a. Write a pseudocode.

```
function next_palindrome
    pass in: number
    number += 1 # 1
    digits = list of extracted digits # d or O(log n)

    # indices
    i = 0 # 1
    j = length of digits list - 1 # 1

    # while i is not equal to j
    while i < j: # d/2 or O(log n)
        if digits[j] > digits[i]: # 1
            digits[j - 1] += 1 # 1

            # carry propagation
            x = j - 1 # 1
            while digits[x]  >= 10: # O(1) amortized
                digits[x] % 10
                x += 1
                digits[x] += 1

        last digit = first digit # 1
        i += 1 # 1
        j -= 1 # 1

    return concatenated digits # O(log n)
```

b. Analyse the performance of the code given the structure of an input.

I. What is the best-case complexity of your program? What is the structure of the input that results in the best case?

O(d) or O(log n), where d is the number of digits in the number. The number of digits in a number can also be calculated using log_10(number) + 1, hence the O(log n) time complexity. Extraction of the digits has a time complexity of O(d) -> O(log n). Looping through half the digits (while i < j) is O(d/2) -> O(log n). Constructing the integer back is also O(d) -> O(log n). Therefore, the best-case complexity of my program is O(log n).

II. What is the worst case complexity of the program? What is the structure of the input that results in the worst case?

O(d) or O(log n). Extraction of the digits has a time complexity of O(d) -> O(log n). Two halves of a list are essentially compared and equalized (while i < j) until the middle of the list is reached. This prevents us from brute forcing a palindrome check from the input number until the next smallest palindrome. The case of a carry propagation is run for a total max of all digits O(log n), that is O(1) amortized throughout the outer loop. Constructing the integer back is also O(d) -> O(log n). Therefore, the worst-case complexity of my program is O(log n).

c. Check the correctness and performance of your program by testing it on a set of inputs.

lab1_final.py

```python
# extract list of digits out of an integer
def extract_digits(n: int) -> list[int]:
    return list(map(int, str(n)))

# concat list of digits back to an integer
def make_int(digits: list[int]) -> int:
    return int("".join(map(str, digits)))

def next_palindrome(n: int) -> int:
    n += 1
    digits = extract_digits(n)

    i = 0
    j = len(digits) - 1
    while i < j:
        if digits[j] > digits[i]:
```

PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL

```
\CMSC 142-Lab1\output\small_output' and 'C:\Users\ASUS\Desktop\BSCS 3
res-and-Algorithms\2025\CMSC 142-Lab1\output\full_output'.
PS C:\Users\ASUS\Desktop\BSCS 3 -  2nd Sem\CMSC 142\Data-Structures-a
 python lab1checker.py
Run checker for: (1) Small Input / (2) Full Input ?
Choice: 1

Checking output for mistakes...

Number of mistakes: 0
Final result: 5/5 = 100.00% accuracy
Time elapsed: 0.0037450790405273438 seconds
```

```
PS C:\Users\ASUS\Desktop\BSCS 3 -  2nd Sem\CMSC 142\Da
 python lab1checker.py
Run checker for: (1) Small Input / (2) Full Input ?
Choice: 2

Checking output for mistakes...

Number of mistakes: 0
Final result: 100001/100001 = 100.00% accuracy
Time elapsed: 0.7273902893066406 seconds
```