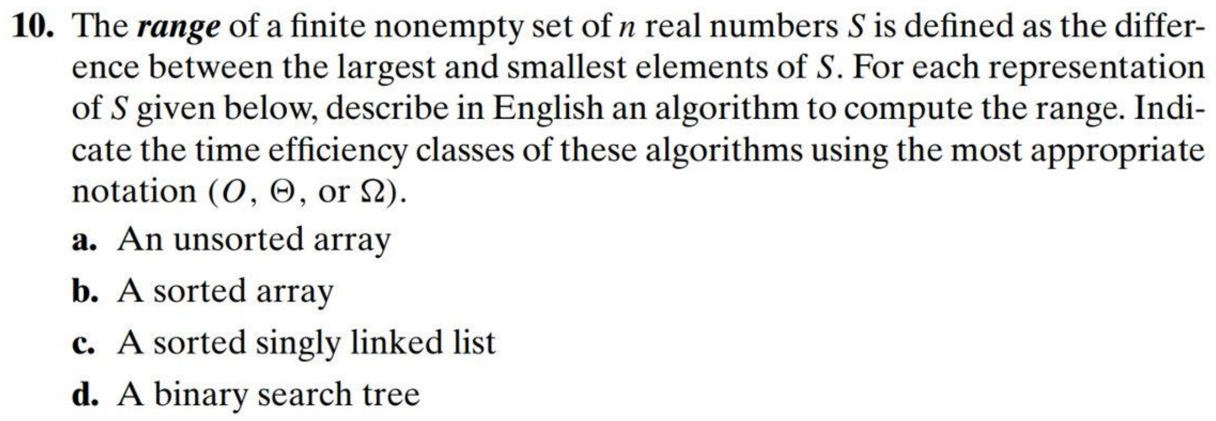
ANALYSIS OF NON-RECURSIVE ALGORITHMS

GROUP 7

Nguyễn Quốc Trường 21521604

Trần Xuân Minh 21520352



10. The ***range*** of a finite nonempty set of n real numbers S is defined as the difference between the largest and smallest elements of S. For each representation of S given below, describe in English an algorithm to compute the range. Indicate the time efficiency classes of these algorithms using the most appropriate notation (O, Θ, or Ω).

a. An unsorted array

b. A sorted array

c. A sorted singly linked list

d. A binary search tree

Answer:

The task is to find an algorithm for computing the range of a finite nonempty set of n real numbers S, which is defined as the difference between the largest and smallest elements of S. The time complexity of each algorithm is also to be determined using the appropriate notation (O, theta, or omega).

a. An unsorted array:

Traverse through each element of the array to find the smallest and largest elements.

Compute the range by taking the difference between the largest and smallest elements.

The time complexity of this algorithm is O(n), as we have to traverse through all the elements of S.

b. A sorted array:

Take the first and last elements of the array to find the smallest and largest elements.

Compute the range by taking the difference between the largest and smallest elements.

The time complexity of this algorithm is O(1), as we can directly access the first and last elements of the array.

c. A sorted singly linked list:

Traverse through each node of the linked list to find the smallest and largest elements.

Compute the range by taking the difference between the largest and smallest elements.

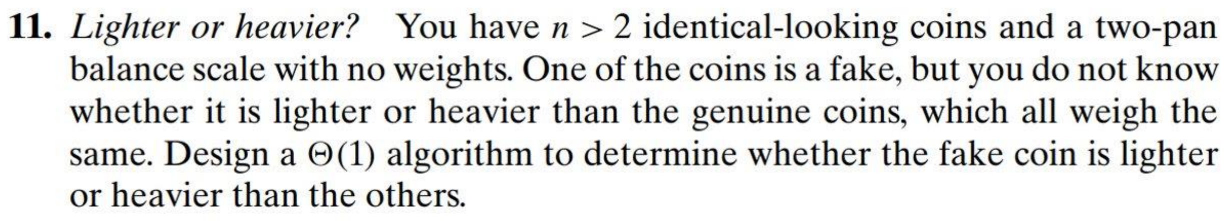
The time complexity of this algorithm is O(n), as we have to traverse through all the nodes of the linked list.

d. A binary search tree:

Traverse to the leftmost leaf node of the tree to find the smallest element and traverse to the rightmost leaf node of the tree to find the largest element.

Compute the range by taking the difference between the largest and smallest elements.

The time complexity of this algorithm is O(log n), as we only need to perform queries on the nodes of the binary search tree.



11. *Lighter or heavier?* You have $n > 2$ identical-looking coins and a two-pan balance scale with no weights. One of the coins is a fake, but you do not know whether it is lighter or heavier than the genuine coins, which all weigh the same. Design a $\Theta(1)$ algorithm to determine whether the fake coin is lighter or heavier than the others.

Answer:

It is not possible to design a Θ(1) algorithm to determine whether the fake coin is lighter or heavier than the others in the given scenario. This is because regardless of the number of coins, we need to use the balance scale to compare the weights of two sets of coins. The act of weighing the coins on the balance scale will always take some amount of time and cannot be done in constant time.

However, we can design an algorithm that has a worst-case time complexity of Θ(1), but requires some preprocessing. The algorithm is as follows:

Divide the n coins into three groups of n/3 coins each, and weigh two of these groups against each other using the balance scale.

There are three possible outcomes:

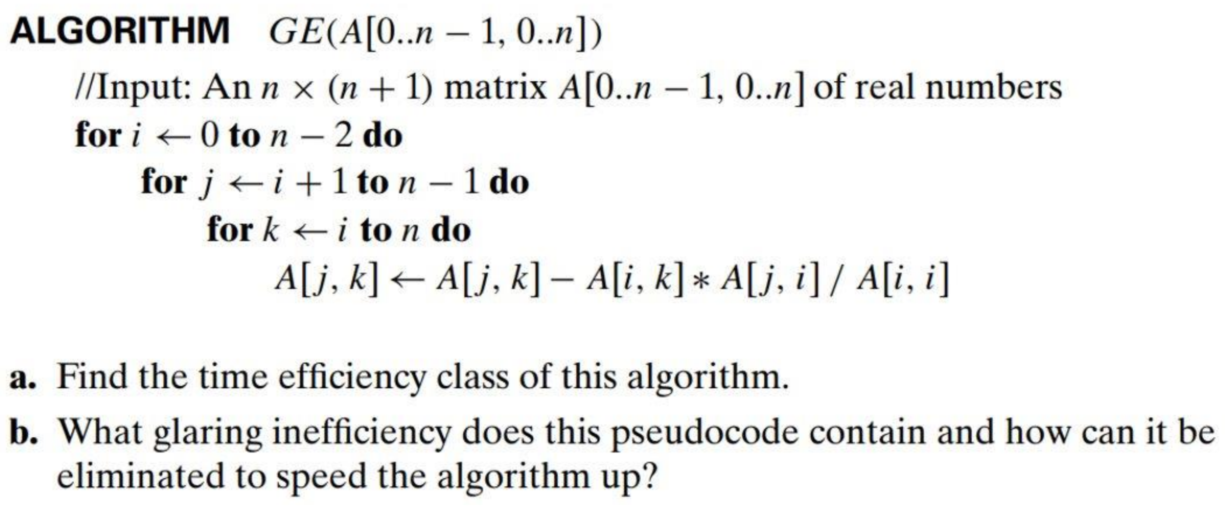
a. If the two groups weigh the same, then the fake coin is in the remaining group of n/3 coins.

b. If one group is heavier than the other, then the fake coin is in that group and we know whether it is heavier or lighter than the genuine coins.

c. If one group is lighter than the other, then the fake coin is in that group and we know whether it is lighter or heavier than the genuine coins.

We can use the information obtained from the first weighing to determine which set of coins contains the fake coin and whether it is lighter or heavier than the genuine coins. We can then repeat the process with that set of coins until we have identified the fake coin.

The worst-case time complexity of this algorithm is Θ(1), since we only need to weigh two sets of coins on the balance scale, regardless of the number of coins. However, the preprocessing step of dividing the coins into groups takes O(n) time, which is not constant. Therefore, this algorithm can be considered Θ(1) only in terms of the number of weighings required to identify the fake coin.



```
ALGORITHM  GE(A[0..n − 1, 0..n])
    //Input: An n × (n + 1) matrix A[0..n − 1, 0..n] of real numbers
    for i ← 0 to n − 2 do
        for j ← i + 1 to n − 1 do
            for k ← i to n do
                A[j, k] ← A[j, k] − A[i, k] ∗ A[j, i] / A[i, i]
```

a. Find the time efficiency class of this algorithm.

b. What glaring inefficiency does this pseudocode contain and how can it be eliminated to speed the algorithm up?

Answer:

a. To determine the time complexity of the GE algorithm, we need to count the number of operations performed. In each iteration of the outer loop, we perform n iterations of the middle loop, and in each iteration of the middle loop, we perform n-i iterations of the inner loop. Within the inner loop, we perform a constant number of operations. Therefore, the total number of operations is:

(n-1) * (n^2 - n) = n^3 - n^2 - n^2 + n = O(n^3)

Thus, the time complexity of the GE algorithm is O(n^3).

b. One inefficiency in the pseudocode is that it performs unnecessary divisions by A[i, i] when j=i, since the value of A[i, i] is always 1. This can be avoided by checking whether j=i before performing the division.

Additionally, the algorithm performs unnecessary subtractions and multiplications when A[j, i] is zero. This can be avoided by checking whether A[j, i] is zero before performing the operations.

By making these two modifications, the algorithm can be made more efficient and run faster. The modified pseudocode for the GE algorithm is as follows:

```
ALGORITHM IMPROVED-GE(A[0..n − 1, 0..n])
    //Input: An n × (n + 1) matrix A[0..n − 1, 0..n] of real numbers
    for i ← 0 to n − 2 do
        for j ← i + 1 to n − 1 do
            if A[j, i] ≠ 0 then
                ratio ← A[j, i] / A[i, i]
            for k ← i to n do
                A[j, k] ← A[j, k] − ratio * A[i, k]
```