Task1

Here's a simple algorithm for black box optimization under the given computation budget of 1,000 solution evaluations. This algorithm is based on a randomized search approach called Random Search.

Algorithm: Random Search for Black Box Optimization

Inputs:

- $f(x)$: Unknown objective function to be optimized.
- n : Dimensionality of the input vector (x).
- budget: Total number of solution evaluations allowed (1,000 in this case).

Outputs:

- x^* : Optimal solution found within the budget.
- y^* : Corresponding value of the objective function at x^*.

Procedure:

1. Initialize x^* to a random point in the search space.
2. Initialize (y^*) to $f(x^*)$.
3. Repeat until the budget is exhausted:
    - Generate a random candidate solution $x_{candidate}$ in the search space.
    - Evaluate the objective function: $y_{candidate} = f(x_{candidate})$.
    - If $y_{candidate}<y^*$:
        - Update ($x^* = x_{candidate}$).
        - Update ($y^* = y_{candidate}$).
4. Return (x^*) and (y^*).

Pseudocode:

```
function RandomSearch(f, n, budget):
    x* = RandomPointInSearchSpace(n)
    y* = f(x*)

    for i = 1 to budget:
        x_candidate = RandomPointInSearchSpace(n)
        y_candidate = f(x_candidate)

        if y_candidate < y*:
            x* = x_candidate
            y* = y_candidate

    return x*, y*
```

Explanation:

- **Initialization**: We start by randomly selecting an initial solution (x^*) and evaluating the objective function (y^*) at this point.
- **Random Search**: At each iteration, we generate a random candidate solution ($x_{\text{candidate}}$) and evaluate the objective function ($y_{\text{candidate}}$) at this point.
- **Update**: If the objective value of the candidate solution is better than the current best objective value (y^*), we update (x^*) and (y^*) accordingly.
- **Termination**: We repeat this process until the budget of solution evaluations is exhausted.
- **Output**: Finally, we return the best solution (x^*) and its corresponding objective value (y^*).

This algorithm is simple and easy to implement, making it suitable for optimization problems with limited computational resources. However, it may not be the most efficient method for complex or high-dimensional optimization problems.

Example:

Let's consider a simple example of a non-linear function in two dimensions:

$$f(x) = x_1^2 + x_2^2 - 2x_1x_2$$

Our objective is to minimize this function within a given search space.

Now, I'll run the random search algorithm to find the minimum value of this function.

```
import numpy as np

def objective_function(x):
    return x[0]**2 + x[1]**2 - 2 * x[0] * x[1]

def random_point_in_search_space(n):
    return np.random.uniform(-5, 5, n)  # Choose a random initial point within
[-5, 5]

def random_search(f, n, budget):
    x_best = random_point_in_search_space(n)
    y_best = f(x_best)

    for _ in range(budget):
        x_candidate = random_point_in_search_space(n)
        y_candidate = f(x_candidate)

        if y_candidate < y_best:
            x_best = x_candidate
            y_best = y_candidate

    return x_best, y_best

# Set parameters
```

```
dimension = 2
total_evaluations = 1000

# Run random search
best_solution, best_value = random_search(objective_function, dimension,
total_evaluations)

print("Best solution:", best_solution)
print("Best value:", best_value)
```

Result:

	Best solution	Best value
1	[1.01427189 1.01175841]	6.3175956319661e-06
2	[-0.53417303 -0.533702]	2.2187404014051282e-07
3	[-0.42680544 -0.41804073]	7.682014548421323e-05
4	[-0.71933676 -0.72223327]	8.389763064498013e-06
5	[-2.91101585 -2.91256956]	2.414001354367201e-06
6	[4.44779413 4.44698485]	6.549290247903627e-07

By the test, I found that, In the random search algorithm, the results can vary due to the randomness involved in selecting the initial random points and the stochastic nature of the algorithm itself. Random search is a heuristic algorithm that does not guarantee finding the global optimum but rather explores the search space through random sampling to find local optima.

In this particular run, our random search algorithm might have converged to a local optimum instead of the global optimum. Additionally, the range of the search space could affect the algorithm's performance. Increasing the range of the search space or the number of iterations of the algorithm might help in finding results closer to the global optimum.

Random Search VS Annealing algorithm

I used Annealing algorithm to solve the example.

	Best solution	Best value
1	[-0.4531938 -0.45222775]	0.0
2	[2.22111627 2.22168406]	0.0
3	[3.2469398 3.24832672]	0.0
4	[0.72767457 0.72736763]	0.0
5	[-4.33428329 -4.3344477]	0.0

	Best solution	Best value
6	[-2.55540414 -2.55682471]	0.0

Summary: Both algorithms managed to find solutions close to the global minimum. However, Simulated Annealing found a solution closer to the origin with a slightly higher objective function value compared to Random Search. This difference could be attributed to the probabilistic nature of Simulated Annealing, which allows it to explore the search space more efficiently, potentially finding solutions in regions where Random Search may not. Overall, both algorithms performed reasonably well for this optimization problem, but Simulated Annealing demonstrated a slight advantage in this comparison.