Quiz 18–23: Time Complexity and Sets

Part A — Time Complexity ($O(1)$, $O(n)$, $O(n^2)$)

1) Quick facts (fill in the blanks)

1. **$O(n)$**
2. **$O(n^2)$**
3. **$O(1)$**

2) Worst-case search (time complexity)

Worst-case time complexity: **$O(n)$**

3) Two passes (time complexity)

Worst-case time complexity: **$O(n)$**

4) Set membership vs list membership (time complexity)

1. Worst-case time complexity of `40 in numbers` : **$O(n)$**
2. Average time complexity of `40 in s` : **$O(1)$**

5) Nested loops with a condition (time complexity)

Time complexity: **$O(n^2)$**

6) “Build then check” (time complexity)

Overall time complexity (whole program): **$O(n)$**

7) Which approach is faster? (explain)

1. Worst-case time complexity of Approach A: **O(n^2)**
2. Worst-case time complexity of Approach B: **O(n)**
3. Explanation (example):
   - Approach B turns `allowed` into a set once (**O(n)**), then each membership check is **O(1)** on average.
   - Approach A does **O(n)** list membership inside a loop of **n** queries.

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

Correct choice: **O(n^2)**

9) Improve the complexity (short answer)

Answer (example):

- Convert the list to a set once, e.g. `s = set(numbers)`, then check `x in s`.

Part B — Set Basics + Basic Operations

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

Output:

```
6
3
True
```

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

One correct fix:

```
seen = set()
```

```
print(type(seen))
```

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

Blanks:

- First blank: **in**
- Second blank: **not in**

Filled code:

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

s.discard("cat")
print("cat" not in s)
```

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

One correct output (order may vary):

```
{1, 2, 3, 4, 5}
{3, 4}
```

14) Unique letters (write code)

One correct filled code:

```
text = "mississippi"

letters = set()  # create an empty set

for ch in text:
    letters.add(ch)

print(letters)
```

One correct output (order may vary):

```
{'m', 'i', 's', 'p'}
```

15) Filter with a set (write code)

One correct solution:

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

long_words = set()

for w in words:
    if len(w) >= 3:
        long_words.add(w)

print(long_words)
```

One correct output (order may vary):

```
{'ant', 'cat', 'dog', 'bee'}
```

Part C — Remove Duplicates + Union + Intersection

16) Stable remove duplicates (write code)

Filled code:

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

seen = set()
unique = []

for x in nums:
    if x not in seen:
        seen.add(x)
        unique.append(x)
```

```
print(unique)
```

17) Find duplicates (write code)

Filled code:

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

seen = set()
dups = set()

for x in nums:
    if x in seen:
        dups.add(x)
    else:
        seen.add(x)

print(dups)
```

One correct output (order may vary):

```
{1, 2}
```

18) Common items from two lists (write code)

One correct solution:

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

both = set(A) & set(B)

print(both)
```

One correct output (order may vary):

```
{'Ben', 'Dylan'}
```

19) Either team (write code)

One correct solution:

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

either = set(team1) | set(team2)

print(either)
```

One correct output (order may vary):

```
{'A', 'B', 'C', 'D', 'E'}
```

---

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

One correct solution:

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

count_either = len(math | swim)
count_both = len(math & swim)

print(count_either)
print(count_both)
```

Output:

```
5
1
```

---

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

Filled code:

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

nonzero = set()  # create an empty set
zero_count = 0

for x in nums:
    if x == 0:
        zero_count = zero_count + 1
    else:
        nonzero.add(x)

print(nonzero)
print(zero_count)
```

Output (order may vary for the set):

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

---

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

Filled code:

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

seen = set()
found = False

for x in nums:
    if target - x in seen:
        found = True
        break
    seen.add(x)

print(found)
```

Worst-case time complexity: **O(n)** (assuming average **O(1)** set membership and insert)