Prep 9 Quiz

Due Mar 15 at 9pm	**Points** 5	**Questions** 5	**Available** until Mar 15 at 9pm	**Time Limit** None	**Allowed Attempts** Unlimited

Instructions

Readings

Please read the following parts of the **Course Notes** .

- Pages 101–110.

General instructions

You can review the general instructions for all prep quizzes on the **Course Syllabus**. Remember that you can submit multiple times! You might consider printing this quiz out so that you can work on paper first.

This quiz was locked Mar 15 at 9pm.

Attempt History

	Attempt	Time	Score
KEPT	**Attempt 3**	less than 1 minute	5 out of 5
LATEST	**Attempt 3**	less than 1 minute	5 out of 5
	Attempt 2	1 minute	4.42 out of 5
	Attempt 1	2 minutes	1.92 out of 5

Correct answers are hidden.

Score for this attempt: **5** out of 5

Submitted Mar 15 at 5:12pm

This attempt took less than 1 minute.

Question 1

1 / 1 pts

Let $S \subseteq \mathbb{R}$ be a non-empty finite set of real numbers, and let $M \in \mathbb{R}$.

Which of the following expressions is equivalent to the English statement "M is an *upper bound* on the maximum value of S"?

- $\exists x \in S,\ x \leq M$
- **$\forall x \in S,\ x \leq M$**
- $S \leq M$
- $\forall x \in S,\ x < M$
- $M \in S \wedge (\forall x \in S,\ x \leq M)$

Question 2

1 / 1 pts

Let $S \subseteq \mathbb{R}$ be a non-empty finite set of real numbers, and let $M \in \mathbb{R}$. Which of the following expressions is equivalent to the English statement "M is a *lower bound* on the maximum value of S"?

- $\forall x \in S,\ x \geq M$
- $\exists x \in S,\ x \leq M$
- $M \leq S$
- $M \in S$
- **$\exists x \in S,\ x \geq M$**

Question 3

1 / 1 pts

Consider the following function:

```
def has_duplicates(lst: list) -> bool:
    """Return whether lst contains any duplicate values."""
    i = 0
    while i < len(lst):
        j = i + 1
        while j < len(lst):
            if lst[i] == lst[j]:
                return True
            j += 1
        i += 1
```

```
    return False
```

Select all of the following Big-O expressions that are an *upper bound on the worst-case running time* of `has_duplicates`.

- [x] $\mathcal{O}(n^2)$
- [ ] $\mathcal{O}(n)$
- [x] $\mathcal{O}(n(n+1))$
- [x] $\mathcal{O}(2^n)$
- [x] $\mathcal{O}(n^3)$

Question 4

1 / 1 pts

Recall function `has_duplicates` from the previous question:

```
def has_duplicates(lst: list) -> bool:
    """Return whether lst contains any duplicate values."""
    i = 0
    while i < len(lst):
        j = i + 1
        while j < len(lst):
            if lst[i] == lst[j]:
                return True
```

```
            j += 1
        i += 1

    return False
```

Select all of the following Omega expressions that are a *lower bound on the worst-case running time* of `has_duplicates`.

- [ ] $\mathcal{O}(2^n)$
- [x] $\Omega(n(n+1))$
- [ ] $\Omega(n^3)$
- [x] $\Omega(n^2)$
- [x] $\Omega(n)$

Question 5

1 / 1 pts

One of the algorithms we'll study in lecture this week is string-based, and this question is designed to prepare you for this analysis. Here are two definitions about strings.

Definition 1 (palindrome). Let *s* be a string. We say that *s* is a **palindrome** when its reversal (the string obtained by taking the letters of *s* in reverse order) equals the original string. For example, the string "abbcbba" is a palindrome. The empty string and all strings of length 1 are palindromes.

Definition 2 (prefix). Let *r* and *s* be strings. We say that *r* is a **prefix** of *s* when *r* appears at the start of *s* (in a contiguous block). For example, the string "abc" is a prefix of "abcdef", but it is not a prefix of "abdcef" or "defabc". Every string is a prefix of itself, and the empty string is a prefix of every string.

Now, consider the following algorithm:

```
def palindrome_prefix_simple(s: str) -> int:
    """Return the length of the longest prefix of s
    that is a palindrome. (Assume s is non-empty.)"""
    n = len(s)
    max_length = 0
    for prefix_length in range(1, n + 1):  # goes from 1 to n
        # Check whether s[0:prefix_length] is a palindrome
        is_palindrome = True
        for i in range(prefix_length):
            if s[i] != s[prefix_length - 1 - i]:
                is_palindrome = False

        # If a palindrome prefix is found, update max_length.
        if is_palindrome:
            max_length = prefix_length

    return max_length
```

Select all of the following Big-O expressions that are an *upper bound* on the worst-case running time of `palindrome_prefix_simple`.

- [ ] $O(n)$
- [x] $O(n^3)$
- [x] $O(n^2)$

- [x] $O(2^n)$

Quiz Score: **5** out of 5