[Click on the logo to find **Problem Statement**]

Approach 1: Brute Force

Intuition

This code takes a list of integers as an input and returns a boolean value indicating whether the list contains any duplicate elements or not. The function uses a nested loop to compare each element of the list with every other element. If it finds two elements that are equal, it returns True, meaning that the list has duplicates. If it does not find any equal elements after checking all the possible pairs, it returns False, meaning that the list has no duplicates.

Approach

1. Initialize a variable n to store the length of the list.
    - This will help us iterate over the list elements using indices.
2. Use a nested loop to compare each element with every other element in the list.
    - The outer loop will go from 0 to n-1, where n is the length of the list.
    - The inner loop will go from i+1 to n, where i is the current index of the outer loop.
    - For each pair of elements, check if they are equal. If yes, return True, meaning that the list has duplicates.
3. If the loop ends without finding any duplicates, return False, meaning that the list has no duplicates.

Complexity

- Time complexity: O(n^2)
- Space complexity: O(1)

Code

```python
class Solution:
    def containsDuplicate(self, nums: List[int]) -> bool:
        n = len(nums)
        for i in range(n - 1):
            for j in range(i + 1, n):
                if nums[i] == nums[j]:
                    return True
        return False
```

Approach 2: Sorting

Intuition

This code takes a list of integers as an input and returns a boolean value indicating whether the list contains any duplicate elements or not. The function uses a sorting algorithm to arrange the elements of the list in ascending order, and then compares each element with its next element. If it finds two elements that are equal, it returns True, meaning that the list has duplicates. If it does not find any equal elements after checking all the pairs, it returns False, meaning that the list has no duplicates.

Approach

1. Sort the list in ascending order.
    - Use a sorting algorithm, such as merge sort or quick sort, to arrange the elements of the list from smallest to largest.
    - This will make it easier to compare adjacent elements in the next step.
2. Compare each element with its next element.
    - Use a loop to iterate over the sorted list from the first element to the second-last element.
    - For each element, check if it is equal to the next element in the list. If yes, return True, meaning that the list has duplicates.
3. If the loop ends without finding any duplicates, return False, meaning that the list has no duplicates.

Complexity

- Time complexity: O(n logn)

- Space complexity: O(n)

Code

```python
class Solution:
    def containsDuplicate(self, nums: List[int]) -> bool:
            sort_list = sorted(nums)
            print(sort_list)
            for i in range(len(sort_list)-1):
                if sort_list[i] == sort_list[i+1]:
                    return True
            return False
```

Approach 3: Hashset

Intuition

It employs a hash set to efficiently keep track of unique elements encountered during the iteration through the input list. The algorithm initializes an empty hash set, denoted as hash_set, to store unique elements. It then iterates through each element in the input list, checking if the current element is already present in the hash set. If the element is found in the hash set, the algorithm concludes that a duplicate exists and returns True. Otherwise, it adds the element to the hash set to maintain a record of encountered elements. If the iteration completes without finding any duplicates, the function returns False.

Approach

1. Create an empty set.
    - This will hold the elements of the list as we iterate over them.
2. Loop through the list elements.
    - For each element, check if it is already in the set. If yes, return True, meaning that the list has duplicates. If not, add the element to the set and continue the loop.
3. If the loop ends without finding any duplicates, return False, meaning that the list has no duplicates.

Complexity

- Time complexity: O(n)
- Space complexity: O(n)

Code

```python
class Solution:
    def containsDuplicate(self, nums: List[int]) -> bool:
        hash_set = set()
        for i in nums:
            if i in hash_set:
                return True
            hash_set.add(i)
        return False
```

If you want to see more solutions to coding problems, you can visit:

GitHub