Quiz 18–23: Time Complexity and Sets

Name: ____________________ Date: ____________________

Instructions

- Answer in the blanks.
- For “write code” questions, write **valid Python code** (**no functions needed**).
- For “what does it print” questions, write the **exact** output (line by line).
- If a question says “order may vary”, any correct order is acceptable.
- You may assume all inputs are valid (unless the question says otherwise).

Part A — Time Complexity (O(1), O(n), O(n²))

1) Quick facts (fill in the blanks)

Fill in the blanks using **O(1)**, **O(n)**, **O(n²)**.

1. A loop that checks every item in a list once is usually ____________________.
2. Two nested loops over the same list (both go from `0` to `n-1`) are usually ____________________.
3. Doing a constant amount of work (no loop over the input) is usually ____________________.

2) Worst-case search (time complexity)

```
numbers = [4, 7, 2, 9, 5]
target = 9

found = False
for x in numbers:
    if x == target:
        found = True
        break
print(found)
```

Worst-case time complexity: ______________________

---

3) Two passes (time complexity)

```
numbers = [3, 1, 4, 1, 5]

total = 0
for x in numbers:
    total = total + x

count_even = 0
for x in numbers:
    if x % 2 == 0:
        count_even = count_even + 1

print(total)
print(count_even)
```

Worst-case time complexity: ______________________

---

4) Set membership vs list membership (time complexity)

Assume `n = len(numbers)`.

```
numbers = [10, 20, 30, 40, 50]
s = set(numbers)

print(40 in numbers)   # list membership
print(40 in s)         # set membership
```

1. Worst-case time complexity of `40 in numbers` : ______________________
2. Average time complexity of `40 in s` : ______________________

5) Nested loops with a condition (time complexity)

Let `n = len(nums)` .

```
nums = [2, 5, 1, 4]
count = 0

for x in nums:
    for y in nums:
        if x < y:
            count = count + 1

print(count)
```

Time complexity: ______________________

6) “Build then check” (time complexity)

Assume `n = len(items)` . Assume **average** set membership is **O(1)**.

```
items = [3, 1, 4, 1, 5]

s = set(items)

# Many membership checks:
print(2 in s)
print(3 in s)
print(99 in s)
```

Overall time complexity (whole program): ______________________

7) Which approach is faster? (explain)

Assume `n = len(allowed) = len(queries)` .

```
allowed = ["red", "green", "blue", "yellow"]   # length = n
queries = ["blue", "pink", "red", "green"]     # length = n
```

Approach A

```
count = 0
for q in queries:
    if q in allowed:   # list membership
        count = count + 1
print(count)
```

Approach B

```
allowed_set = set(allowed)
count = 0
for q in queries:
    if q in allowed_set:   # set membership
        count = count + 1
print(count)
```

1. Worst-case time complexity of Approach A: ____________________
2. Worst-case time complexity of Approach B: ____________________
3. In 1–2 sentences, explain *why* Approach B is usually faster:

   ____

   ____

8) Choose the better Big-O (circle one)

Let `n = len(numbers)`.

```
numbers = [1, 2, 3, 4, 5]
pairs = 0

for x in numbers:
    if x in numbers:
        pairs = pairs + 1

print(pairs)
```

Note: even though `x` comes from `numbers`, the check `x in numbers` still scans the

Note: even though `n` comes from `numbers`, the check `n in numbers` still scans the list.

Time complexity (choose one):

- **O(1)**
- **O(n)**
- **O(n^2)**

9) Improve the complexity (short answer)

You need to do many membership checks on the **same** list `numbers`.

What is one simple change you can make so membership checks are faster on average?

Answer:

____

Part B — Set Basics + Basic Operations

10) Sets remove duplicates (what does it print?)

```
items = [1, 1, 2, 2, 2, 3]
s = set(items)

print(len(items))
print(len(s))
print(2 in s)
```

Output:

```

```

11) `{}` is not a set (fix the bug)

This code has a bug: it does **not** create a set.

This code has a bug: it does **not** create a set.

Fix it so `seen` is an empty set.

```
seen = {}
print(type(seen))
```

Write the fixed code:

```

```

12) Add, remove, and membership (fill in the blanks)

Fill in the blanks so the code prints `True` and `True`.

Choose from: `in` and `not in`.

```
s = set()
s.add("cat")
print("cat" ______ s)

s.discard("cat")
print("cat" ______ s)
```

Expected output:

```
True
True
```

13) Union + intersection (what does it print?)

(order may vary)

```
A = {1, 2, 3, 4}
B = {3, 4, 5}

print(A | B)
print(A & B)
```

Output:

```

```

14) Unique letters (write code)

Given:

```
text = "mississippi"
```

Write code to create a set named `letters` that contains all **unique** letters in `text`, then print `letters`.

```
text = "mississippi"

letters = _____________  # create an empty set

for ch in text:  # in each iteration, ch is a character
    letters.___________

print(letters)
```

Output (order may vary):

```

```

15) Filter with a set (write code)

You are given:

```
words = ["ant", "cat", "a", "dog", "to", "bee"]
```

Write code to build a set named `long_words` that contains only the words with length **>= 3**. Then print `long_words`.

Then print `long_words`.

```
words = ["ant", "cat", "a", "dog", "to", "bee"]

long_words = set()

for w in words:
    # your code here







print(long_words)
```

Output (order may vary):

```

```

Part C — Remove Duplicates + Union + Intersection

16) Stable remove duplicates (write code)

We want to remove duplicates but keep the **first** time each value appears.

Given:

```
nums = [2, 5, 2, 3, 5, 1, 3]
```

Write code to build `unique` as:

```
[2, 5, 3, 1]
```

Hint: use a set named `seen`.

```
nums = [2, 5, 2, 3, 5, 1, 3]

seen = set()
unique = []

for x in nums:
    if ______________________:
         _____________________
         _____________________

print(unique)
```

17) Find duplicates (write code)

Given:

```
nums = [1, 2, 1, 3, 2, 2, 4]
```

Write code to build a set `dups` that contains numbers that appear **at least twice**. Then print `dups`.

```
nums = [1, 2, 1, 3, 2, 2, 4]

seen = _____________________
dups = _____________________

for x in nums:
    if x in seen:
         ____________________
    else:
         ____________________

print(dups)
```

Output (order may vary):

```

```

18) Common items from two lists (write code)

You are given:

```
A = ["Amy", "Ben", "Chloe", "Dylan"]
B = ["Ben", "Eva", "Dylan", "Frank"]
```

Write code to build a set `both` that contains the names in **both** lists. Then print `both`.

```
A = ["Amy", "Ben", "Chloe", "Dylan"]
B = ["Ben", "Eva", "Dylan", "Frank"]

# your code here






print(both)
```

Output (order may vary):

```

```

19) Either team (write code)

You are given two lists:

```
team1 = ["A", "B", "C", "C"]
team2 = ["B", "D", "E"]
```

Write code to build a set `either` that contains everyone who is in **team1 or team2**. Then print `either`.

```
team1 = ["A", "B", "C", "C"]
team2 = ["B", "D", "E"]

# your code here








print(either)
```

Output (order may vary):

```

```

20) Two sentences (union + intersection) (write code)

Given two sets:

```
math = {"Amy", "Ben", "Chloe"}
swim = {"Ben", "Dylan", "Eva"}
```

Write code that prints:

- 1) how many students are in **either** club, and
- 2) how many students are in **both** clubs.

```
math = {"Amy", "Ben", "Chloe"}
swim = {"Ben", "Dylan", "Eva"}

# your code here









print(count_either)
print(count_both)
```

Output:

```

```

21) One-pass “two groups” (write code)

You are given:

```
nums = [0, 1, 2, 0, 3, 2, 4, 0]
```

Write code that builds:

- a set `nonzero` containing all **unique nonzero** numbers
- a variable `zero_count` counting how many zeros appear

Then print `nonzero` and `zero_count`.

```
nums = [0, 1, 2, 0, 3, 2, 4, 0]

nonzero = _________________  # create an empty set
zero_count = 0

for x in nums:
    if _________________:
        zero_count = zero_count + 1
    else:
        nonzero.___________________

print(nonzero)
print(zero_count)
```

Expected output (order may vary for the set):

```
{1, 2, 3, 4}
3
```

22) Two-sum idea with a set (write code + time complexity)

You are given a list and a target. Assume the list items are unique.

```
nums = [4, 7, 1, 9, 5]
target = 10
```

Write code that prints `True` if there exist **two different numbers** in `nums` that add to `target`, otherwise prints `False`.

Hint: Use a set named `seen` and check if `target - x` is already in `seen`.

```
nums = [4, 7, 1, 9, 5]
target = 10

seen = set()
found = False

for x in nums:
    if ______________ in seen:
        found = ______________
        break
    seen.______________________

print(found)
```

Expected output:

```
True
```

Let `n = len(nums)`.

Worst-case time complexity: ____________________