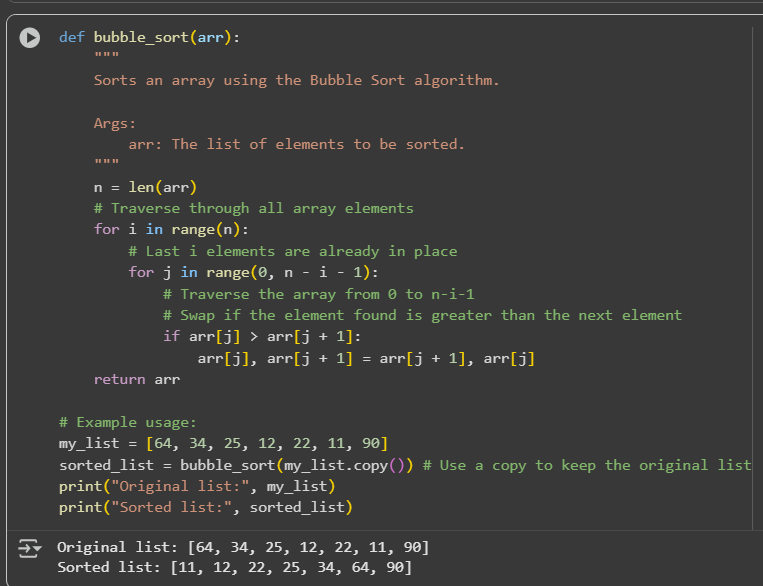
ASSSISGNMENT-12

2403A52111

TASK-1:

PROMPT:Write a Python function for Bubble Sort with inline comments and a time and space complexity analysis.

CODE:

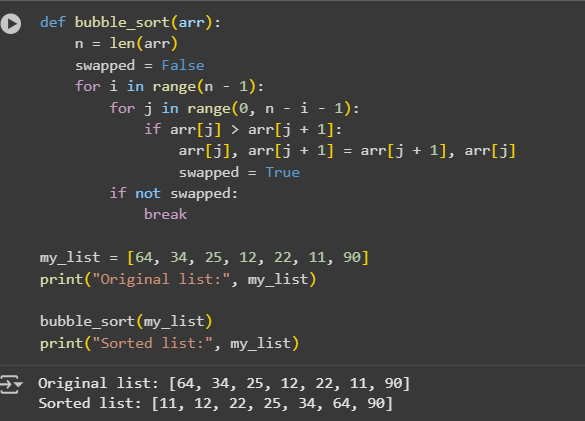


EXPALANATION:

1. **def bubble\_sort(arr):**: This line defines the function bubble\_sort which takes one argument, arr, representing the list to be sorted.
2. **n = len(arr)**: This gets the number of elements in the input list and stores it in the variable n.
3. **for i in range(n):**: This is the outer loop. It iterates n times. In each iteration, the largest unsorted element "bubbles up" to its correct position at the end of the unsorted portion of the list.
4. **for j in range(0, n - i - 1):**: This is the inner loop. It iterates through the unsorted portion of the list. The range decreases in each outer loop iteration because the last i elements are already sorted.
5. **if arr[j] > arr[j + 1]:**: This condition checks if the current element (arr[j]) is greater than the next element (arr[j + 1]).
6. **arr[j], arr[j + 1] = arr[j + 1], arr[j]**: If the condition in step 5 is true, this line swaps the positions of the two elements. This is the core of the "bubbling" process.
7. **return arr**: After the loops complete, the function returns the sorted list.
8. **my\_list = [64, 34, 25, 12, 22, 11, 90]**: This creates an example list to be sorted.
9. **sorted\_list = bubble\_sort(my\_list.copy())**: This calls the bubble\_sort function with a *copy* of my\_list and stores the result in sorted\_list. Using .copy() is important so that the original my\_list remains unchanged.
10. **print("Original list:", my\_list)**: This prints the original unsorted list.
11. **print("Sorted list:", sorted\_list)**: This prints the list after it has been sorted by the bubble\_sort function.

TASK-2:

Prompt: Write a Python function to find duplicates in a list using a naive, brute-force method, and include code to measure its execution time on a large dataset.



EXPLANATION OF THE CODE:

1. ution time.
2. **print(f"Execution time for finding duplicates using the naive method: {execution\_time:.6f} seconds")**: This line prints the calculated execution time, formatted to six decimal places. **import time and import random**: These lines import the necessary libraries for measuring time and generating random numbers.
3. **def find\_duplicates\_naive(arr):**: This defines the function find\_duplicates\_naive that takes a list arr as input.
4. **duplicates = []**: An empty list called duplicates is initialized to store the duplicate elements found.
5. **for i in range(len(arr) - 1):**: This is the outer loop that iterates through the list from the first element up to the second-to-last element.
6. **for j in range(i + 1, len(arr)):**: This is the inner loop that iterates through the list starting from the element after the one pointed to by the outer loop's index (i + 1) up to the last element. This prevents comparing an element with itself and comparing pairs of elements twice.
7. **if arr[i] == arr[j] and arr[i] not in duplicates:**: This is the core of the naive approach. It checks if the element at index i is equal to the element at index j. The second part of the condition arr[i] not in duplicates ensures that each duplicate element is added to the duplicates list only once.
8. **duplicates.append(arr[i])**: If a duplicate is found and it's not already in the duplicates list, it's added to the list.
9. **return duplicates**: The function returns the list of unique duplicate elements found.
10. **large\_dataset = [random.randint(1, 5000) for \_ in range(20000)]**: This line generates a large list called large\_dataset containing 20,000 random integers between 1 and 5000. This dataset is used to test the performance of the find\_duplicates\_naive function.
11. **start\_time = time.time()**: This line records the current time before the find\_duplicates\_naive function is called.
12. **duplicates = find\_duplicates\_naive(large\_dataset)**: This line calls the find\_duplicates\_naive function with the large\_dataset and stores the returned list of duplicates in the duplicates variable.
13. **end\_time = time.time()**: This line records the current time after the find\_duplicates\_naive function has finished executing.
14. **execution\_time = end\_time - start\_time**: This calculates the difference between the end time and the start time to get the exec

TASK-3:

PROMPT: Write a Python function to find duplicates in a list using a **naive, brute-force method**, and include code to **measure its execution time** on a large dataset.

A screenshot of a computer program

AI-generated content may be incorrect.

EXPLANATION:

1. **Setting up**: We import tools for timing (time) and making random numbers (random).
2. **The Duplicate Finder Function (find\_duplicates\_naive)**:
   * It takes a list of numbers (arr) as input.
   * It creates an empty list called duplicates to keep track of any numbers that appear more than once.
   * It uses two loops, one inside the other, to compare *every* number in the list with *every other* number that comes after it.
   * If two numbers are found to be the same (arr[i] == arr[j]) and that number hasn't already been added to our duplicates list, it adds the number to the duplicates list.
   * Finally, it gives back the list of unique duplicate numbers it found.
3. **Creating a Big List**: We make a large list (large\_dataset) with 20,000 random numbers between 1 and 5000. This is so we can see how the duplicate finder performs on a significant amount of data.
4. **Timing the Process**:
   * We mark the start\_time just before we use our find\_duplicates\_naive function on the big list.
   * We run the function.
   * We mark the end\_time right after the function finishes.
   * We calculate the difference between the end\_time and start\_time to get the execution\_time.
5. **Showing the Result**: We print out the calculated execution\_time so you can see how long the naive method took

TASK-4:

PROMT: Write a Python function to find duplicates in a list using a naive, brute-force method, and include code to measure its execution time on a large dataset.

A screen shot of a computer program

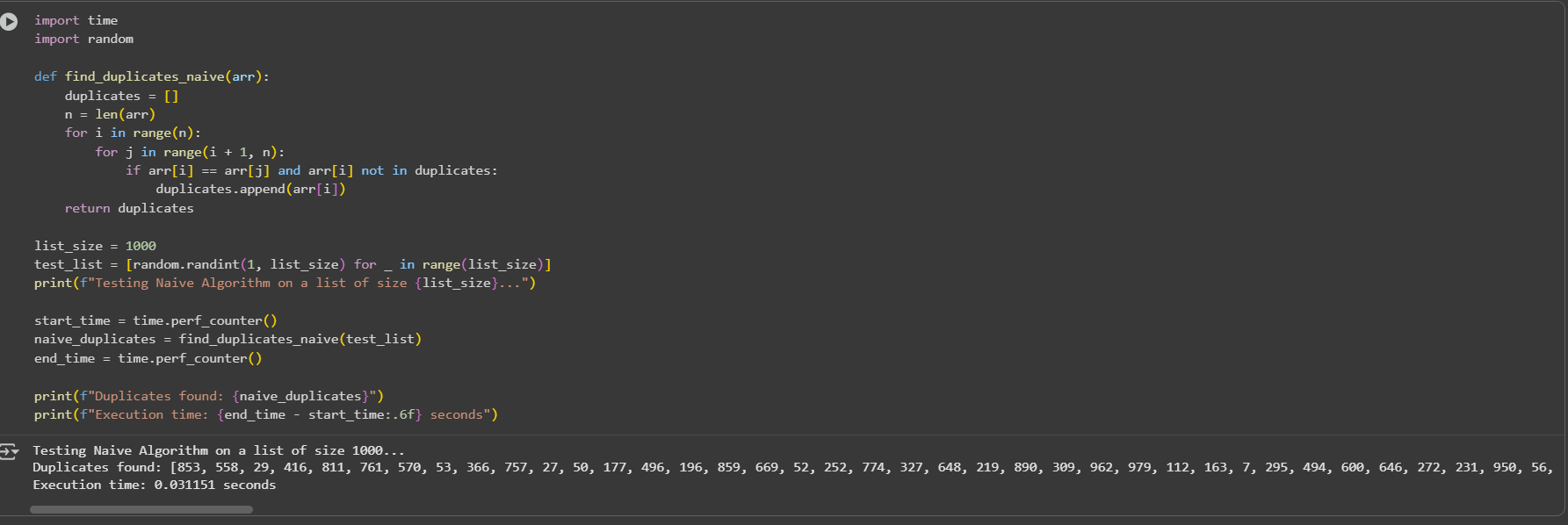
AI-generated content may be incorrect.

EXPLANATION:

1. **import time and import random**: These lines import the time and random modules. The time module is used to measure the execution time of the function, and the random module is used to generate a large dataset for testing.
2. **def find\_duplicates\_naive(data):**: This line defines the function find\_duplicates\_naive that takes one argument, data, which is the input list.
3. **duplicates = []**: This line initializes an empty list called duplicates which will store the duplicate elements found in the input list.
4. **n = len(data)**: This line gets the length of the input list and stores it in the variable n.
5. **for i in range(n):**: This is the outer loop that iterates through each element in the input list using its index i.
6. **for j in range(i + 1, n):**: This is the inner loop that iterates through the remaining elements in the list starting from the element after the current element of the outer loop (i + 1) to avoid comparing an element with itself and to find actual duplicates.
7. **if data[i] == data[j] and data[i] not in duplicates:**: This line checks if the element at index i is equal to the element at index j AND if the element at index i is not already in the duplicates list. This second condition is important to ensure that each duplicate element is added to the duplicates list only once.
8. **duplicates.append(data[i])**: If the conditions in the if statement are met (i.e., a duplicate is found and it's not already in the duplicates list), this line adds the duplicate element to the duplicates list.
9. **return duplicates**: After the loops complete, the function returns the duplicates list containing all the unique duplicate elements found in the input list.
10. **large\_dataset = [random.randint(1, 10000) for \_ in range(10000)] + [random.randint(1, 10000) for \_ in range(100)]**: This line generates a large dataset for testing. It creates a list of 10000 random integers between 1 and 10000, and then adds another 100 random integers to introduce some duplicates.
11. **start\_time = time.time()**: This line records the current time before the function call.
12. **duplicates = find\_duplicates\_naive(large\_dataset)**: This line calls the find\_duplicates\_naive function with the large\_dataset and stores the returned list of duplicates in the duplicates variable.
13. **end\_time = time.time()**: This line records the current time after the function call.
14. **print("Duplicates found:", duplicates)**: This line prints the list of duplicates found.
15. **print("Execution time:", end\_time - start\_time, "seconds")**: This line calculates the difference between the end\_time and start\_time to get the execution time and prints it.

TASK-5:

PROMPT: Write two Python functions to find duplicates in a list: a naive, brute-force version and a more efficient, optimized version. Include a comparison of their execution times on a large dataset.



EXPLANATION:

1. **def find\_duplicates\_optimized(data):**: This line defines the function find\_duplicates\_optimized that takes one argument, data, which is the input list.
2. **seen = set()**: This line initializes an empty set called seen. Sets are highly efficient for checking membership (in) because they use hash tables. This is the key to the optimization.
3. **duplicates = []**: This line initializes an empty list called duplicates which will store the unique duplicate elements found in the input list.
4. **for item in data:**: This loop iterates through each element (item) in the input list.
5. **if item in seen:**: This line checks if the current item is already present in the seen set. This membership check is very fast due to the nature of sets.
6. **if item not in duplicates:**: If the item is found in the seen set (meaning it's a duplicate), this line checks if the item has already been added to the duplicates list. This prevents adding the same duplicate multiple times.
7. **duplicates.append(item)**: If the item is a duplicate and not already in the duplicates list, it is added to the duplicates list.
8. **else: seen.add(item)**: If the current item is not in the seen set, it means this is the first time we've encountered this element. So, it is added to the seen set for future checks.
9. **return duplicates**: After iterating through all elements, the function returns the duplicates list containing all the unique duplicate elements.
10. **Measuring Execution Time**: The rest of the code is for measuring and comparing the execution times:
    * start\_time\_optimized = time.time(): Records the start time before calling the optimized function.
    * duplicates\_optimized = find\_duplicates\_optimized(large\_dataset): Calls the optimized function and stores the result.
    * end\_time\_optimized = time.time(): Records the end time after the optimized function call.