CHIARA THIEN THAO NGUYEN BA 10727985

EXERCISE 4

- POINT A

The distance between each senson and the sink is calculated as: $d^2 = (x_s - x)^2 + (y_s - y)^2$ [m^2]

The energy consumption of each senson is calculated as :

$$E = b \cdot (E_c + E_{tx}(d)) = 2000 \cdot (50 + d^2)\ [nJ]$$

The lifetime of each senson is calculated as : $L = \frac{E_b}{E} \cdot 10$ [minutes]

In the following table I reported for each senson the distance d^2, the energy consumption E and the lifetime L.

SENSOR	DISTANCE SENSOR-SINK d^2	ENERGY E [nJ]	LIFETIME L [minutes]
1	685	1470000	34.01
2	389	878000	56.95
3	400	900000	55.55
4	194	488000	102.25
5	557	1214000	41.19
6	185	470000	106.38
7	292	684000	73.10
8	389	878000	56.95
9	338	776000	64.43
10	100	300000	166.67

The system lifetime is the lowest lifetime between sensons (the most energy consumption), i.e. the lifetime of senson 1, 34 minutes.
Also we can notice that senson 1 has the highest distance from the sink, thus it is clearly the most energy consumption since the formula of the energy consumption depends on the distance between the senson and the sink.

- POINT B

For this point I give two different approaches to resolve it: the first one is by hand through an optimal resolution problem that gives an approximative solution and the second one is made by an algorithm in Python that gives the precise coordinates.

- First case: by hand
  I calculated the average coordinates of the sink through the coordinates of the sensons and I obtained:

$$x_s = \frac{1+10+4+15+6+9+14+3+7+12}{10} \simeq 8$$

$$y_s = \frac{2+3+8+7+1+12+4+10+7+14}{10} \simeq 7$$

By considering $(x_s, y_s) = (8,7)$ I obtained the following distances $d^2 = (x_s - x)^2 + (y_s - y)^2$:

SENSOR	d^2
1	74
2	20
3	17
4	49
5	40
6	26
7	45
8	34
9	1
10	65

In this case I have to minimize d^2 of Senson 1 since we want to minimize the max. energy consumption (i.e. maximize the min. lifetime between the sensons).

Through an optimation problem we can find to select other possible coordinates close to the previous ones that could lead to a better solution.

$x_s = 8 \quad y_s = 7$
$\max d_i^2 = d_1^2 = 74$

$x_s = 8 \quad y_s = 6$
$\max d_i^2 = d_{10}^2 = 80 \neq d_1^2$ X

$x_s = 7 \quad y_s = 7$
$\max d_i^2 = d_{10}^2 = 74 \neq d_1^2$ X

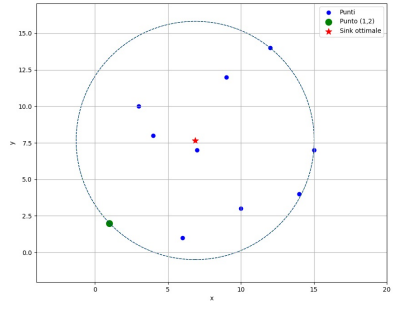
We can notice that by selecting other coordinates the worst case senson has changed and became the senson 10, thus these solutions are not feasible. In this case, the best solution is $x_s = 8$ and $y_s = 7$.

- Second case: Python optimization

The python script optimizes the position of the sink relative to the other coordinates, ensuring it is farther from the specific point (1,2), i.e. the worst case senson. The optimization minimizes the distance to the point (1,2) while it maintains this point as the fartest one from the sink.

The optimization gives as result: $x_s = 6.8717$ and $y_s = 7.6593$.

SENSOR	d^2
1	66.5042
2	31.4954
3	8.3626
4	66.5042
5	45.1060
6	23.3715
7	64.2034
8	20.4688
9	0.4511
10	66.5042



In this case the distance of senson 1 to the sink is $d^2 = 66.5042$.

The following Python script solves the optimization problem using a constrained minimization technique. The goal is to find the optimal coordinates of the sink that satisfy two conditions:

1. Minimize the distance from the sink to the point of interest with coordinates (1,2).
2. The distance from the sink to the point of interest should be greater than the maximum distance from the sink to any of the other points.

The code uses the SLSQP (Sequential Least Squares Programming) method to solve this constrained minimization problem.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize

# List of coordinates
coords = [(1, 2), (10, 3), (4, 8), (15, 7), (6, 1), (9, 12), (14, 4), (3, 10), (7, 7), (12, 14)]

# Point of interest
point_of_interest = (1, 2)

# Objective function to minimize
def objective(sink):
    x_s, y_s = sink

    # Calculate distance from the point of interest (1,2)
    dist_to_poi = (x_s - point_of_interest[0])**2 + (y_s - point_of_interest[1])**2

    # Calculate distances from all other points
    other_distances = []
    for x, y in coords:
        if (x, y) != point_of_interest:
            dist = (x_s - x)**2 + (y_s - y)**2
            other_distances.append(dist)

    # If the distance from the point of interest is smaller than the maximum distance from other points,
    # we add a high penalty
    if dist_to_poi <= max(other_distances):
        penalty = 1000 + (max(other_distances) - dist_to_poi)
    else:
        penalty = 0

    # Objective: minimize the distance from the point of interest + penalty
    return dist_to_poi + penalty

# Constraint function: we want the distance from (1,2) to be greater than all the other distances
def constraint(sink):
    x_s, y_s = sink

    # Distance from the point of interest (1,2)
    dist_to_poi = (x_s - point_of_interest[0])**2 + (y_s - point_of_interest[1])**2

    # Maximum distances from other points
    max_other_dist = max([(x_s - x)**2 + (y_s - y)**2 for x, y in coords if (x, y) != point_of_interest])

    # We want dist_to_poi > max_other_dist, so we return dist_to_poi - max_other_dist
    # The constraint is satisfied if this value is > 0
    return dist_to_poi - max_other_dist

# Define constraints for optimization
constraints = {'type': 'ineq', 'fun': constraint}
```

```
# Initial guess for optimization (far enough from all points)
initial_guess = [0.0, 0.0]

# Perform optimization with constraints
result = minimize(objective, initial_guess, method='SLSQP', constraints=constraints)

# Check if optimization was successful
if result.success:
    optimal_x_s, optimal_y_s = result.x
        print(f"Optimization successful! Optimal sink coordinates: x_s = {optimal_x_s:.4f}, y_s =
{optimal_y_s:.4f}")
else:
    print("Optimization failed.")
```

The objective function to be minimized is defined as:

$$f(x_s, y_s) = ((x_s - x_{poi})^2 + (y_s - y_{poi})^2) + penalty$$

where:

- (x_s, y_s) are the coordinates of the sink.
- $(x_{poi}, y_{poi}) = (1, 2)$ are the coordinates of the point of interest.
- $penalty$ is applied if the sink is too close to the point of interest, ensuring the second condition is met.

The penalty is calculated as: $penalty = 1000 + (max\ other\ distance - distance\ to\ point)$ if $distance\ to\ point \leq max\ other\ distance$, 0 otherwise.

The constraint enforces that the sink's distance to the point of interest is strictly greater than its maximum distance to any other point:

$$(x_s - x_{poi})^2 + (y_s - y_{poi})^2 - max((x_s - x)^2 + (y_s - y)^2) > 0 \quad \forall (x, y) \neq (x_{poi}, y_{poi})$$

The returned optimal solution is $(x_s, y_s) = (6.8717, 7.6593)$.

- POINT C

	Fixed Sink	Dynamic Sink
Pros	• Simpler to implement and manage • Easier to calculate the energy consumption of each sensor since the distance between each sensor and the sink is always constant	• Moving the sink will cause an even energy consumption between all sensors • It can adapt to new network updates or changes by moving the sink to the best position
Cons	• The sensor with the greatest distance from the sink has always the highest energy consumption, thus the lifetime will depend on it • It cannot adapt to new network conditions that could potentially worst the lifetime of the system	• Require complex manage for moving the sink with potential cost • Add energy consumption for moving the sink in a new position • Require specific implementation for dynamically calculating the energy consumption of the sensors each time the sink moves