Quiz 18-25: Time Complexity, Set, and Dictionary

Name: ____________________ Date: ____________________

Instructions

- Answer in the blanks.
- For **time complexity** questions, choose from: **O(1)**, **O(n)**, **O(n²)**.
- `n` means the input size (usually `n = len(list)` or `n = len(string)`).
- For "what does it print" questions, write the **exact** output (line by line).
  If a set is printed, **order may vary**.

Part A — Concepts + smart choices

1) Choose the best container (synergy)

For each task, choose the best data structure: `list`, `set`, or `dict`.
Then write the **main operation** and its **time complexity**.

Example format: `set — check membership — O(1)`

1. Track **unique** guest names at a party, and quickly check if `"Ben"` has arrived:
   Answer: ____
2. Store **phone numbers** by name, like `"Amy" → "555-1234"` and look up `"Amy"` fast:
   Answer: ____
3. Keep **scores in order** (first score, second score, third score…) and print them in that order:
   Answer: ____

---

2) Fill in the blanks (set vs list membership)

Complete the sentences.

- Checking `x in my_list` (a normal Python list) is usually __________. (Choose from: `O(1)` and `O(n)`.)
- Checking `x in my_set` (a Python set) is usually __________. (Choose from: `O(1)` and `O(n)`.)
- Checking `x in my_dict` checks only the dictionary's __________. (Choose from: `keys` and `values`.)

3) What can be a set element / dict key?

Circle **YES** or **NO** for each item.

Item	Can be in a `set`?	Can be a `dict` key?
`7` (int)	YES / NO	YES / NO
`"cat"` (string)	YES / NO	YES / NO
`(1, 2)` (tuple)	YES / NO	YES / NO
`[1, 2]` (list)	YES / NO	YES / NO

Hint:

- A set element / dict key must be **hashable** (usually **immutable**).
- Numbers, strings, and tuples are **hashable**.
- Lists, sets, and dictionaries are **not hashable**.

4) Common keys between dictionaries

You have two dictionaries: `a` and `b`. Write code to make a set named `common` that contains the names that are keys in **both** dictionaries.

```
a = {"Amy": 3, "Ben": 5, "Chloe": 1}
b = {"Ben": 2, "Drew": 9}

s1 = _____________________ # set containing keys of dictionary `a`
s2 = _____________________ # set containing keys of dictionary `b`
s = ______________________

print(s)
```

5) Fast “seen before?” logic

You are reading words one by one. You want to detect if a word has appeared before.

Which is better for `seen_before(word)` and why?

- Option A: store words in a `list` and use `word in words_list`
- Option B: store words in a `set` and use `word in words_set`

Answer:

```
Time complexity of Option A is ______________.
Time complexity of Option B is ______________.
Option ___ is better.
```

6) Time complexity of a “build then look up” plan

You have a list `pairs` of `n` (name, score) pairs, like:

```
pairs = [("Amy", 3), ("Ben", 5), ...]
```

Plan:

- 1) Build a dictionary `d` from `pairs` (so `d[name] = score`)
- 2) Look up one name: `d["Ben"]`

Time complexity:

- Step 1 is ____________________
- Step 2 is ____________________

Part B — Code reading (what does it print?)

7) Unique vs counts (set + dict together)

```
items = ["A", "B", "A", "C", "B"]

seen = set()
count = {}

for x in items:
    if x not in seen:
        seen.add(x)
    count[x] = count.get(x, 0) + 1

print(seen)        # order may vary
print(count["B"])
print(len(seen))
```

Output:

```

```

8) Common players (dict keys → set intersection)

```
team1 = {"Amy": 10, "Ben": 8, "Chloe": 6}
team2 = {"Ben": 7, "Drew": 9, "Eva": 5}

common = set(team1) & set(team2)

print(common)   # order may vary
print(len(common))
```

Output:

```

```

9) Updating scores (dict overwrite)

```
scores = {"Amy": 2, "Ben": 4}
updates = [("Ben", 10), ("Chloe", 1), ("Ben", 7)]

for name, new_score in updates:
    scores[name] = new_score

print(scores)   # dict order may vary
print(scores["Ben"])
```

Output (order may vary):

```

```

10) “Any overlap?” two ways

Assume `n = len(a)` and `n = len(b)`.

```
a = [1, 2, 3, 4, 5]
b = [5, 6, 7, 8, 9]

overlap = False
for x in a:
    if x in b:
        overlap = True
        break
```

Time complexity (choose one): **O(1)** / **O(n)** / **O(n^2)**

Explanation:

- The loop runs up to ____________ times.
- `x in b` is a ___________ (list/set/dict?) membership check. Its time complexity is ___________.
- Thus the overall time complexity is ___________.

11) “Any overlap?” using a set (faster idea)

Assume `n = len(a)` and `n = len(b)`.

```
a = [1, 2, 3, 4, 5]
b = [5, 6, 7, 8, 9]

b_set = set(b)

overlap = False
for x in a:
    if x in b_set:
        overlap = True
        break
```

Time complexity (choose one): **O(1)** / **O(n)** / **O(n²)**
Answer: ____________________

Why is it faster than Question 10 (one sentence)?

- Building `b_set` has ____________ time complexity.
- Then we do ____________ membership checks in a set. Each costs ____________ time complexity.
- Totally ____________ time complexity.

12) One bug: set method name

The code has **one bug**. Fix it.

```
colors = set()
colors.append("red")
colors.append("blue")
print(colors)
```

Write the corrected code line(s) only:

```

```

Part C — Write / fill code (synergistic tasks)

13) Build a frequency dictionary (fill in blanks)

Complete the code so it counts how many times each number appears.

Choose from: `get` , `0` , `+ 1` , `nums` , `counts` .

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

counts = {}

for x in nums:
    counts[x] = counts.__________(x, ____) ________

print(counts)
```

Expected output (order may vary):

```
{2: 3, 5: 2}
```

14) Find duplicates (dict + set together)

Write code to build:

- `counts` : a dictionary counting each fruit
- `dupes` : a set of fruits that appear **2 or more times**

```
fruits = ["apple", "banana", "apple", "orange", "banana", "apple"]

counts = {}
dupes = set()

# Write code here:









print(counts)
print(dupes)   # order may vary
```

15) Set operations from lists (union + intersection)

You have two lists of students:

```
math = ["Amy", "Ben", "Chloe", "Drew"]
music = ["Ben", "Eva", "Chloe"]
```

Write code to print:

- 1) everyone in **either** club (union)
- 2) everyone in **both** clubs (intersection)

```
# Write code here:
```

16) First repeated word (fast check with a set)

Write code that prints the **first** word that repeats.

```
words = ["hi", "cat", "dog", "cat", "hi"]
```

Expected output:

```
cat
```

Write code here (no functions needed):

```
words = ["hi", "cat", "dog", "cat", "hi"]

seen = ___________________ # list, set, or dict?

for w in words:
    if ________________:
        print(w)
        break
    seen.add(w)
```

17) Mini inverted index (dict of sets) — fill in blanks

We have “lines” of text. Build a dictionary `index` where:

- key = a word
- value = a set of line numbers where the word appears

Example: if `"cats"` appears in lines 0 and 1, then `index["cats"] == {0, 1}`

```
lines = [
    "cats like milk",
    "dogs like bones",
    "cats and dogs"
]

index = {}

for i in range(len(lines)):
    words = lines[i].split()
    for w in words:
        # Fill in the blanks:
        if w not in index:
            index[w] = ____________
        index[w].__________(i)

print(index["cats"])
print(index["like"])
```

Expected output (order may vary for sets):

```
{0, 2}
{0, 1}
```

18) Pick the faster plan (big-O + explanation)

You want to remove duplicates from a list of `n` items.

Plan A:

- Start with an empty list `unique`
- For each item, check `if item not in unique:` then append

Plan B:

- Convert to a set once: `unique = set(items)`

Questions:

- 1) Plan A time complexity: ______________________
- 2) Plan B time complexity: ______________________
- 3) Why?
    - Plan A does up to ___________ list membership checks. Each check can take up to ___________ time complexity.
    - Plan B scans the list once to build a set.