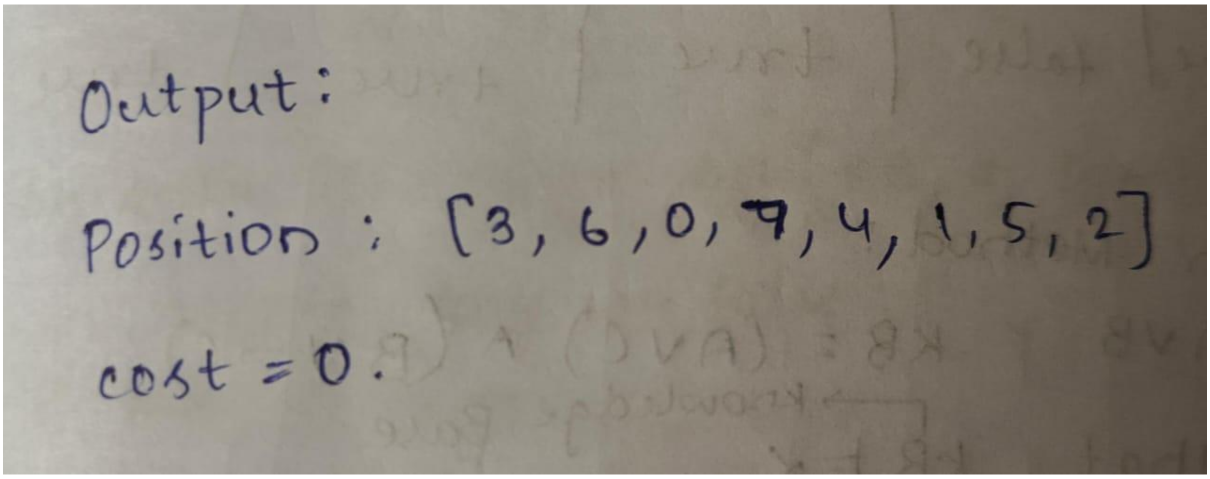
Week – 5

Implement Simulated Annealing:

Algorithm:

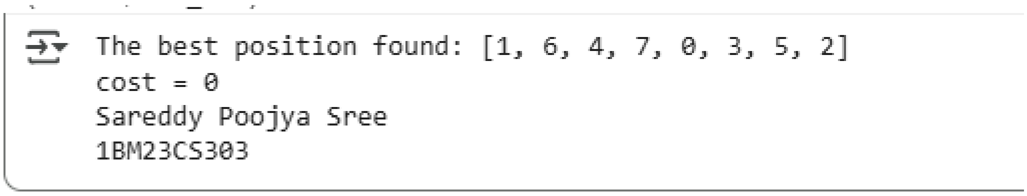


Output:

Position : [3, 6, 0, 7, 4, 1, 5, 2]

cost = 0.

Output:



```
The best position found: [1, 6, 4, 7, 0, 3, 5, 2]
cost = 0
Sareddy Poojya Sree
1BM23CS303
```

Code:

```
import random

import math# Heuristic: number of attacking pairs

def calculate_cost(state):

    cost = 0

    n = len(state)

    for i in range(n):

        for j in range(i + 1, n):

            if state[i] == state[j] or abs(state[i] - state[j]) == abs(i - j):

                cost += 1

    return cost


# Generate a random neighbor

def get_random_neighbor(state):

    n = len(state)
    new_state = list(state)
    col = random.randint(0, n - 1)   # pick random column
    row = random.randint(0, n - 1)   # new row
    new_state[col] = row
    return new_state

def simulated_annealing(n=8, max_iterations=10000, initial_temp=100.0, cooling_rate=0.99):
    # start with a random state
    current = [random.randint(0, n - 1) for _ in range(n)]
    current_cost = calculate_cost(current)
    best = current
    best_cost = current_cost
    temperature = initial_temp

    for _ in range(max_iterations):
        if current_cost == 0:
            break  # found solution

        neighbor = get_random_neighbor(current)
        neighbor_cost = calculate_cost(neighbor)
        delta = neighbor_cost - current_cost

        if delta < 0 or random.random() < math.exp(-delta / temperature):
            current, current_cost = neighbor, neighbor_cost

            if current_cost < best_cost:
                best, best_cost = current, current_cost

        temperature *= cooling_rate
        if temperature < 1e-6:
            break
    return best, best_cost


best_state, best_cost = simulated_annealing()


print("The best position found:", best_state)
print("cost =", best_cost)
```