In [4]:

```python
import numpy as np

def write_distance_matrix(n, mean, sigma):
    distance_matrix = np.zeros((n, n))

    for row in range(n):
        for col in range(n):
            distance = 0
            while distance <= 0:
                distance = np.random.normal(mean, sigma)
                distance_matrix[row][col] = distance
    return distance_matrix
```

In [1]:

```
import random2
import time

from py2opt.solver import Solver


class RouteFinder:
    def __init__(self, distance_matrix, cities_names, iterations=5, wri
        self.distance_matrix = distance_matrix
        self.iterations = iterations
        self.writer_flag = writer_flag
        self.cities_names = cities_names

    def solve(self):
        start_time = round(time.time() * 1000)
        elapsed_time = 0
        iteration = 0
        best_distance = 0
        best_route = []
        best_distances = []

        while iteration < self.iterations:
            num_cities = len(self.distance_matrix)
            print(round(elapsed_time), 'msec')
            initial_route = [0] + random2.sample(range(1, num_cities),
            tsp = Solver(self.distance_matrix, initial_route)
            new_route, new_distance, distances = tsp.two_opt()

            if iteration == 0:
                best_distance = new_distance
                best_route = new_route
            else:
                pass

            if new_distance < best_distance:
                best_distance = new_distance
                best_route = new_route
                best_distances = distances

            elapsed_time = round(time.time() * 1000) - start_time
            iteration += 1

        if self.writer_flag:
            self.writer(best_route, best_distance, self.cities_names)

        if self.cities_names:
            best_route = [self.cities_names[i] for i in best_route]
            return best_distance, best_route
        else:
            return best_distance, best_route
```

In [2]:

```
import itertools
import numpy as np


class Solver:
    def __init__(self, distance_matrix, initial_route):
        self.distance_matrix = distance_matrix
        self.num_cities = len(self.distance_matrix)
        self.initial_route = initial_route
        self.best_route = []
        self.best_distance = 0
        self.distances = []

    def update(self, new_route, new_distance):
        self.best_distance = new_distance
        self.best_route = new_route
        return self.best_distance, self.best_route

    def exhaustive_search(self):
        self.best_route = [0] + list(range(1, self.num_cities))
        self.best_distance = self.calculate_path_dist(self.distance_mat

        for new_route in itertools.permutations(list(range(1, self.num_
            new_distance = self.calculate_path_dist(self.distance_matri

            if new_distance < self.best_distance:
                self.update([0] + list(new_route[:]), new_distance)
                self.distances.append(self.best_distance)

        return self.best_route, self.best_distance, self.distances

    def two_opt(self, improvement_threshold=0.01):
        self.best_route = self.initial_route
        self.best_distance = self.calculate_path_dist(self.distance_mat
        improvement_factor = 1

        while improvement_factor > improvement_threshold:
            previous_best = self.best_distance
            for swap_first in range(1, self.num_cities - 2):
                for swap_last in range(swap_first + 1, self.num_cities
                    before_start = self.best_route[swap_first - 1]
                    start = self.best_route[swap_first]
                    end = self.best_route[swap_last]
                    after_end = self.best_route[swap_last+1]
                    before = self.distance_matrix[before_start][start]
                    after = self.distance_matrix[before_start][end] + s
                    if after < before:
                        new_route = self.swap(self.best_route, swap_fir
                        new_distance = self.calculate_path_dist(self.di
                        self.update(new_route, new_distance)

            improvement_factor = 1 - self.best_distance/previous_best
        return self.best_route, self.best_distance, self.distances

    def calculate_path_dist(distance_matrix, path):
        """
```

```
        This method calculates the total distance between the first cit
        """
        path_distance = 0
        for ind in range(len(path) - 1):
            path_distance += distance_matrix[path[ind]][path[ind + 1]]
        return float("{0:.2f}".format(path_distance))

    def swap(path, swap_first, swap_last):
        path_updated = np.concatenate((path[0:swap_first],
                                       path[swap_last:-len(path) + swap
                                       path[swap_last + 1:len(path)]))
        return path_updated.tolist()
```

In [28]:
```
cities = write_distance_matrix(100,100,100)
cities_names = list(range(100))
print(cities_names)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 2
0, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 3
8, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 5
6, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 7
4, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 9
2, 93, 94, 95, 96, 97, 98, 99]
```

In [30]:
```
route_finder = RouteFinder(cities, cities_names, iterations=1000)
best_distance, best_route = route_finder.solve()
print(best_distance)
print(best_route)
```

```
93991 msec
94091 msec

94158 msec
94288 msec
94347 msec
94416 msec
94482 msec
94645 msec
94796 msec
94862 msec
94931 msec
95097 msec
2834.47
[0, 10, 35, 50, 62, 89, 29, 84, 23, 5, 74, 87, 52, 4, 85, 92, 2, 39, 93,
75, 98, 8, 97, 49, 22, 66, 34, 82, 19, 37, 81, 20, 71, 3, 44, 86, 17, 12,
56, 53, 45, 55, 73, 33, 78, 95, 21, 68, 18, 69, 1, 90, 11, 59, 28, 24, 7
9, 54, 15, 40, 13, 61, 43, 42, 36, 57, 31, 48, 26, 64, 77, 41, 94, 83, 2
5, 67, 51, 14, 99, 70, 91, 6, 63, 9, 58, 76, 30, 60, 7, 72, 88, 32, 38, 4
6, 65, 16, 47, 27, 80, 96]
```

In [31]:
```
route_finder = RouteFinder(cities, cities_names, iterations=100)
best_distance, best_route = route_finder.solve()
print(best_distance)
print(best_route)
```

```
0 msec
172 msec
236 msec
306 msec
377 msec
511 msec
580 msec
653 msec
753 msec
813 msec
911 msec
975 msec
1136 msec
1308 msec
1443 msec
1510 msec
1612 msec
1680 msec
1776 msec
1937 msec
2034 msec
2098 msec
2191 msec
2255 msec
2351 msec
2480 msec
2549 msec
2620 msec
2690 msec
2754 msec
2854 msec
2919 msec
3047 msec
3180 msec
3245 msec
3315 msec
3383 msec
3482 msec
3578 msec
3643 msec
3712 msec
3817 msec
3950 msec
4054 msec
4154 msec
4222 msec
4294 msec
4362 msec
4460 msec
4564 msec
4662 msec
4755 msec
```

```
4820 msec
4888 msec
4960 msec
5092 msec
5156 msec
5243 msec
5347 msec
5453 msec
5587 msec
5657 msec
5723 msec
5825 msec
5890 msec
5990 msec
6056 msec
6222 msec
6326 msec
6456 msec
6525 msec
6616 msec
6683 msec
6808 msec
6936 msec
7032 msec
7136 msec
7297 msec
7467 msec
7535 msec
7637 msec
7762 msec
7862 msec
7965 msec
8035 msec
8138 msec
8267 msec
8366 msec
8437 msec
8505 msec
8641 msec
8769 msec
8895 msec
8992 msec
9089 msec
9157 msec
9320 msec
9418 msec
9517 msec
9585 msec
2972.29
[0, 14, 27, 49, 2, 30, 34, 41, 94, 85, 3, 37, 32, 69, 28, 38, 19, 72, 93,
52, 70, 56, 8, 90, 59, 96, 81, 47, 74, 87, 42, 98, 46, 25, 5, 15, 29, 11,
92, 7, 16, 99, 61, 48, 23, 33, 78, 66, 58, 64, 50, 53, 89, 79, 24, 1, 65,
10, 95, 21, 84, 44, 86, 9, 22, 26, 12, 4, 35, 62, 18, 54, 17, 55, 73, 20,
75, 76, 57, 13, 97, 31, 45, 91, 43, 51, 63, 71, 60, 36, 39, 82, 88, 68,
6, 40, 80, 67, 83, 77]
```

In [ ]:
```
1
```