Task 6 – Fake Coin Detection Algorithm

1. Problem Description

In this task, we are given n coins (where n > 2). All the coins appear identical, but exactly one of them is counterfeit. The fake coin is either heavier or lighter than all genuine coins, and all real coins have the same weight.

We are also given a two-pan balance scale that can only return three outcomes:

•Left side heavier

•Right side heavier

•Both sides are equal

The goal is to design an O(1) algorithm to determine whether the fake coin is heavier or lighter

2. Solution Idea

An O(1) algorithm means the number of operations must remain constant, regardless of how many coins exist. If we attempted to use all n coins, we would need a loop to compare them, which would immediately result in O(n) time. Therefore, an O(1) solution cannot involve scanning all coins, and to keep the algorithm in constant time, I limited the decision process to three coins only.

Using exactly two weighings with three coins (A, B, C), we can determine:
1. Which coin is fake
2. Whether it is heavier or lighter.

3. Modeling the Balance Scale

A real scale gives only three outcomes:
• Left side heavier
• Right side heavier
• Both equal

To model this, weight() returns:
• 1 → left heavier
• 0 → equal
• -1 → right heavier

4. Algorithm Explanation

Step 1 – Compare A and B:
• If equal → C is fake

• If different → fake is A or B

Step 2 – Compare A and C:
Case 1: A and B equal → compare C with A:
• C heavier → fake heavier
• C lighter → fake lighter

Case 2: A and B different → compare A with C:
• A heavier than both → A is fake (heavier)
• A lighter than both → A is fake (lighter)
• Otherwise → B is fake

5. Code Implementation (O(1))

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

int weight(int left, int right) {
   int random = rand() % 3;
   if (random == 0) return 0;
   if (random == 1) return 1;
   return -1;
}

string faketype(int A, int B, int C) {
   int w1 = weight(A, B);
   if (w1 == 0) {
      int w2 = weight(C, A);
      if (w2 > 0) return "Fake coin is C and heavier";
      else return "Fake coin is C and lighter";
   }
   else {
      int w2 = weight(A, C);
      if (w1 == 1) {
         if (w2 == 1) return "Fake coin is A and heavier";
         else return "Fake coin is B and lighter";
      }
      else {
         if (w2 == -1) return "Fake coin is A and lighter";
         else return "Fake coin is B and heavier";
      }
```

```
    }
}
```

Code O(n):

```
#include <iostream>
#include <vector>
#include <cstdlib>
#include <ctime>
#include <string>
using namespace std;

int weight(int left, int right) {
    int r = rand() % 3;
    if (r == 0) return 0;
    if (r == 1) return 1;
    return -1;
}

string findFakeCoin(vector<int> coins) {
    int index_reference = coins[0];
    for (int i = 1; i < coins.size(); i++) {
        int w = weight(coins[i], index_reference);
        if (w != 0) {
            if (w == 1) return "Fake coin " + to_string(i) + " heavier";
            else return "Fake coin " + to_string(i) + " lighter";
        }
    }
    return "All coins same";
}
```

6. Complexity Analysis

weight():

- Executes a fixed number of instructions.
- No loops.
- Always constant time.
    - Complexity: O(1)

faketype():

- Two weight() calls
- Uses only simple if-statements
- No loops, no recursion.
- Complexity: O(1)

Overall Algorithm Complexity:

Total Complexity: O (1)

O(n) Algorithm (findFakeCoin) :

The linear version checks each coin one by one:

- The loop runs as many times as the number of coins.
- Each coin is inspected exactly once.
- More coins = more time.
- Complexity: O(n) — runtime increases linearly with the number of coins.

7. Challenges Faced

Using more than three coins immediately forces a loop, making the solution O(n). The main challenge was understanding that true O(1) requires a fixed, constant number of operations.

8. Conclusion

The algorithm identifies the fake coin and determines whether it is heavier or lighter using only two comparisons. Limiting the logic to three coins ensures constant time O(1). Any solution involving more coins becomes O(n).