Question 1

Zeal Liang

Z5325156

(a)

First, we subtract all the numbers inside array A from its own index to get an array B of differences, with $O(n)$ time complexity.

The second step is to count the number of times the same numbers inside the array B and store them in a hash table C, with a time complexity of $O(n)$.

The third step is to traverse the hash table C, find all occurrences greater than 1, and calculate all these numbers by substituting them into the combinatorial number formula (in this problem, n = number of occurrences greater than 1 and m = 2).

$$\binom{n}{m} = \frac{n!}{m! * (n-m)!}$$

Finally add all the results together to get the number of pairs of consistent indices. Since the maximum number of occurrences cannot be greater than the length of the array A and the time complexity of factorial is $O(n)$, the time complexity of this step is also $O(n)$.

The final total expected time complexity of the algorithm is $O(n)$.

Supplement Python code

```python
from math import *

def q1a_solution(a):
    number_showed = []
    showed_times = []

    for i in range(len(a)):
        if a[i] - i not in number_showed:
            number_showed.append(a[i] - i)
            showed_times.append(0)
        showed_times[number_showed.index(a[i] - i)] += 1

    result = 0
    for i in showed_times:
        if i > 1:
            result += factorial(i) // (2 * factorial(i-2))

    print(a)
    print("number_showed =>", number_showed)
    print("showed_times  =>", showed_times)
    print("Result =", result, "\n")

q1a_solution([0,1,2,10,6,4,13])
q1a_solution([0,1,2,3,4,5,6,7])

# Output
# [0, 1, 2, 10, 6, 4, 13]
# number_showed => [0, 7, 2, -1]
# showed_times  => [3, 2, 1, 1]
# Result = 4
#
# [0, 1, 2, 3, 4, 5, 6, 7]
# number_showed => [0]
# showed_times  => [8]
# Result = 28
```

(b)

We use AVLTree instead of hash table to solve this problem.

First, we subtract all the numbers inside array A from its own index to get the differences, and then insert these differences into an AVLTree as key, also use the value inside the key to store the number of occurrences, with $\boldsymbol{O(n\ log\ n)}$ time complexity.

The second step is to traverse the AVLTree, find all occurrences greater than 1, and calculate all these numbers by substituting them into the combinatorial number formula (in this problem, $\boldsymbol{n}$ = number of occurrences greater than 1 and $\boldsymbol{m}$ = 2).

$$\binom{\boldsymbol{n}}{\boldsymbol{m}} = \frac{\boldsymbol{n}!}{\boldsymbol{m}! * (\boldsymbol{n} - \boldsymbol{m})!}$$

Finally add all the results together to get the number of pairs of consistent indices. Since the maximum number of occurrences cannot be greater than the length of the array A and the time complexity of factorial is $\boldsymbol{O(n)}$, also the time complexity of each access to an element in AVLTree is $\boldsymbol{O(log\ n)}$, the time complexity of this step is $\boldsymbol{O(n\ log\ n)}$.

The final total worst case time complexity of the algorithm is $\boldsymbol{O(n\ log\ n)}$.

Supplement Python code

```python
from bintrees import *
from math import *

def q1b_solution(a):
    tree = AVLTree()
    for i in range(len(a)):
        key = a[i] - i
        if not tree.__contains__(key):
            tree.insert(key, 1)
        else:
            tree.__setitem__(key, tree.__getitem__(key) + 1)

    result = 0
    for i in tree.values():
        if i > 1:
            result += factorial(i) // (2 * factorial(i-2))

    print(tree)
    print(result)

q1b_solution([0,1,2,10,6,4,13])
q1b_solution([0,1,2,3,4,5,6,7])

# Output
# AVLTree({-1: 1, 0: 3, 2: 1, 7: 2})
# 4
#
# AVLTree({0: 8})
# 28
```