1)

We can use a trace table for the first step (loop invariant).
Let us take a small input, say lst = [1, 2, 3, 4, 5]

n = len(lst)
  = 5

i = 0

j = n - 1
  = 5 - 1
  = 4

Tracing Table:

i	j	Loop Guard ($i \leq j$)	Swapping process swap(lst[i], lst[j])
0	4	$0 \leq 4$	[5, 2, 3, 4, 1]
1	3	$1 \leq 3$	[5, 4, 3, 2, 1]
2	2	$2 \leq 2$	[5, 4, 3, 2, 1]
3	1	$3 \leq 1$	[5, 4, 3, 2, 1]

By now, we see a pattern in the trace table. We can see that there is an inverse linear relationship between variables ‘ i ‘ and ‘ j ‘. As ‘ i ‘ increments by 1 every iteration, ‘ j ‘ decrements by 1 every iteration.

An important thing to note is that when i = j, the loop will terminate in the next iteration because in the next iteration, the loop guard will be violated.

Based on the given loop, the variables associated with the loop, and the trace table, here is the loop invariant:

$inv(i, j)$: $(0 \leq i \leq n) \ \wedge \ (-1 < j \leq n - 1)$

Proof of Invariant:

Base Case:

Before entering the loop, the following values are assigned to the variables in question (i and j).

i = 0 and j = n - 1

So,
$inv(0,\ n-1)\colon (0 \le 0 \le n) \wedge (-1 \le n-1 \le n-1)$

The first part of the invariant (0 ≤ 0 ≤ n) will hold because the variable ‘ n ‘ is the length of the input list, and that is always greater than or equal to zero.

The second part of the invariant (-1 ≤ n - 1 ≤ n - 1) will also hold because the length of the input list subtracted by one will be greater than or equal to a negative one.

Both parts hold so the base case is satisfied.

Inductive Hypothesis:

Let i_1 and j_1 be variables in some random iteration of the loop. Upto this iteration ‘ L ’ the invariant holds. At this iteration ‘ L ‘ the values of the variables are i_1 and j_1

So,

$inv(i_1,\ j_1)\colon (0 \le i_1 \le n) \wedge (-1 \le j_1 \le n-1)$

Performing one more iteration. The following values of ‘ i ‘ and ‘ j ‘ are after the iteration $L+1$ is complete. So we will have,

$i_2 = i_1 + 1$ and $j_2 = j_1 - 1$

$inv(i_2,\ j_2)\colon (0 \le i_2 \le n) \wedge (-1 \le j_2 \le n-1)$

Let us examine parts by parts of the invariant.

Let us examine $(0 \le i_2 \le n)$

$(0 \leq i_2 \leq n)$

= $(0 \leq i_1 + 1 \leq n)$

This holds true since i < n and incrementing the variable i by 1 at each iteration keeps it within the bounds.

Let us examine $(-1 \leq j_2 \leq n - 1)$

$(-1 \leq j_2 \leq n - 1)$

= $(-1 \leq j_1 + 1 \leq n - 1)$

This also holds true since j > -1 and the variable j is decremented by 1 at each iteration keeping it within the bounds.

Assuming the loop terminates, we show the postcondition.

Assuming the loop terminates, the loop guard does not hold, so the negation of the guard holds, which is i > j.

When the loop terminates (our assumption is that it does terminate), the invariant $(0 \leq i \leq n) \wedge (-1 \leq j \leq n - 1)$ holds true. The condition i > j holds because the indices i and j have crossed each other, meaning every pair of elements that needed to be swapped has been swapped.

(Note that we do not know the implementation of the swap function, assuming the swap function is implemented correctly, the elements in the list will be swapped.)

Furthermore, when i > j,

$i = \lfloor \frac{n}{2} \rfloor + 1$ and $j = \lfloor \frac{n}{2} \rfloor - 1$

$inv(i, j)$: $(0 \leq \lfloor \frac{n}{2} \rfloor + 1 \leq n) \wedge (-1 \leq \lfloor \frac{n}{2} \rfloor - 1 \leq n - 1)$

The pairs being swapped are: $(0, n-1), (1, n-2), \ldots, (\lfloor \frac{n}{2} \rfloor - 1, \lceil \frac{n}{2} \rceil)$

The condition i > j makes sure that pairs have been swapped in the list (satisfies the postcondition).

Proof of Termination:

With the variable ‘ i ‘ incrementing and ‘ j ‘ decrementing every iteration, ‘ i ‘ will eventually be greater than ‘ j ‘, so i ≤ j will become false.

We have the following loop variant:

V = 1+ j - i

On the first iteration, the value of the variant is the highest, and the value of the variant at the end of the iteration is going to be the lowest.

We prove the following: $V \in \mathbb{N}$

Before we enter the loop,
i = 0
j = n - 1

n = len(lst)
Due to the precondition, n ≥ 0 (array of any length)

Initially,

V = 1+ j - i
  = 1+ (n - 1) - i
  = 1 + (n - 1) - 0
  = 1 + n - 1
  = n

So this is natural. The highest value initially.

When we are in any iteration, that means we are passed the loop guard. The loop guard is i ≤ j

So,

i ≤ j
⇒ 0 ≤ j - i
⇒ j - i ≥ 0

To show V = 1 + j - i decreases,

Let j_1 and i_1 be on a iteration. Let's perform one more iteration, we have:

$j_2 = j_1 - 1$ and $i_2 = i_1 + 1$

Checking the difference:

$$(1 + j_1 - i_1) - (1 + j_2 - i_2)$$
$$= 1 + j_1 - i_1 - 1 - j_2 + i_2$$
$$= j_1 - i_1 - j_2 + i_2$$
$$= j_1 - i_1 - (j_1 - 1) + (i_1 + 1)$$
$$= 2$$

Since 2 > 0, the function is strictly decreasing at each iteration of the loop. This shows that the code terminates.

2)

Let us examine the first loop at line 2. We can see that the loop starts at 0 and runs till it reaches n - 1, so it can be written as the following:

$$\sum_{i=0}^{n-1} c \quad , c \in \mathbb{N}$$

The variable ‘c’ is used to denote some constant. The body of the first loop performs a simple arithmetic operation that takes constant time so we are denoting it as ‘c’ for that constant operation. Let us evaluate the summation for the first loop.

With the help of the following summation formula, we can evaluate our original sum.

$$\sum_{i=m}^{n} c = c(n + 1 - m)$$

So,

$$\sum_{i=0}^{n-1} c = c(n - 1 + 1 - 0)$$
$$= c(n)$$

So, we have:
$O(c * n)$ but c is just some constant so we can drop it. We are left with:
$O(n)$

So the average runtime for the first loop is $O(n)$

The second loop is also running ‘n’ times and in the body of the loop, we are checking the contents of the input array’s index to see if it matches the ‘check_digit’ variable. Those simple operations take constant time.

To compute the average runtime for this algorithm we do the following:

For the input we have,
$A[i] \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, where i is the index of the array A.

We have to calculate the expectation of running time,

$$E[t_n] = \sum_{t=1}^{n+1} t * p(t_n = t)$$

To get the probability distribution, we do the following:

$$p(t_n = 1) = \frac{1}{10}$$

$$p(t_n = 2) = (\frac{9}{10})(\frac{1}{10})$$

$$p(t_n = 3) = (\frac{9}{10})(\frac{9}{10})(\frac{1}{10}) = (\frac{9}{10})^2(\frac{1}{10})$$

$$p(t_n = 4) = (\frac{9}{10})(\frac{9}{10})(\frac{9}{10})(\frac{1}{10}) = (\frac{9}{10})^3(\frac{1}{10})$$

…

$$p(t_n = n) = (\frac{9}{10})^{n-1}(\frac{1}{10})$$

$$p(t_n = n + 1) = (\frac{9}{10})^n(\frac{1}{10})$$

$$p(t_n = t) = \begin{cases} \left(\frac{9}{10}\right)^{t-1}\left(\frac{1}{10}\right) & t_n \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \\ \left(\frac{9}{10}\right)^{n}\left(\frac{1}{10}\right) & t = n + 1 \end{cases}$$

So we have the following, and we simplify it

$$E[t_n] = \sum_{t=1}^{n} t[(\frac{9}{10})^{t-1}(\frac{1}{10})] + (n + 1)[(\frac{9}{10})^n(\frac{1}{10})]$$

$$= \frac{1}{10}\sum_{t=1}^{n} t(\frac{9}{10})^{t-1} + (n + 1)[(\frac{9}{10})^n(\frac{1}{10})]$$

Let m = $\frac{9}{10}$, so we have:

$$= \frac{1}{10}\sum_{t=1}^{n} t(m)^{t-1} + (n + 1)[(m)^n(\frac{1}{10})]$$

$$= \frac{1}{10}\sum_{t=1}^{n} t(m)^{t-1} + \frac{nm^n}{10} + \frac{m^n}{10}$$

$$= \frac{1}{10}\sum_{t=1}^{n} t(m)^{t-1} + \frac{1}{10}(nm^{n} + m^{n})$$

Lets examine at the summation further. Notice that the expression $t(m)^{t-1}$ is the derivative of m^{t}, we will come back to this. For now, we can use the infinite geometric series formula, which is given as,

$$\sum_{i=0}^{\infty} ar^{k} = \frac{a}{1-r} \quad , r \neq 1$$

In our case,

$a = \frac{1}{10}$ where a is some constant that is the same throughout the entire summation. So we can leave it out of the sum.

$r = m = \frac{9}{10}$

$k = t - 1$

We can use that derivative of the sum of geometric series,

$\frac{d}{dr}(\frac{1}{1-r})$

$= \frac{d}{dr}\left[(1 - r)^{-1}\right]$

$= -1(1 - r)^{-1-1}(-1)$ Applying chain rule and simplifying further

$= -(1 - r)^{-2}(-1)$

$= -\frac{1}{(1-r)^{2}}(-1)$

$= \frac{1}{(1-r)^{2}}$

So we have,

$$\sum_{t=1}^{\infty} k(r)^{k-1} = \frac{1}{(1-r)^{2}}$$

Let us plug this into the original equation,

$$E[t_n] = \frac{1}{10}\left[\sum_{t=1}^{n} t(m)^{t-1} + (nm^n + m^n)\right]$$
$$= \frac{1}{10}\left[\frac{1}{(1-r)^2} + (nm^n + m^n)\right]$$

We know the value of r which is m = $\frac{9}{10}$, so we can plug that in too and simplify,

$$= \frac{1}{10}\left[\frac{1}{(1-m)^2} + (nm^n + m^n)\right]$$

$$= \frac{1}{10}\left[\frac{1}{\left[1-\left(\frac{9}{10}\right)\right]^2} + n\left(\frac{9}{10}\right)^n + \left(\frac{9}{10}\right)^n\right]$$

$$= \frac{1}{10}\left[100 + n\left(\frac{9^n}{10^n}\right) + \frac{9^n}{10^n}\right]$$

$$= 10 + \frac{n * 9^n}{10^{n+1}} + \frac{9^n}{10^{n+1}}$$

$$= 10 + \left(\frac{1}{10^{n+1}}\right)\left(n * 9^n + 9^n\right)$$

$$= 10 + 9^n\left(\frac{n+1}{10^{n+1}}\right)$$

Now, let us look at the dominant terms of the above. Let us look at this expression in our equation:

$$\frac{n+1}{10^{n+1}}$$

We can determine the dominant term by performing an asymptotic limit comparison of the numerator and denominator. Asymptotic limit comparison says that function $f(n)$ becomes insignificant compare to $g(n)$ as n approaches infinity. This can be written as the following:

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

In our case,

$f(n) = n + 1$ and $g(n) = 10^{n+1}$

So,

$$\lim_{n \to \infty} \frac{f(n)}{g(n)}$$

$= \lim_{n \to \infty} \frac{n+1}{10^{n+1}}$ Using L'Hopital's Rule and simplify further

$= \lim_{n \to \infty} \frac{1}{ln(10) * 10^{n+1}}$

$= \left[\frac{1}{ln(10)}\right]\left(\lim_{n \to \infty} \frac{1}{10^{n+1}}\right)$

$= \left[\frac{1}{ln(10)}\right]\left(\frac{1}{\infty}\right)$

$= 0$

The limit comparison gave us 0, so that means the the numerator $f(n)$ is insignificant compare to the denominator $g(n)$.

We can now continue with our original equation,

$$E[t_n] = 10 + 9^n\left(\frac{n+1}{10^{n+1}}\right)$$

$$= 10 + 9^n(0)$$

$$= 10$$

We have just evaluated the average runtime for the second loop at line 4 which is constant time.

$E[t_n] \in O(1)$

Finally, to calculate the average runtime for this algorithm, we combine the analysis for both of the loops. By combining $O(n)$ from the first loop, and $O(1)$ from the second loop, we get $O(n)$ as the average runtime for this algorithm.

3 i) This function returns true if the input is a prime number, and it returns false if the input is not a prime number. So, the postcondition is the following:

POST: Returns true if the input is a prime number, otherwise returns false.

ii) The given while loop in the code has the following loop guard: x < n, and in the body of the loop, we have an if statement checking for equality, and the variable ‘ x ‘ incrementing by one each iteration. Let us look at the worst-case scenario, which is when the input is a prime number. This is the worst case because the code will have to perform additional steps to determine that the input is indeed a prime number. In the worst case, the program will execute arithmetic operations (modulus and addition), which is done in constant time.

The complexity can be found by the number of iterations, so it is **O(n)**.

iii) We can say that the big-theta complexity of this function is $\Theta(n)$

4)

f	O(g)
$3 * 2^n$	$O(2^n)$
$\frac{2n^4+1}{n^3+2n-1}$	$O(n)$
$(n^5 + 7)(n^5 - 7)$	$O(n^{10})$
$\frac{n^4-nlog_2 n}{n^2+1}$	$O(n^2)$
$\frac{nlog_2 n}{n-5}$	$O(log_2 n)$
$8 + \frac{1}{n^2}$	$O(1)$
2^{3n+1}	$O(10^n)$
$n!$	$O(n^n)$
$\frac{5log_2 n+1}{1+nlog_2 3n}$	$O(\frac{1}{n})$
$(n - 1)log_2(n^3 + 4)$	$O(nlog_2 n)$