fittness: 30  Mean: 127.5 ,Variance: 97.5 Time :  30.200909900000003  ticks: 30.200867176055908
0.2650000000000001

selection pressure: 4.048387096774194   Diversity:  2.7429999999999994
 Best:h,e,l,l,o,l,w,o,r,l,d,!, ,fittness: 30  Mean: 124.5 ,Variance: 94.5 Time :  30.990103800000004  ticks: 30.990882873535156
0.27000000000000013

selection pressure: 122.49999999999999   Diversity:  2.6938
 Best:h,e,l,l,o, ,w,o,r,l,d,!, ,fittness: 0  Mean: 121.5 ,Variance: 121.5 Time :  31.6987353  ticks: 31.698997974395752
 number of generations :  33
Overall runtime : 31.6987812
```

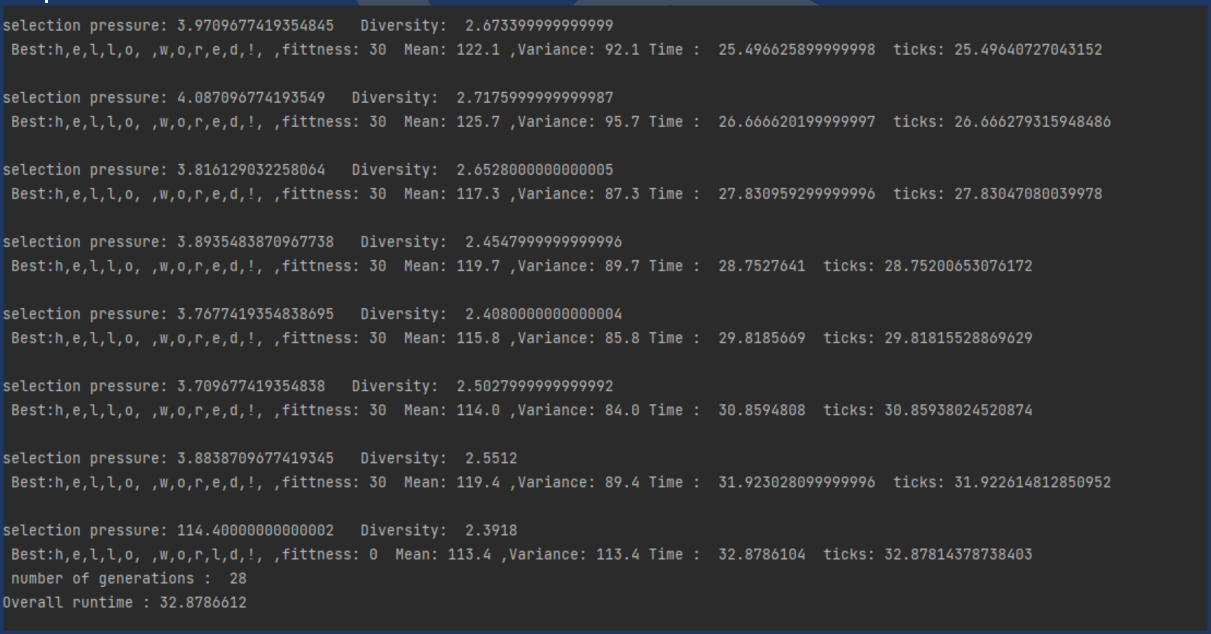
## Population size=300



```
selection pressure: 2.2296703296703293   Diversity:  5.196088888888889
 Best:d,e,l,!,o, ,w,o,r,h,d,!, ,fittness: 90  Mean: 201.9 ,Variance: 111.9 Time :  127.85266419999999  ticks: 127.85233640670776
0.17000000000000007

selection pressure: 2.1252747252747253   Diversity:  5.002133333333335
 Best:d,e,l,!,o, ,w,o,r,h,d,!, ,fittness: 90  Mean: 192.4 ,Variance: 102.4 Time :  137.4043643  ticks: 137.40465712547302
0.17500000000000007

selection pressure: 5.754838709677419   Diversity:  4.627888888888888
 Best:h,e,l,h,o, ,w,o,r,l,d,!, ,fittness: 30  Mean: 177.4 ,Variance: 147.4 Time :  146.480774  ticks: 146.48031497001648
0.18000000000000008

selection pressure: 5.345161290322581   Diversity:  4.245888888888891
 Best:h,e,l,h,o, ,w,o,r,l,d,!, ,fittness: 30  Mean: 164.7 ,Variance: 134.7 Time :  155.5897098  ticks: 155.5895025730133
0.18500000000000008

selection pressure: 4.938709677419356   Diversity:  4.003488888888887
 Best:h,e,l,h,o, ,w,o,r,l,d,!, ,fittness: 30  Mean: 152.1 ,Variance: 122.1 Time :  164.61762050000002  ticks: 164.6176633834839
0.19000000000000009

selection pressure: 4.632258064516129   Diversity:  3.941155555555557
 Best:h,e,l,h,o, ,w,o,r,l,d,!, ,fittness: 30  Mean: 142.6 ,Variance: 112.6 Time :  173.6350033  ticks: 173.63465404510498
0.1950000000000001

selection pressure: 4.4161290322580635   Diversity:  3.628888888888893
 Best:h,e,l,h,o, ,w,o,r,l,d,!, ,fittness: 30  Mean: 135.9 ,Variance: 105.9 Time :  182.7402023  ticks: 182.73967719078064
0.2000000000000001

selection pressure: 4.103225806451613   Diversity:  3.512444444444444
 Best:h,e,l,h,o, ,w,o,r,l,d,!, ,fittness: 30  Mean: 126.2 ,Variance: 96.2 Time :  191.5082353  ticks: 191.50839710235596
0.2050000000000001

selection pressure: 3.8161290322580634   Diversity:  3.254755555555557
 Best:h,e,l,h,o, ,w,o,r,l,d,!, ,fittness: 30  Mean: 117.3 ,Variance: 87.3 Time :  200.3478611  ticks: 200.3475923538208
0.2100000000000001

selection pressure: 116.3   Diversity:  3.048222222222221
 Best:h,e,l,l,o, ,w,o,r,l,d,!, ,fittness: 0  Mean: 115.3 ,Variance: 115.3 Time :  209.0021326  ticks: 209.00203108787537
 number of generations :  21
Overall runtime : 209.0022042
```

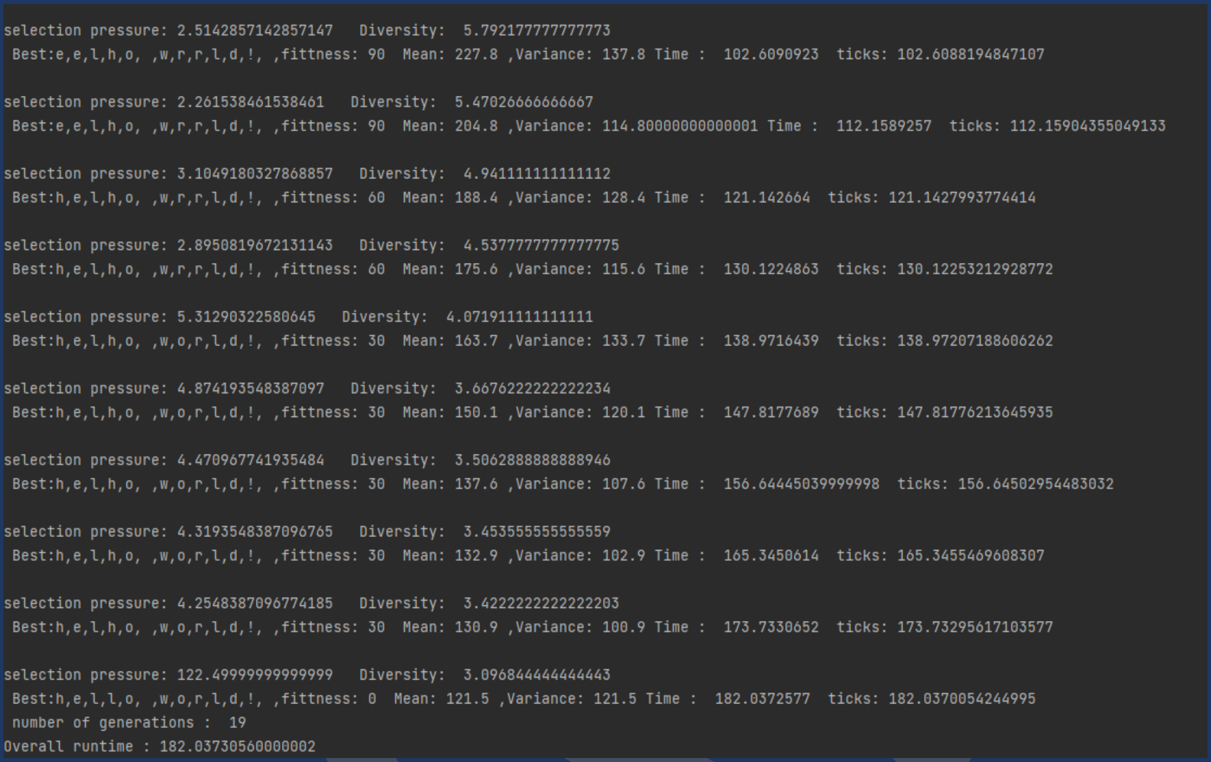
# Hyper triggered:

## Population size=100:



```
selection pressure: 3.9709677419354845   Diversity:  2.673399999999999
 Best:h,e,l,l,o, ,w,o,r,e,d,!, ,fittness: 30  Mean: 122.1 ,Variance: 92.1 Time :  25.496625899999998  ticks: 25.49640727043152

selection pressure: 4.087096774193549   Diversity:  2.7175999999999987
 Best:h,e,l,l,o, ,w,o,r,e,d,!, ,fittness: 30  Mean: 125.7 ,Variance: 95.7 Time :  26.666620199999997  ticks: 26.666279315948486

selection pressure: 3.816129032258064   Diversity:  2.6528000000000005
 Best:h,e,l,l,o, ,w,o,r,e,d,!, ,fittness: 30  Mean: 117.3 ,Variance: 87.3 Time :  27.830959299999996  ticks: 27.83047080039978

selection pressure: 3.8935483870967738   Diversity:  2.4547999999999996
 Best:h,e,l,l,o, ,w,o,r,e,d,!, ,fittness: 30  Mean: 119.7 ,Variance: 89.7 Time :  28.7527641  ticks: 28.75200653076172

selection pressure: 3.7677419354838695   Diversity:  2.4080000000000004
 Best:h,e,l,l,o, ,w,o,r,e,d,!, ,fittness: 30  Mean: 115.8 ,Variance: 85.8 Time :  29.8185669  ticks: 29.81815528869629

selection pressure: 3.709677419354838   Diversity:  2.5027999999999992
 Best:h,e,l,l,o, ,w,o,r,e,d,!, ,fittness: 30  Mean: 114.0 ,Variance: 84.0 Time :  30.8594808  ticks: 30.85938024520874

selection pressure: 3.8838709677419345   Diversity:  2.5512
 Best:h,e,l,l,o, ,w,o,r,e,d,!, ,fittness: 30  Mean: 119.4 ,Variance: 89.4 Time :  31.923028099999996  ticks: 31.922614812850952

selection pressure: 114.40000000000002   Diversity:  2.3918
 Best:h,e,l,l,o, ,w,o,r,l,d,!, ,fittness: 0  Mean: 113.4 ,Variance: 113.4 Time :  32.8786104  ticks: 32.87814378738403
 number of generations :  28
Overall runtime : 32.8786612
```

## Population size=300

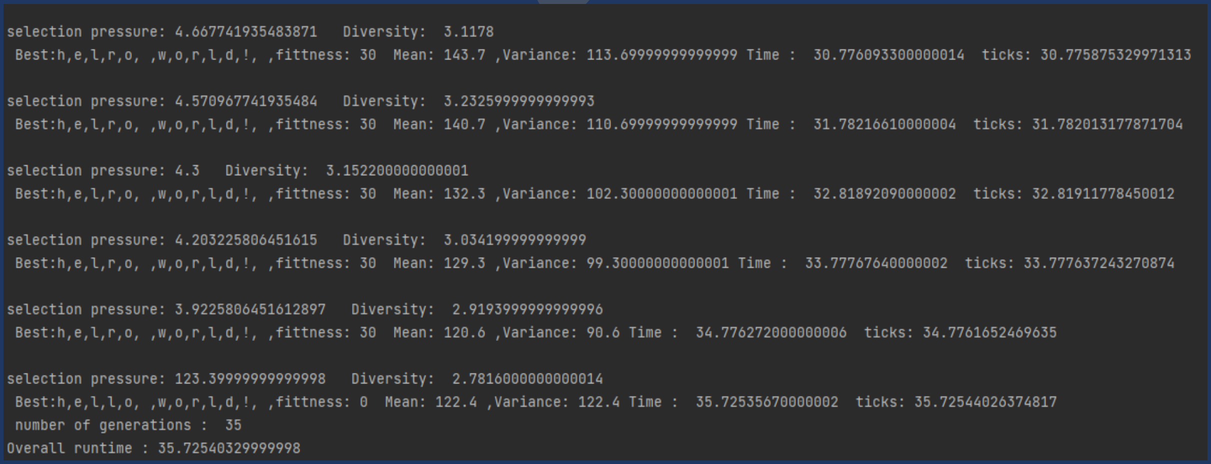


```
selection pressure: 2.5142857142857147   Diversity:  5.792177777777773
 Best:e,e,l,h,o, ,w,r,r,l,d,!, ,fittness: 90  Mean: 227.8 ,Variance: 137.8 Time :  102.6090923  ticks: 102.6088194847107

selection pressure: 2.261538461538461   Diversity:  5.47026666666667
 Best:e,e,l,h,o, ,w,r,r,l,d,!, ,fittness: 90  Mean: 204.8 ,Variance: 114.80000000000001 Time :  112.1589257  ticks: 112.15904355049133

selection pressure: 3.1049180327868857   Diversity:  4.941111111111112
 Best:h,e,l,h,o, ,w,r,r,l,d,!, ,fittness: 60  Mean: 188.4 ,Variance: 128.4 Time :  121.142664  ticks: 121.1427993774414

selection pressure: 2.8950819672131143   Diversity:  4.5377777777777775
 Best:h,e,l,h,o, ,w,r,r,l,d,!, ,fittness: 60  Mean: 175.6 ,Variance: 115.6 Time :  130.1224863  ticks: 130.12253212928772

selection pressure: 5.31290322580645   Diversity:  4.071911111111111
 Best:h,e,l,h,o, ,w,o,r,l,d,!, ,fittness: 30  Mean: 163.7 ,Variance: 133.7 Time :  138.9716439  ticks: 138.97207188606262

selection pressure: 4.874193548387097   Diversity:  3.6676222222222234
 Best:h,e,l,h,o, ,w,o,r,l,d,!, ,fittness: 30  Mean: 150.1 ,Variance: 120.1 Time :  147.8177689  ticks: 147.81776213645935

selection pressure: 4.470967741935484   Diversity:  3.5062888888888946
 Best:h,e,l,h,o, ,w,o,r,l,d,!, ,fittness: 30  Mean: 137.6 ,Variance: 107.6 Time :  156.64445039999998  ticks: 156.64502954483032

selection pressure: 4.3193548387096765   Diversity:  3.453555555555559
 Best:h,e,l,h,o, ,w,o,r,l,d,!, ,fittness: 30  Mean: 132.9 ,Variance: 102.9 Time :  165.3450614  ticks: 165.3455469608307

selection pressure: 4.2548387096774185   Diversity:  3.4222222222222203
 Best:h,e,l,h,o, ,w,o,r,l,d,!, ,fittness: 30  Mean: 130.9 ,Variance: 100.9 Time :  173.7330652  ticks: 173.73295617103577

selection pressure: 122.49999999999999   Diversity:  3.096844444444443
 Best:h,e,l,l,o, ,w,o,r,l,d,!, ,fittness: 0  Mean: 121.5 ,Variance: 121.5 Time :  182.0372577  ticks: 182.0370054244995
 number of generations :  19
Overall runtime : 182.03730560000002
```

## Fixed mutation parameter:

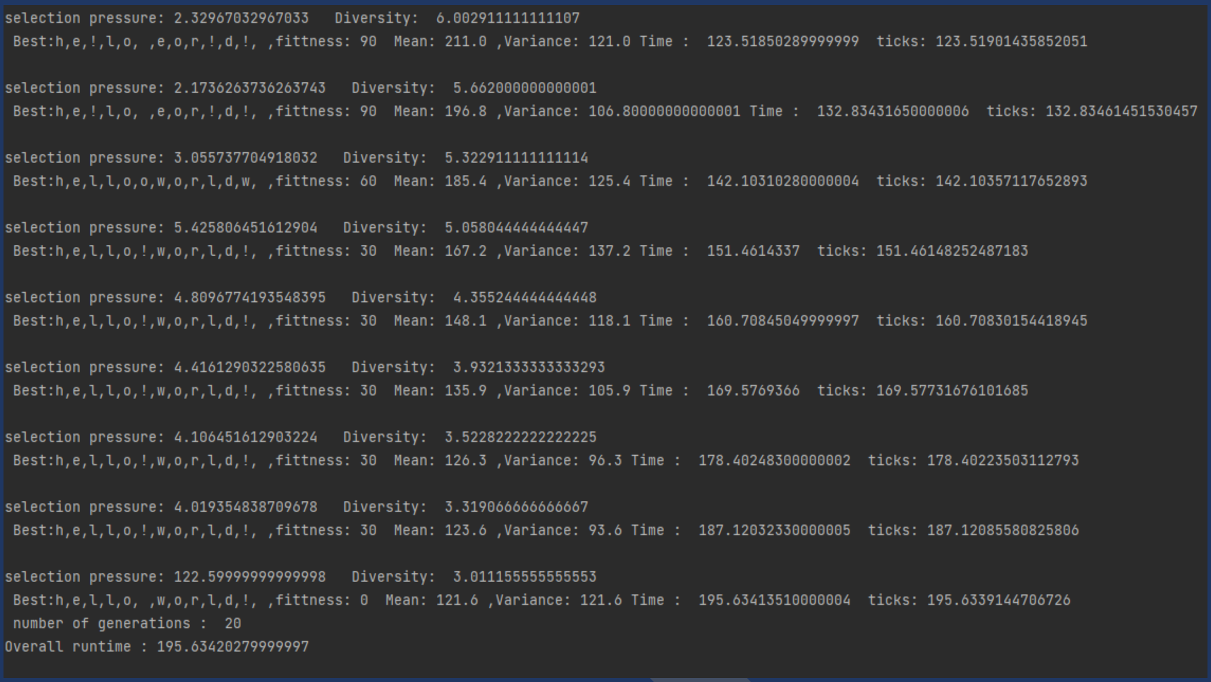
### Mutation rate=0.25:

### Population size=100:



```
selection pressure: 4.667741935483871   Diversity:  3.1178
 Best:h,e,l,r,o, ,w,o,r,l,d,!, ,fittness: 30  Mean: 143.7 ,Variance: 113.69999999999999 Time :  30.776093300000014  ticks: 30.775875329971313

selection pressure: 4.570967741935484   Diversity:  3.2325999999999993
 Best:h,e,l,r,o, ,w,o,r,l,d,!, ,fittness: 30  Mean: 140.7 ,Variance: 110.69999999999999 Time :  31.78216610000004  ticks: 31.782013177871704

selection pressure: 4.3   Diversity:  3.152200000000001
 Best:h,e,l,r,o, ,w,o,r,l,d,!, ,fittness: 30  Mean: 132.3 ,Variance: 102.30000000000001 Time :  32.81892090000002  ticks: 32.81911778450012

selection pressure: 4.203225806451615   Diversity:  3.034199999999999
 Best:h,e,l,r,o, ,w,o,r,l,d,!, ,fittness: 30  Mean: 129.3 ,Variance: 99.30000000000001 Time :  33.77767640000002  ticks: 33.777637243270874

selection pressure: 3.9225806451612897   Diversity:  2.9193999999999996
 Best:h,e,l,r,o, ,w,o,r,l,d,!, ,fittness: 30  Mean: 120.6 ,Variance: 90.6 Time :  34.776272000000006  ticks: 34.7761652469635

selection pressure: 123.39999999999998   Diversity:  2.7816000000000014
 Best:h,e,l,l,o, ,w,o,r,l,d,!, ,fittness: 0  Mean: 122.4 ,Variance: 122.4 Time :  35.72535670000002  ticks: 35.72544026374817
 number of generations :  35
Overall runtime : 35.72540329999998
```

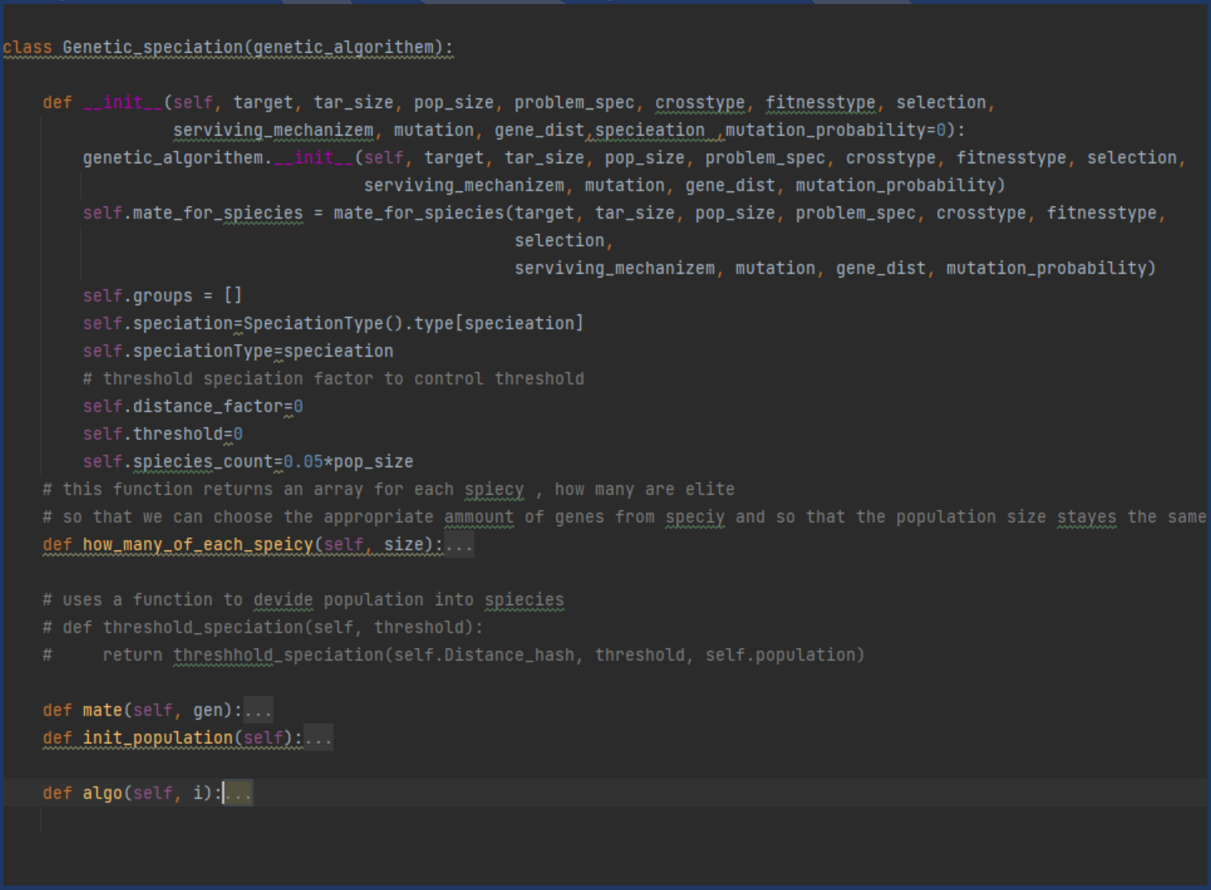
## Population size=300:



```
selection pressure: 2.32967032967033   Diversity:  6.002911111111107
 Best:h,e,!,l,o, ,e,o,r,!,d,!, ,fittness: 90  Mean: 211.0 ,Variance: 121.0 Time :  123.51850289999999  ticks: 123.51901435852051

selection pressure: 2.1736263736263743   Diversity:  5.662000000000001
 Best:h,e,!,l,o, ,e,o,r,!,d,!, ,fittness: 90  Mean: 196.8 ,Variance: 106.80000000000001 Time :  132.83431650000006  ticks: 132.83461451530457

selection pressure: 3.055737704918032   Diversity:  5.322911111111114
 Best:h,e,l,l,o,o,w,o,r,l,d,w, ,fittness: 60  Mean: 185.4 ,Variance: 125.4 Time :  142.10310280000004  ticks: 142.10357117652893

selection pressure: 5.425806451612904   Diversity:  5.058044444444447
 Best:h,e,l,l,o,!,w,o,r,l,d,!, ,fittness: 30  Mean: 167.2 ,Variance: 137.2 Time :  151.4614337  ticks: 151.46148252487183

selection pressure: 4.8096774193548395   Diversity:  4.355244444444448
 Best:h,e,l,l,o,!,w,o,r,l,d,!, ,fittness: 30  Mean: 148.1 ,Variance: 118.1 Time :  160.70845049999997  ticks: 160.70830154418945

selection pressure: 4.4161290322580635   Diversity:  3.9321333333333293
 Best:h,e,l,l,o,!,w,o,r,l,d,!, ,fittness: 30  Mean: 135.9 ,Variance: 105.9 Time :  169.5769366  ticks: 169.57731676101685

selection pressure: 4.106451612903224   Diversity:  3.5228222222222225
 Best:h,e,l,l,o,!,w,o,r,l,d,!, ,fittness: 30  Mean: 126.3 ,Variance: 96.3 Time :  178.40248300000002  ticks: 178.40223503112793

selection pressure: 4.019354838709678   Diversity:  3.319066666666667
 Best:h,e,l,l,o,!,w,o,r,l,d,!, ,fittness: 30  Mean: 123.6 ,Variance: 93.6 Time :  187.12032330000005  ticks: 187.12085580825806

selection pressure: 122.59999999999998   Diversity:  3.011155555555553
 Best:h,e,l,l,o, ,w,o,r,l,d,!, ,fittness: 0  Mean: 121.6 ,Variance: 121.6 Time :  195.63413510000004  ticks: 195.6339144706726
 number of generations :  20
Overall runtime : 195.63420279999997
```

## **סעיף 4:**

## Implementation of GA with speciation:



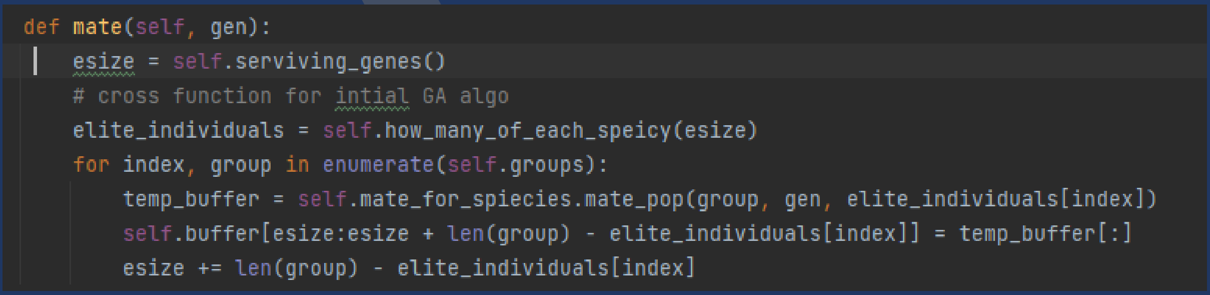
```python
class Genetic_speciation(genetic_algorithem):

    def __init__(self, target, tar_size, pop_size, problem_spec, crosstype, fitnesstype, selection,
                 serviving_mechanizem, mutation, gene_dist,specieation ,mutation_probability=0):
        genetic_algorithem.__init__(self, target, tar_size, pop_size, problem_spec, crosstype, fitnesstype, selection,
                                    serviving_mechanizem, mutation, gene_dist, mutation_probability)
        self.mate_for_spiecies = mate_for_spiecies(target, tar_size, pop_size, problem_spec, crosstype, fitnesstype,
                                                   selection,
                                                   serviving_mechanizem, mutation, gene_dist, mutation_probability)
        self.groups = []
        self.speciation=SpeciationType().type[specieation]
        self.speciationType=specieation
        # threshold speciation factor to control threshold
        self.distance_factor=0
        self.threshold=0
        self.spiecies_count=0.05*pop_size
    # this function returns an array for each spiecy , how many are elite
    # so that we can choose the appropriate ammount of genes from speciy and so that the population size stayes the same
    def how_many_of_each_speicy(self, size):...

    # uses a function to devide population into spiecies
    # def threshold_speciation(self, threshold):
    #     return threshhold_speciation(self.Distance_hash, threshold, self.population)

    def mate(self, gen):...
    def init_population(self):...

    def algo(self, i):...
```

## Explanation of each parameter :

1. Groups is an array that has k clusters where k is defined from either threshold speciation or k-means clustering .
2. distance_factor : is a minimal distance metric between two identical genes that one of them was mutated.
3. Speciation : a function that returns clusters in an array .
4. SpeciationType : dictates which algorithm to use for speciation ,i.e. threshold speciation or k-means clustering
5. Self.threshold: is used either as k or as the current threshold based on the speciationType

## Explanation of each function :

1. `how_many_of_each_speicy(self,size)`
   - specifies the number and positions of the elite member in each cluster/specie
2. `def mate(self, gen):`
   - treats each specie as a singl population and mates only the ones that are in the cluster/specie
   - after going over all cluster , this function puts all genes of all kind of species in the buffer as separate genes.(not as clusters)

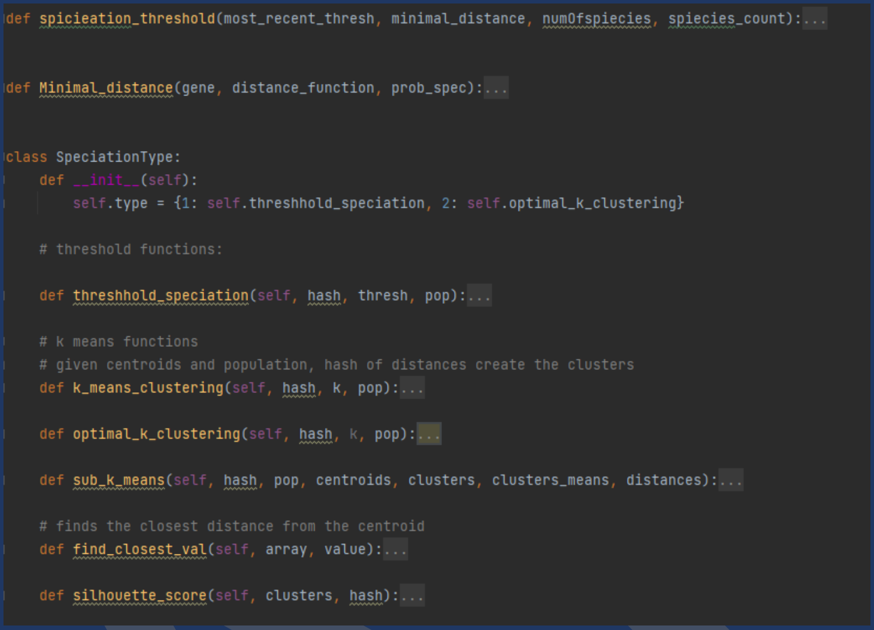


```
def mate(self, gen):
    esize = self.serviving_genes()
    # cross function for intial GA algo
    elite_individuals = self.how_many_of_each_speicy(esize)
    for index, group in enumerate(self.groups):
        temp_buffer = self.mate_for_spiecies.mate_pop(group, gen, elite_individuals[index])
        self.buffer[esize:esize + len(group) - elite_individuals[index]] = temp_buffer[:]
        esize += len(group) - elite_individuals[index]
```

3. `def init_population(self)`
   - this function initiates the population as it was done in previous implementations and calculates the distance_factor or k so that we can use those parameters in the speciation itself
4. `def algo(self, i):`
   - This is the algorithm itself:
     - First calculate diversity for all population
     - Sort them by fitness
     - Change current threshold based on previous iteration
     - Rank population for selection methods
     - Use the new mate function to mate the population

    - Then new hash is created for distance measurements

## Implementation of speciation types:



```python
def spicieation_threshold(most_recent_thresh, minimal_distance, numOfspiecies, spiecies_count):...


def Minimal_distance(gene, distance_function, prob_spec):...


class SpeciationType:
    def __init__(self):
        self.type = {1: self.threshhold_speciation, 2: self.optimal_k_clustering}

    # threshold functions:

    def threshhold_speciation(self, hash, thresh, pop):...

    # k means functions
    # given centroids and population, hash of distances create the clusters
    def k_means_clustering(self, hash, k, pop):...

    def optimal_k_clustering(self, hash, k, pop):...

    def sub_k_means(self, hash, pop, centroids, clusters, clusters_means, distances):...

    # finds the closest distance from the centroid
    def find_closest_val(self, array, value):...

    def silhouette_score(self, clusters, hash):...
```

```
1. def spicieation_threshold
```

- returns the new threshold

```
2. def Minimal_distance(gene, distance_function, prob_spec)
```

- returns the minimal distance to reduce or add to threshold

```
3. def threshhold_speciation(self, hash, thresh, pop):
```

- retuns the new clusters
- basically starts with the first gene checks if the threshold condition applies then adds member to a cluster
- after insertion of first 2 genes in a specific cluster, the distance check is used on all members of the specific cluster
- continues to add clusters and members until it finishes going over the full population

```
4. def optimal_k_clustering(self, hash, k, pop):
```

- starts with k=3 up to maximum k
- uses k-means clustering until silhouette_score retuns True , i.e. until k is optimal

```
5.  def k_means_clustering(self, hash, k, pop):
```

- Creates iterations of sub_k_means until population converges ,i.e. until centroids do not change from previous iteration

```
6.  def sub_k_means(self, hash, pop, centroids, clusters,
clusters_means, distances):
```

- Appoints genes to centroids based on minimal distance from centroid
- Returns new clusters with mean of distances in each cluster

Comparison between Threshold Speciation and  k-means Clustering Speciation :

Runs on strings matching (bul pgiaa):

| | Threshold | | | K-means clustering |
|---|---|---|---|---|
| population size | 5 % of poulation size | 15% of population | 30 | |
| 100 | generation:30 runtime: 34.5 | generations:34 runtime:30 | generation:90 runtime: 90 | generation:28 runtime: 25 |
| 200 | generation:30 runtime: 118 | generations:29 runtime:129 | generation:33 runtime: 143.5 | generation:27 runtime: 111 |
| 512 | generation:17 runtime: 502 | generations:23 runtime:637 | generation:17 runtime: 492 | generation:11 runtime: 340 |

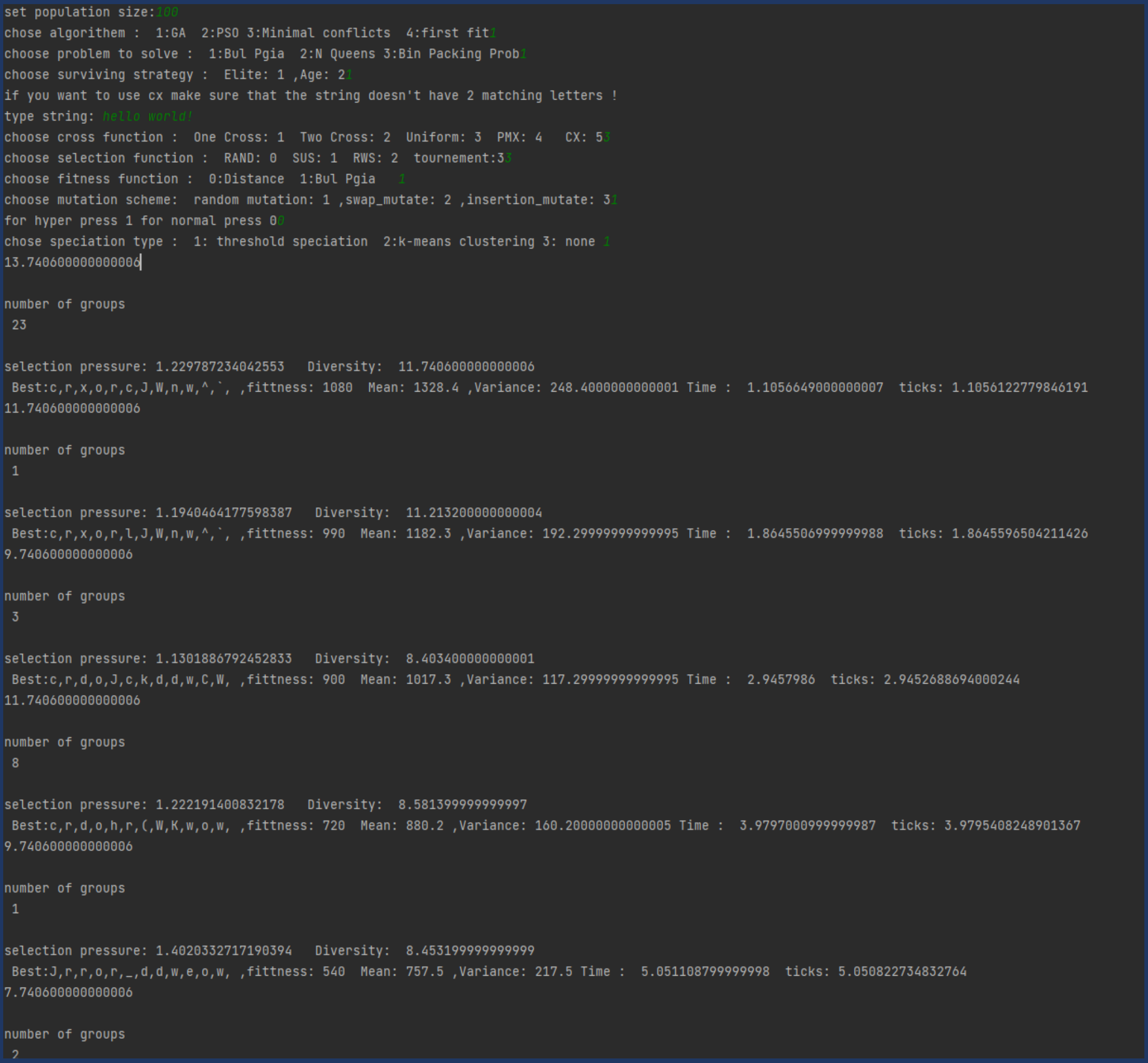
There are 2 main metrics to consider when comparing the two algorithms

1. Time to get to the solution
2. The quality of the solution

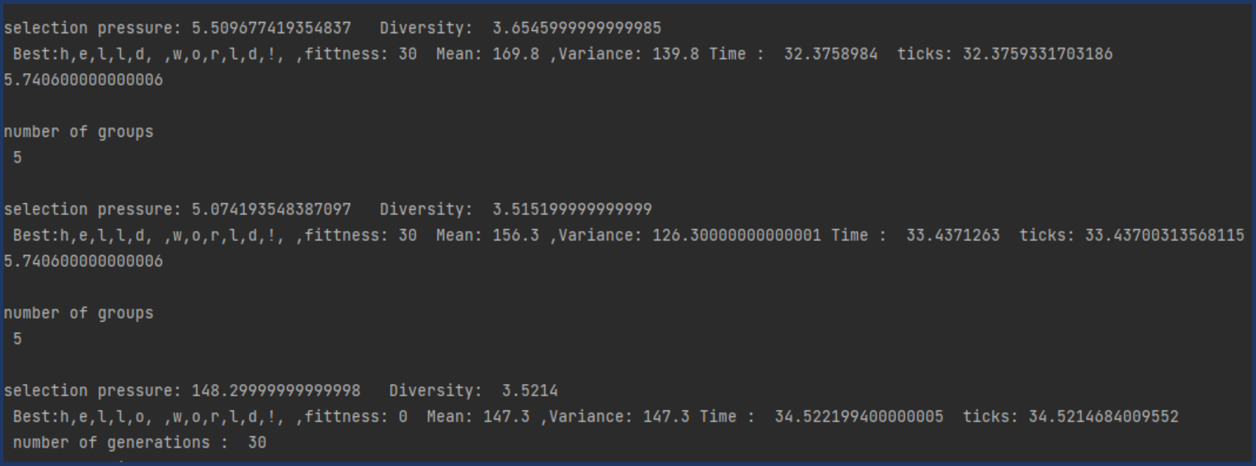
The results above are quite clear:

1. The time to get to the solution in k means is a lot faster than threshold speciation ,although in small populations it doesn't amount to much difference but in larger populations we can see a significant time reduction /
2. The quality of the solution on both algorithms is outstanding ,as we see in the mock trials below of the code running :
   - On both we can see that the initial mating of the species produces better solutions, we can see that in the fitness value and mean of the fitness value in both algorithms
   - But we can see that k means clustering handles local minima much better than threshold speciation  as it takes 3 to 4 iterations on average for threshold speciation to get out of local minima, yet K-means hardly stumbles on local minima

Threshold=5 , population size=100

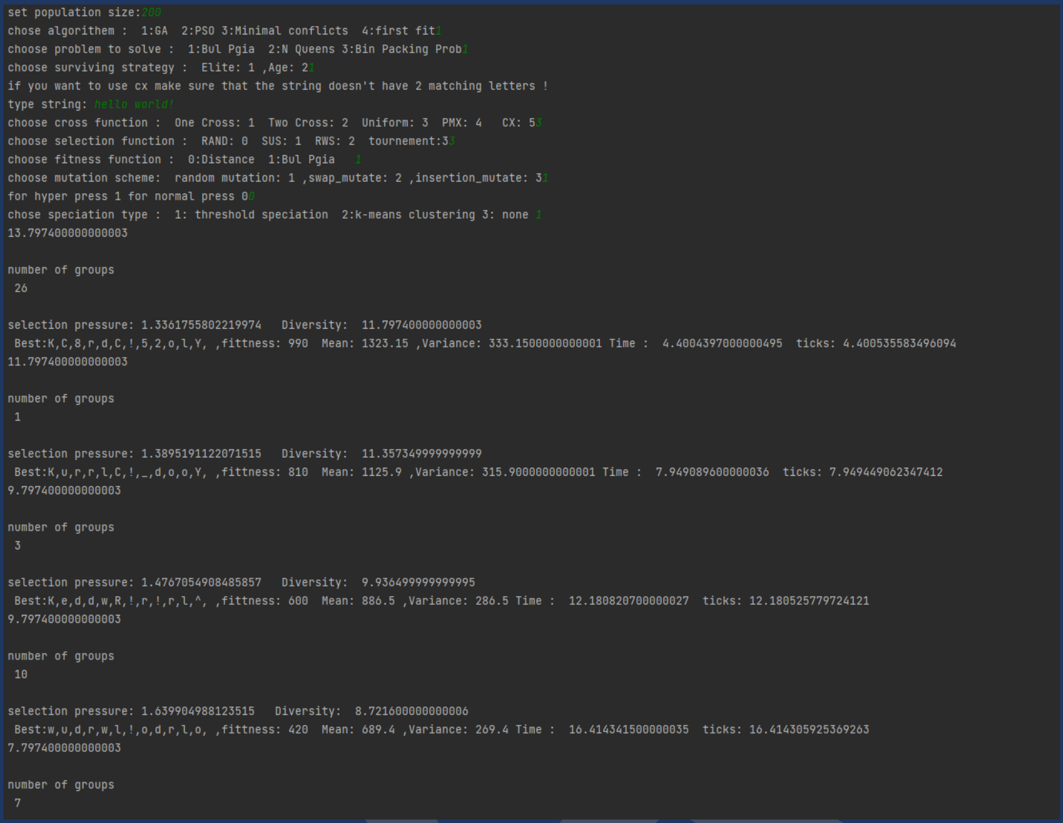


```
set population size:100
chose algorithem :  1:GA  2:PSO 3:Minimal conflicts  4:first fit1
choose problem to solve :  1:Bul Pgia  2:N Queens 3:Bin Packing Prob1
choose surviving strategy :  Elite: 1 ,Age: 21
if you want to use cx make sure that the string doesn't have 2 matching letters !
type string: hello world!
choose cross function :  One Cross: 1  Two Cross: 2  Uniform: 3  PMX: 4   CX: 53
choose selection function :  RAND: 0  SUS: 1  RWS: 2  tournement:33
choose fitness function :  0:Distance  1:Bul Pgia   1
choose mutation scheme:  random mutation: 1 ,swap_mutate: 2 ,insertion_mutate: 31
for hyper press 1 for normal press 00
chose speciation type :  1: threshold speciation  2:k-means clustering 3: none 1
13.740600000000006

number of groups
 23

selection pressure: 1.229787234042553   Diversity:  11.740600000000006
 Best:c,r,x,o,r,c,J,W,n,w,^,`, ,fittness: 1080  Mean: 1328.4 ,Variance: 248.4000000000001 Time :  1.1056649000000007  ticks: 1.1056122779846191
11.740600000000006

number of groups
 1

selection pressure: 1.1940464177598387   Diversity:  11.213200000000004
 Best:c,r,x,o,r,l,J,W,n,w,^,`, ,fittness: 990  Mean: 1182.3 ,Variance: 192.29999999999995 Time :  1.8645506999999988  ticks: 1.8645596504211426
9.740600000000006

number of groups
 3

selection pressure: 1.1301886792452833   Diversity:  8.403400000000001
 Best:c,r,d,o,J,c,k,d,d,w,C,W, ,fittness: 900  Mean: 1017.3 ,Variance: 117.29999999999995 Time :  2.9457986  ticks: 2.9452688694000244
11.740600000000006

number of groups
 8

selection pressure: 1.222191400832178   Diversity:  8.581399999999997
 Best:c,r,d,o,h,r,(,W,K,w,o,w, ,fittness: 720  Mean: 880.2 ,Variance: 160.20000000000005 Time :  3.9797000999999987  ticks: 3.9795408248901367
9.740600000000006

number of groups
 1

selection pressure: 1.4020332717190394   Diversity:  8.453199999999999
 Best:J,r,r,o,r,_,d,d,w,e,o,w, ,fittness: 540  Mean: 757.5 ,Variance: 217.5 Time :  5.051108799999998  ticks: 5.050822734832764
7.740600000000006

number of groups
 2
```

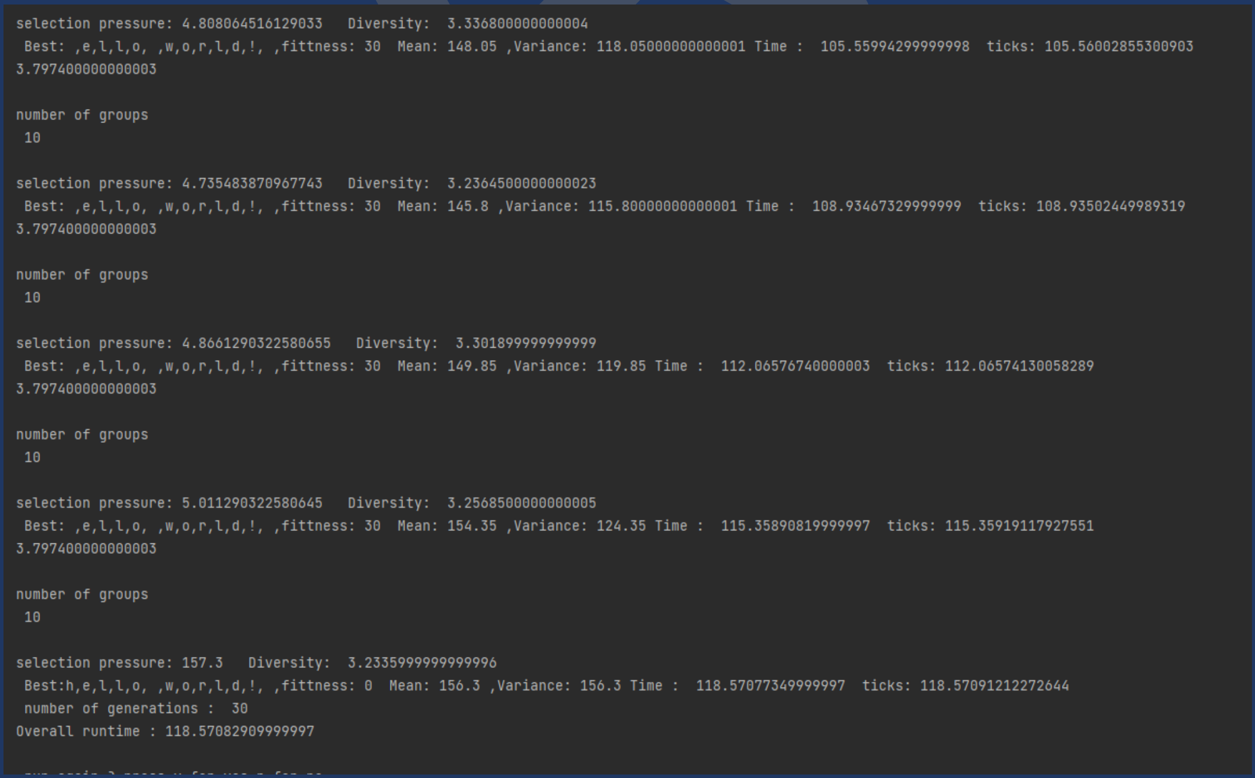


```
selection pressure: 5.509677419354837   Diversity:  3.6545999999999985
 Best:h,e,l,l,d, ,w,o,r,l,d,!, ,fittness: 30  Mean: 169.8 ,Variance: 139.8 Time :  32.3758984  ticks: 32.3759331703186
5.740600000000006

number of groups
 5

selection pressure: 5.074193548387097   Diversity:  3.515199999999999
 Best:h,e,l,l,d, ,w,o,r,l,d,!, ,fittness: 30  Mean: 156.3 ,Variance: 126.30000000000001 Time :  33.4371263  ticks: 33.43700313568115
5.740600000000006

number of groups
 5

selection pressure: 148.29999999999998   Diversity:  3.5214
 Best:h,e,l,l,o, ,w,o,r,l,d,!, ,fittness: 0  Mean: 147.3 ,Variance: 147.3 Time :  34.522199400000005  ticks: 34.5214684009552
 number of generations :  30
```

Threshold=10 , population size=200

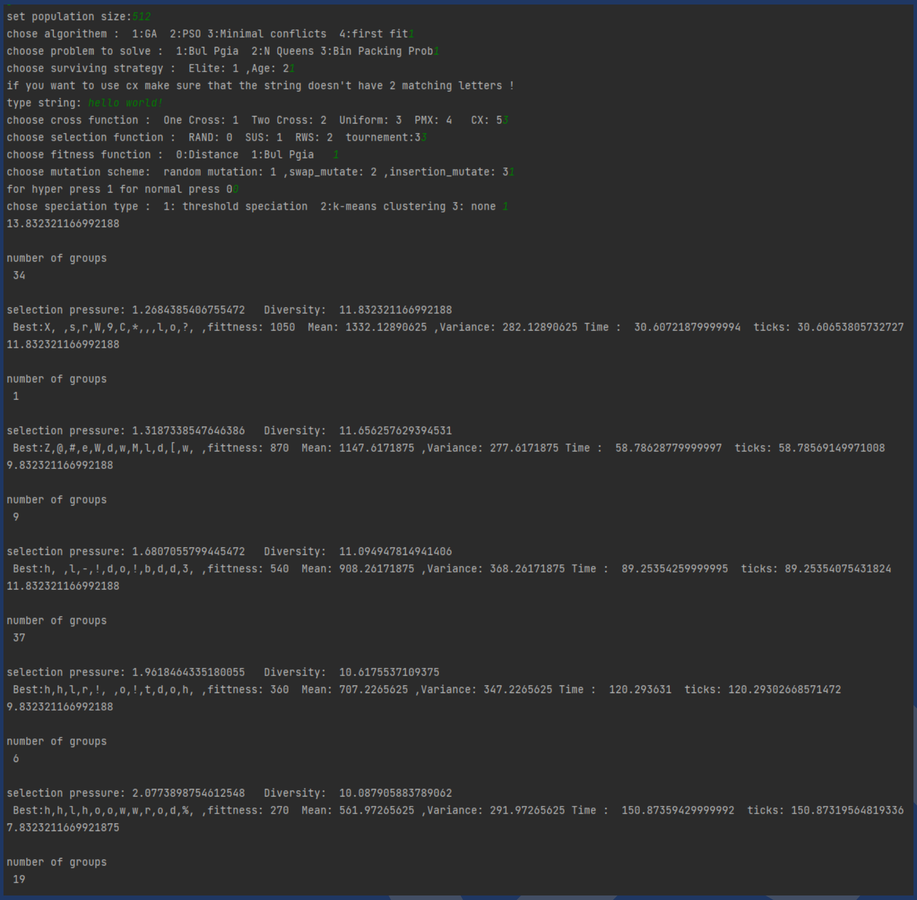


```
set population size:200
chose algorithem :  1:GA  2:PSO 3:Minimal conflicts  4:first fit1
choose problem to solve :  1:Bul Pgia  2:N Queens 3:Bin Packing Prob1
choose surviving strategy :  Elite: 1 ,Age: 21
if you want to use cx make sure that the string doesn't have 2 matching letters !
type string: hello world!
choose cross function :  One Cross: 1  Two Cross: 2  Uniform: 3  PMX: 4   CX: 53
choose selection function :  RAND: 0  SUS: 1  RWS: 2  tournement:33
choose fitness function :  0:Distance  1:Bul Pgia   1
choose mutation scheme:  random mutation: 1 ,swap_mutate: 2 ,insertion_mutate: 31
for hyper press 1 for normal press 00
chose speciation type :  1: threshold speciation  2:k-means clustering 3: none 1
13.797400000000003

number of groups
 26

selection pressure: 1.3361755802219974   Diversity:  11.797400000000003
 Best:K,C,8,r,d,C,!,5,2,o,l,Y, ,fittness: 990  Mean: 1323.15 ,Variance: 333.1500000000001 Time :  4.4004397000000495  ticks: 4.400535583496094
11.797400000000003

number of groups
 1

selection pressure: 1.3895191122071515   Diversity:  11.357349999999999
 Best:K,u,r,r,l,C,!,_,d,o,o,Y, ,fittness: 810  Mean: 1125.9 ,Variance: 315.9000000000001 Time :  7.949089600000036  ticks: 7.949449062347412
9.797400000000003

number of groups
 3

selection pressure: 1.4767054908485857   Diversity:  9.936499999999995
 Best:K,e,d,d,w,R,!,r,!,r,l,^, ,fittness: 600  Mean: 886.5 ,Variance: 286.5 Time :  12.180820700000027  ticks: 12.180525779724121
9.797400000000003

number of groups
 10

selection pressure: 1.639904988123515   Diversity:  8.721600000000006
 Best:w,u,d,r,w,l,!,o,d,r,l,o, ,fittness: 420  Mean: 689.4 ,Variance: 269.4 Time :  16.414341500000035  ticks: 16.414305925369263
7.797400000000003

number of groups
 7
```

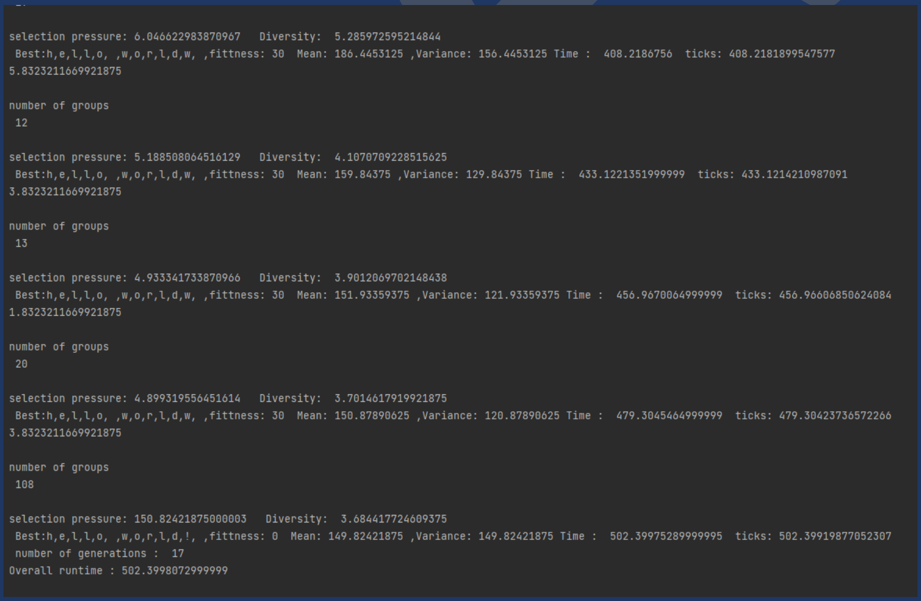


```
selection pressure: 4.808064516129033   Diversity:  3.336800000000004
 Best: ,e,l,l,o, ,w,o,r,l,d,!, ,fittness: 30  Mean: 148.05 ,Variance: 118.05000000000001 Time :  105.55994299999998  ticks: 105.56002855300903
3.797400000000003

number of groups
 10

selection pressure: 4.735483870967743   Diversity:  3.2364500000000023
 Best: ,e,l,l,o, ,w,o,r,l,d,!, ,fittness: 30  Mean: 145.8 ,Variance: 115.80000000000001 Time :  108.93467329999999  ticks: 108.93502449989319
3.797400000000003

number of groups
 10

selection pressure: 4.8661290322580655   Diversity:  3.301899999999999
 Best: ,e,l,l,o, ,w,o,r,l,d,!, ,fittness: 30  Mean: 149.85 ,Variance: 119.85 Time :  112.06576740000003  ticks: 112.06574130058289
3.797400000000003

number of groups
 10

selection pressure: 5.011290322580645   Diversity:  3.2568500000000005
 Best: ,e,l,l,o, ,w,o,r,l,d,!, ,fittness: 30  Mean: 154.35 ,Variance: 124.35 Time :  115.35890819999997  ticks: 115.35919117927551
3.797400000000003

number of groups
 10

selection pressure: 157.3   Diversity:  3.2335999999999996
 Best:h,e,l,l,o, ,w,o,r,l,d,!, ,fittness: 0  Mean: 156.3 ,Variance: 156.3 Time :  118.57077349999997  ticks: 118.57091212272644
 number of generations :  30
Overall runtime : 118.57082909999997
```

## Threshold=25 , population size=512

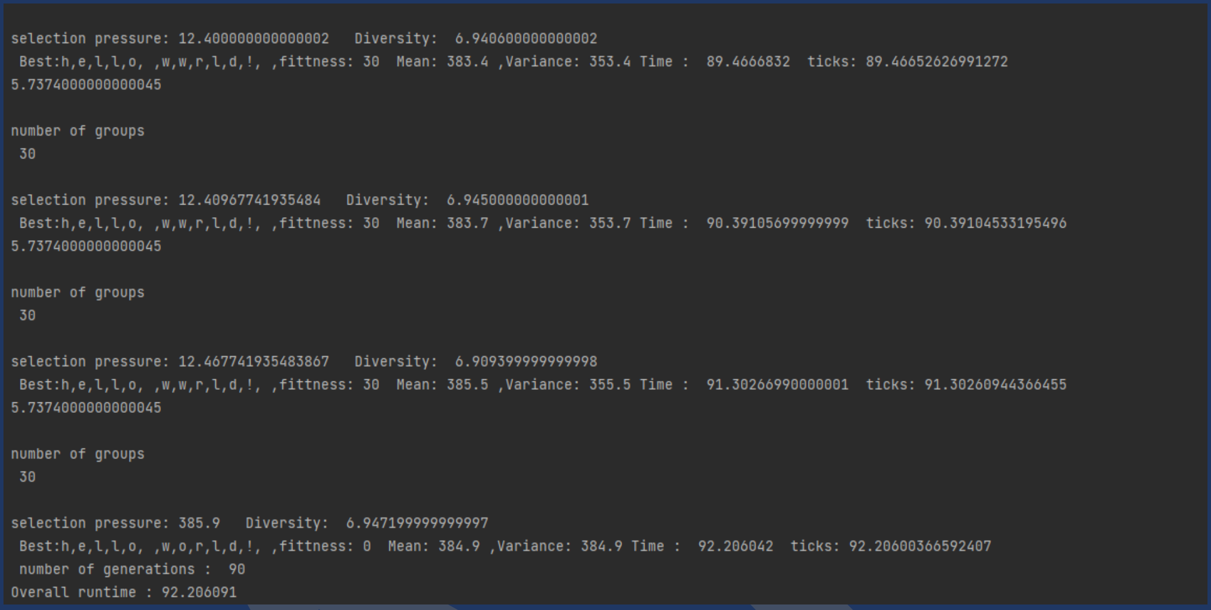


```
set population size:512
chose algorithem :  1:GA  2:PSO 3:Minimal conflicts  4:first fit1
choose problem to solve :  1:Bul Pgia  2:N Queens 3:Bin Packing Prob1
choose surviving strategy :  Elite: 1 ,Age: 21
if you want to use cx make sure that the string doesn't have 2 matching letters !
type string: hello world!
choose cross function :  One Cross: 1  Two Cross: 2  Uniform: 3  PMX: 4   CX: 53
choose selection function :  RAND: 0  SUS: 1  RWS: 2  tournement:33
choose fitness function :  0:Distance  1:Bul Pgia   1
choose mutation scheme:  random mutation: 1 ,swap_mutate: 2 ,insertion_mutate: 31
for hyper press 1 for normal press 00
chose speciation type :  1: threshold speciation  2:k-means clustering 3: none 1
13.832321166992188

number of groups
 34

selection pressure: 1.2684385406755472   Diversity:  11.832321166992188
 Best:X, ,s,r,W,9,C,*,,,l,o,?, ,fittness: 1050  Mean: 1332.12890625 ,Variance: 282.12890625 Time :  30.60721879999994  ticks: 30.60653805732727
11.832321166992188

number of groups
 1

selection pressure: 1.3187338547646386   Diversity:  11.656257629394531
 Best:Z,@,#,e,W,d,w,M,l,d,[,w, ,fittness: 870  Mean: 1147.6171875 ,Variance: 277.6171875 Time :  58.78628779999997  ticks: 58.78569149971008
9.832321166992188

number of groups
 9

selection pressure: 1.6807055799445472   Diversity:  11.094947814941406
 Best:h, ,l,-,!,d,o,!,b,d,d,3, ,fittness: 540  Mean: 908.26171875 ,Variance: 368.26171875 Time :  89.25354259999995  ticks: 89.25354075431824
11.832321166992188

number of groups
 37

selection pressure: 1.9618464335180055   Diversity:  10.6175537109375
 Best:h,h,l,r,!, ,o,!,t,d,o,h, ,fittness: 360  Mean: 707.2265625 ,Variance: 347.2265625 Time :  120.293631  ticks: 120.29302668571472
9.832321166992188

number of groups
 6

selection pressure: 2.0773898754612548   Diversity:  10.087905883789062
 Best:h,h,l,h,o,o,w,w,r,o,d,%, ,fittness: 270  Mean: 561.97265625 ,Variance: 291.97265625 Time :  150.87359429999992  ticks: 150.87319564819336
7.8323211669921875

number of groups
 19
```



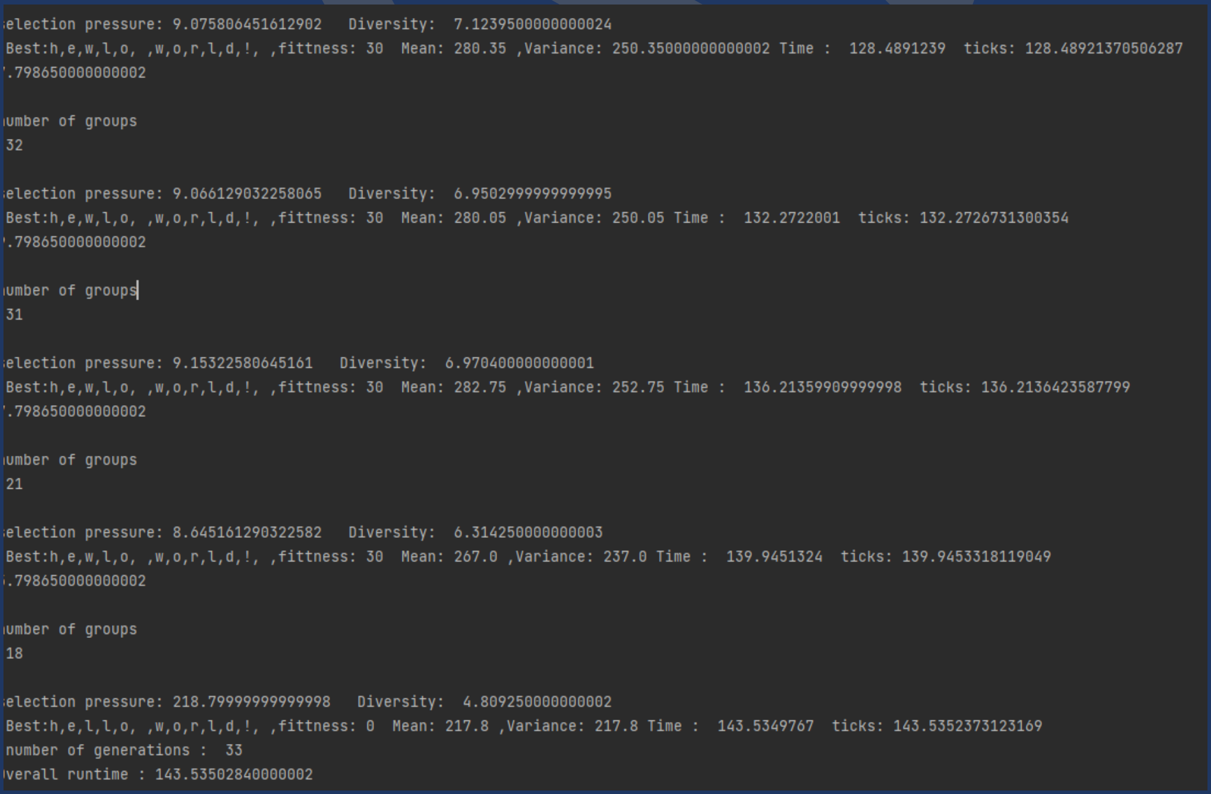
```
selection pressure: 6.046622983870967   Diversity:  5.285972595214844
 Best:h,e,l,l,o, ,w,o,r,l,d,w, ,fittness: 30  Mean: 186.4453125 ,Variance: 156.4453125 Time :  408.2186756  ticks: 408.2181899547577
5.8323211669921875

number of groups
 12

selection pressure: 5.188508064516129   Diversity:  4.1070709228515625
 Best:h,e,l,l,o, ,w,o,r,l,d,w, ,fittness: 30  Mean: 159.84375 ,Variance: 129.84375 Time :  433.1221351999999  ticks: 433.1214210987091
3.8323211669921875

number of groups
 13

selection pressure: 4.933341733870966   Diversity:  3.9012069702148438
 Best:h,e,l,l,o, ,w,o,r,l,d,w, ,fittness: 30  Mean: 151.93359375 ,Variance: 121.93359375 Time :  456.9670064999999  ticks: 456.96606850624084
1.8323211669921875

number of groups
 20

selection pressure: 4.899319556451614   Diversity:  3.7014617919921875
 Best:h,e,l,l,o, ,w,o,r,l,d,w, ,fittness: 30  Mean: 150.87890625 ,Variance: 120.87890625 Time :  479.3045464999999  ticks: 479.30423736572266
3.8323211669921875

number of groups
 108

selection pressure: 150.82421875000003   Diversity:  3.684417724609375
 Best:h,e,l,l,o, ,w,o,r,l,d,!, ,fittness: 0  Mean: 149.82421875 ,Variance: 149.82421875 Time :  502.39975289999995  ticks: 502.39919877052307
 number of generations :  17
Overall runtime : 502.3998072999999
```

Threshold=30 , population size=100



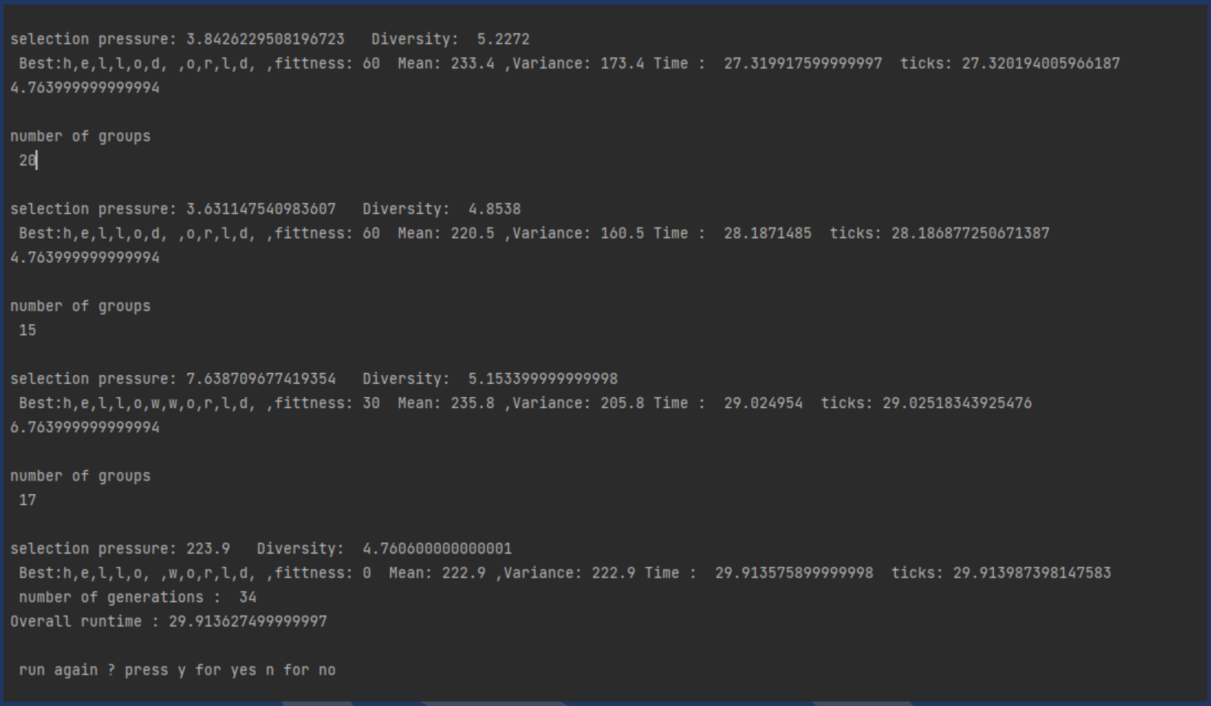
```
selection pressure: 12.400000000000002   Diversity:  6.940600000000002
 Best:h,e,l,l,o, ,w,w,r,l,d,!, ,fittness: 30  Mean: 383.4 ,Variance: 353.4 Time :  89.4666832  ticks: 89.46652626991272
5.7374000000000045

number of groups
 30

selection pressure: 12.40967741935484   Diversity:  6.945000000000001
 Best:h,e,l,l,o, ,w,w,r,l,d,!, ,fittness: 30  Mean: 383.7 ,Variance: 353.7 Time :  90.39105699999999  ticks: 90.39104533195496
5.7374000000000045

number of groups
 30

selection pressure: 12.467741935483867   Diversity:  6.909399999999998
 Best:h,e,l,l,o, ,w,w,r,l,d,!, ,fittness: 30  Mean: 385.5 ,Variance: 355.5 Time :  91.30266990000001  ticks: 91.30260944366455
5.7374000000000045

number of groups
 30

selection pressure: 385.9   Diversity:  6.947199999999997
 Best:h,e,l,l,o, ,w,o,r,l,d,!, ,fittness: 0  Mean: 384.9 ,Variance: 384.9 Time :  92.206042  ticks: 92.20600366592407
 number of generations :  90
Overall runtime : 92.206091
```

Threshold=30 , population size=200



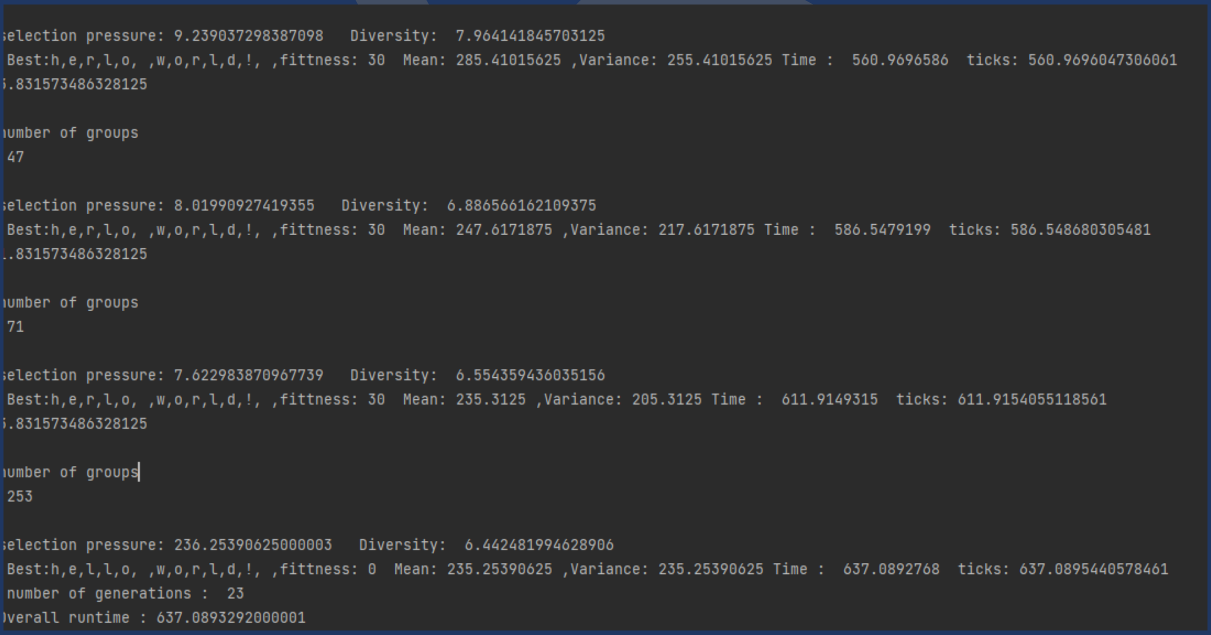
```
election pressure: 9.075806451612902   Diversity:  7.123950000000024
Best:h,e,w,l,o, ,w,o,r,l,d,!, ,fittness: 30  Mean: 280.35 ,Variance: 250.35000000000002 Time :  128.4891239  ticks: 128.48921370506287
.798650000000002

umber of groups
32

election pressure: 9.066129032258065   Diversity:  6.9502999999999995
Best:h,e,w,l,o, ,w,o,r,l,d,!, ,fittness: 30  Mean: 280.05 ,Variance: 250.05 Time :  132.2722001  ticks: 132.2726731300354
.798650000000002

umber of groups
31

election pressure: 9.15322580645161   Diversity:  6.970400000000001
Best:h,e,w,l,o, ,w,o,r,l,d,!, ,fittness: 30  Mean: 282.75 ,Variance: 252.75 Time :  136.21359909999998  ticks: 136.2136423587799
.798650000000002

umber of groups
21

election pressure: 8.645161290322582   Diversity:  6.314250000000003
Best:h,e,w,l,o, ,w,o,r,l,d,!, ,fittness: 30  Mean: 267.0 ,Variance: 237.0 Time :  139.9451324  ticks: 139.9453318119049
.798650000000002

umber of groups
18

election pressure: 218.79999999999998   Diversity:  4.809250000000002
Best:h,e,l,l,o, ,w,o,r,l,d,!, ,fittness: 0  Mean: 217.8 ,Variance: 217.8 Time :  143.5349767  ticks: 143.5352373123169
number of generations :  33
verall runtime : 143.53502840000002
```

Threshold=32 , population size=200



```
selection pressure: 3.8426229508196723   Diversity:  5.2272
 Best:h,e,l,l,o,d, ,o,r,l,d, ,fittness: 60  Mean: 233.4 ,Variance: 173.4 Time :  27.319917599999997  ticks: 27.320194005966187
4.763999999999994

number of groups
 20

selection pressure: 3.631147540983607   Diversity:  4.8538
 Best:h,e,l,l,o,d, ,o,r,l,d, ,fittness: 60  Mean: 220.5 ,Variance: 160.5 Time :  28.1871485  ticks: 28.186877250671387
4.763999999999994

number of groups
 15

selection pressure: 7.638709677419354   Diversity:  5.153399999999998
 Best:h,e,l,l,o,w,w,o,r,l,d, ,fittness: 30  Mean: 235.8 ,Variance: 205.8 Time :  29.024954  ticks: 29.02518343925476
6.763999999999994

number of groups
 17

selection pressure: 223.9   Diversity:  4.760600000000001
 Best:h,e,l,l,o, ,w,o,r,l,d, ,fittness: 0  Mean: 222.9 ,Variance: 222.9 Time :  29.913575899999998  ticks: 29.913987398147583
 number of generations :  34
Overall runtime : 29.913627499999997

 run again ? press y for yes n for no
```

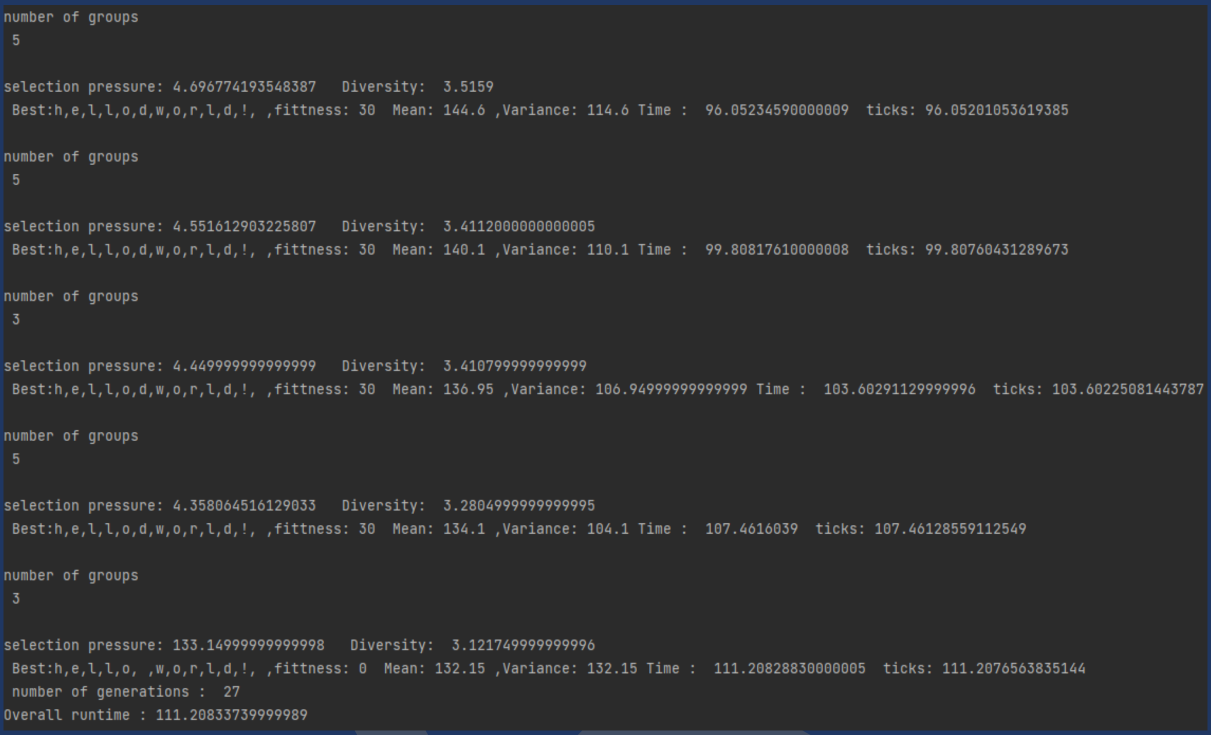
Threshold=76 , population size=512



```
election pressure: 9.239037298387098   Diversity:  7.964141845703125
 Best:h,e,r,l,o, ,w,o,r,l,d,!, ,fittness: 30  Mean: 285.41015625 ,Variance: 255.41015625 Time :  560.9696586  ticks: 560.9696047306061
.831573486328125

umber of groups
 47

election pressure: 8.01990927419355   Diversity:  6.886566162109375
 Best:h,e,r,l,o, ,w,o,r,l,d,!, ,fittness: 30  Mean: 247.6171875 ,Variance: 217.6171875 Time :  586.5479199  ticks: 586.548680305481
.831573486328125

umber of groups
 71

election pressure: 7.622983870967739   Diversity:  6.554359436035156
 Best:h,e,r,l,o, ,w,o,r,l,d,!, ,fittness: 30  Mean: 235.3125 ,Variance: 205.3125 Time :  611.9149315  ticks: 611.9154055118561
.831573486328125

umber of groups
 253

election pressure: 236.25390625000003   Diversity:  6.442481994628906
 Best:h,e,l,l,o, ,w,o,r,l,d,!, ,fittness: 0  Mean: 235.25390625 ,Variance: 235.25390625 Time :  637.0892768  ticks: 637.0895440578461
 number of generations :  23
verall runtime : 637.0893292000001
```

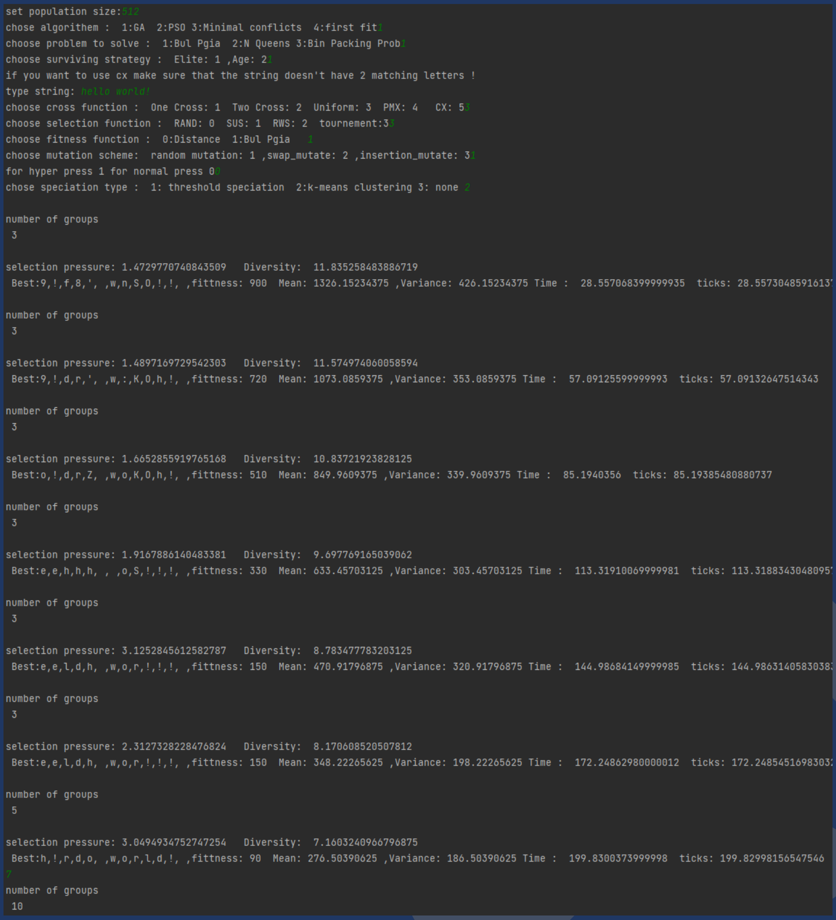
## K-means clustering with optimal k each iteration:

## Population size =200

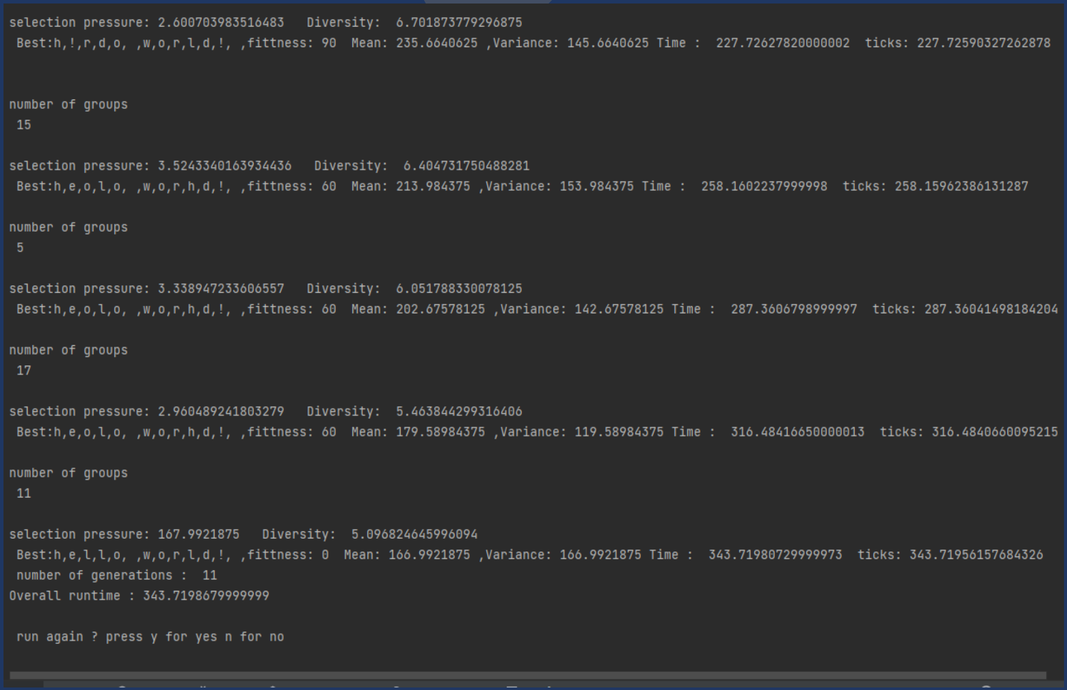


```
number of groups
 5

selection pressure: 4.696774193548387   Diversity:  3.5159
 Best:h,e,l,l,o,d,w,o,r,l,d,!, ,fittness: 30  Mean: 144.6 ,Variance: 114.6 Time :  96.05234590000009  ticks: 96.05201053619385

number of groups
 5

selection pressure: 4.551612903225807   Diversity:  3.4112000000000005
 Best:h,e,l,l,o,d,w,o,r,l,d,!, ,fittness: 30  Mean: 140.1 ,Variance: 110.1 Time :  99.80817610000008  ticks: 99.80760431289673

number of groups
 3

selection pressure: 4.449999999999999   Diversity:  3.410799999999999
 Best:h,e,l,l,o,d,w,o,r,l,d,!, ,fittness: 30  Mean: 136.95 ,Variance: 106.94999999999999 Time :  103.60291129999996  ticks: 103.60225081443787

number of groups
 5

selection pressure: 4.358064516129033   Diversity:  3.2804999999999995
 Best:h,e,l,l,o,d,w,o,r,l,d,!, ,fittness: 30  Mean: 134.1 ,Variance: 104.1 Time :  107.4616039  ticks: 107.46128559112549

number of groups
 3

selection pressure: 133.14999999999998   Diversity:  3.121749999999996
 Best:h,e,l,l,o, ,w,o,r,l,d,!, ,fittness: 0  Mean: 132.15 ,Variance: 132.15 Time :  111.20828830000005  ticks: 111.2076563835144
 number of generations :  27
Overall runtime : 111.20833739999989
```

## Population size=512



```
set population size:512
chose algorithem :  1:GA  2:PSO 3:Minimal conflicts  4:first fit1
choose problem to solve :  1:Bul Pgia  2:N Queens 3:Bin Packing Prob1
choose surviving strategy :  Elite: 1 ,Age: 21
if you want to use cx make sure that the string doesn't have 2 matching letters !
type string: hello world!
choose cross function :  One Cross: 1  Two Cross: 2  Uniform: 3  PMX: 4   CX: 53
choose selection function :  RAND: 0  SUS: 1  RWS: 2  tournement:33
choose fitness function :  0:Distance  1:Bul Pgia   1
choose mutation scheme:  random mutation: 1 ,swap_mutate: 2 ,insertion_mutate: 31
for hyper press 1 for normal press 00
chose speciation type :  1: threshold speciation  2:k-means clustering 3: none 2

number of groups
 3

selection pressure: 1.4729770740843509   Diversity:  11.835258483886719
 Best:9,!,f,8,', ,w,n,S,O,!,!, ,fittness: 900  Mean: 1326.15234375 ,Variance: 426.15234375 Time :  28.557068399999935  ticks: 28.5573048591613

number of groups
 3

selection pressure: 1.4897169729542303   Diversity:  11.574974060058594
 Best:9,!,d,r,', ,w,:,K,O,h,!, ,fittness: 720  Mean: 1073.0859375 ,Variance: 353.0859375 Time :  57.09125599999993  ticks: 57.09132647514343

number of groups
 3

selection pressure: 1.6652855919765168   Diversity:  10.83721923828125
 Best:o,!,d,r,Z, ,w,o,K,O,h,!, ,fittness: 510  Mean: 849.9609375 ,Variance: 339.9609375 Time :  85.1940356  ticks: 85.19385480880737

number of groups
 3

selection pressure: 1.9167886140483381   Diversity:  9.697769165039062
 Best:e,e,h,h,h, , ,o,S,!,!,!, ,fittness: 330  Mean: 633.45703125 ,Variance: 303.45703125 Time :  113.31910069999981  ticks: 113.318834304809

number of groups
 3

selection pressure: 3.1252845612582787   Diversity:  8.783477783203125
 Best:e,e,l,d,h, ,w,o,r,!,!,!, ,fittness: 150  Mean: 470.91796875 ,Variance: 320.91796875 Time :  144.98684149999985  ticks: 144.986314058303

number of groups
 3

selection pressure: 2.3127328228476824   Diversity:  8.170608520507812
 Best:e,e,l,d,h, ,w,o,r,!,!,!, ,fittness: 150  Mean: 348.22265625 ,Variance: 198.22265625 Time :  172.24862980000012  ticks: 172.248545169830

number of groups
 5

selection pressure: 3.0494934752747254   Diversity:  7.1603240966796875
 Best:h,!,r,d,o, ,w,o,r,l,d,!, ,fittness: 90  Mean: 276.50390625 ,Variance: 186.50390625 Time :  199.8300373999998  ticks: 199.82998156547546
7
number of groups
 10
```



```
selection pressure: 2.600703983516483   Diversity:  6.701873779296875
 Best:h,!,r,d,o, ,w,o,r,l,d,!, ,fittness: 90  Mean: 235.6640625 ,Variance: 145.6640625 Time :  227.72627820000002  ticks: 227.72590327262878


number of groups
 15

selection pressure: 3.5243340163934436   Diversity:  6.404731750488281
 Best:h,e,o,l,o, ,w,o,r,h,d,!, ,fittness: 60  Mean: 213.984375 ,Variance: 153.984375 Time :  258.1602237999998  ticks: 258.15962386131287

number of groups
 5

selection pressure: 3.338947233606557   Diversity:  6.051788330078125
 Best:h,e,o,l,o, ,w,o,r,h,d,!, ,fittness: 60  Mean: 202.67578125 ,Variance: 142.67578125 Time :  287.3606798999997  ticks: 287.36041498184204

number of groups
 17

selection pressure: 2.960489241803279   Diversity:  5.463844299316406
 Best:h,e,o,l,o, ,w,o,r,h,d,!, ,fittness: 60  Mean: 179.58984375 ,Variance: 119.58984375 Time :  316.48416650000013  ticks: 316.4840660095215

number of groups
 11

selection pressure: 167.9921875   Diversity:  5.096824645996094
 Best:h,e,l,l,o, ,w,o,r,l,d,!, ,fittness: 0  Mean: 166.9921875 ,Variance: 166.9921875 Time :  343.71980729999973  ticks: 343.71956157684326
 number of generations :  11
Overall runtime : 343.7198679999999

 run again ? press y for yes n for no
```

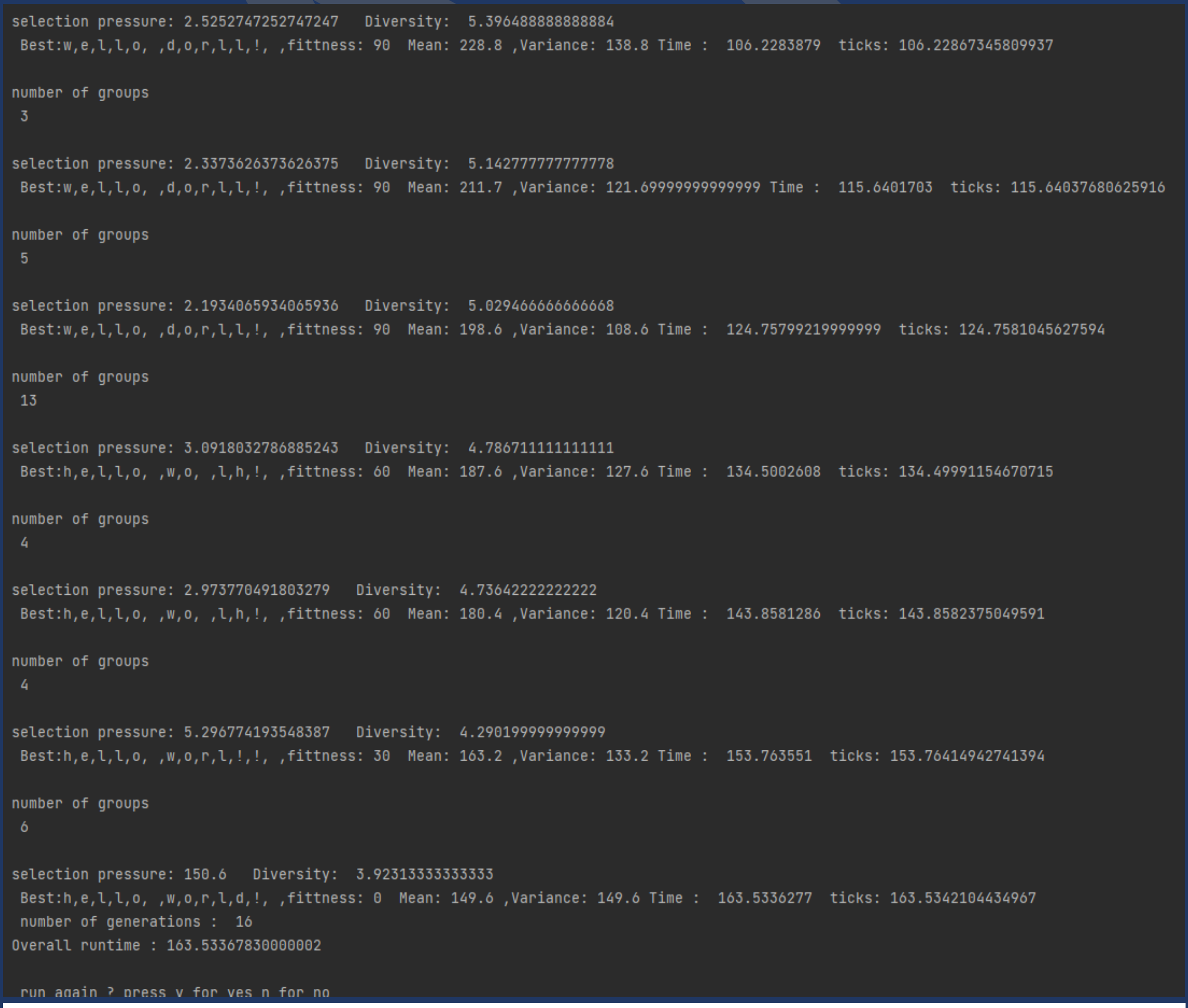
## סעיף 5:

## תוצאות כלליות בשביל ניתוח סופי :

| population size | exploration to exploitaion | Hyper triggered | mutation parameter=0.25 |
|---|---|---|---|
| 100 | generation:33 ,runtime: 31 | generations:28 ,runtime:32 | generation:35 ,runtime: 35.7 |
| 300 | generation:21 ,runtime: 209 | generations:19 ,runtime:182 | generation:20 ,runtime: 195 |

| | Threshold | | | K-means clustering |
|---|---|---|---|---|
| population size | 5 % of poulation size | 15% of population | 30 | |
| 100 | generation:30 runtime: 34.5 | generations:34 runtime:30 | generation:90 runtime: 90 | generation:28 runtime: 25 |
| 200 | generation:30 runtime: 118 | generations:29 runtime:129 | generation:33 runtime: 143.5 | generation:27 runtime: 111 |
| 300 | | | | Generations:16 runtime=163 |
| 512 | generation:17 runtime: 502 | generations:23 runtime:637 | generation:17 runtime: 492 | generation:11 runtime: 340 |

## k-means-clustering + population 300:



```
selection pressure: 2.5252747252747247   Diversity:  5.396488888888884
 Best:w,e,l,l,o, ,d,o,r,l,l,!, ,fittness: 90  Mean: 228.8 ,Variance: 138.8 Time :  106.2283879  ticks: 106.22867345809937

number of groups
 3

selection pressure: 2.3373626373626375   Diversity:  5.142777777777778
 Best:w,e,l,l,o, ,d,o,r,l,l,!, ,fittness: 90  Mean: 211.7 ,Variance: 121.69999999999999 Time :  115.6401703  ticks: 115.64037680625916

number of groups
 5

selection pressure: 2.1934065934065936   Diversity:  5.029466666666668
 Best:w,e,l,l,o, ,d,o,r,l,l,!, ,fittness: 90  Mean: 198.6 ,Variance: 108.6 Time :  124.75799219999999  ticks: 124.7581045627594

number of groups
 13

selection pressure: 3.0918032786885243   Diversity:  4.786711111111111
 Best:h,e,l,l,o, ,w,o, ,l,h,!, ,fittness: 60  Mean: 187.6 ,Variance: 127.6 Time :  134.5002608  ticks: 134.49991154670715

number of groups
 4

selection pressure: 2.973770491803279   Diversity:  4.73642222222222
 Best:h,e,l,l,o, ,w,o, ,l,h,!, ,fittness: 60  Mean: 180.4 ,Variance: 120.4 Time :  143.8581286  ticks: 143.8582375049591

number of groups
 4

selection pressure: 5.296774193548387   Diversity:  4.290199999999999
 Best:h,e,l,l,o, ,w,o,r,l,!,!, ,fittness: 30  Mean: 163.2 ,Variance: 133.2 Time :  153.763551  ticks: 153.76414942741394

number of groups
 6

selection pressure: 150.6   Diversity:  3.92313333333333
 Best:h,e,l,l,o, ,w,o,r,l,d,!, ,fittness: 0  Mean: 149.6 ,Variance: 149.6 Time :  163.5336277  ticks: 163.5342104434967
 number of generations :  16
Overall runtime : 163.53367830000002

 run again ? press y for yes n for no
```

**סעיף 6+7:**

מכיוון שלא הצלחנו לשחזר את random immigrants שאיבדנו עם רוב חלק ב אזי נשווה את הפתרון האופטימלי שקיבלנו מ -k means clustering :

א. שלימות : האלגוריתם עם k means תמיד מביא לפתרון סופי , כיוון שהוא תמיד יוצא מ local optima ובכל האיטירציות שהרצנו , למראת שגם כן האלגוריתם הקודם תמיד היה מוציא פתרון , אך לא היה מוגבל מבחינת ריצות ואז היה רץ עד max iterations אך בסופו של דבר שניהן מגיעות לפתרון גם אם לא היה האופטימלי.

ב. מכיוון ש k-means יוצא מ local optima מצאנו שבכל המקרים שעברנו עליהן מול האלגוריתם הישן היה תמיד מוצא את הפתרון האופטימלי אך בפחות צעדים , ספציפית עם אוכלוסייה קטנה כמו 100 האלגוריתם הישן לא היה תמיד מגיע לפתרון למראת ה k-means.

ג. כמו שאמרנו בסעיפים קודמים , הפתרונות הן איכותיות יותר למשל ב k-means עם אוכלוסייה קטנה size=100 ,הגענו לפתרון תוך 28 generations למראת האלגוריתם הישן שהגיע תוך 35 generations שהיה עם mutation rate=0.25 . (כל התוצאות הללו נמצאות בטבלות בסעיפים הקודמים ) ,גם כן נראה שבאוכלוסיות יותר גדולות לאלגוריתם החדש לוקח 16 generations למראת הישן שלוקח 20 generation .

ד. מבחינת זמן אין הרבה מה להשוות כיוון שכל אטירציה שיש בה חישוב של diversity של כל האוכלוסייה לוקחת יותר זמן מהאלגוריתם הישן כדי למצוא תשובה .
הכוונה שהאלגוריתם של המעבדה קודמת היה מסיים בפחות משניה לפעמים למרות האלגוריתם החדש שלוקח לא פחות מ 5 שניות במקרה הטוב ביותר .

שיפורים אפשריים בעתיד:

1. Distance function ליניארי , או לפחות יותר טוב ממה שיש לנו עכשיו , אולי שימוש ב hamming distance או Euclidian distance היו אופציות יותר הגיוניות .
2. אלגוריתם random immigrants והאלגוריתם הממטי היה מדהים ומעניין , אך לא הצלחנו לשחזר אותו ,לכך עלינו להיות יותר זהרים בעתיד.
3. 
4. למראת שכל הפונקציות ככליות עלינו לעשות אותם מודולריות בעתיד.

חלק ב :

כמו ששלחנו קודם בבקשה לאיחור ,לא הספקנו לשחזר אותו .