Use the definition of "big Oh" to prove that $\frac{3n+4}{n^2+2}$ is $O(\frac{1}{n})$.

To prove, we need to show a constant $C > 0$ and value $n_0 \geq 1$, such that for all $n \geq n_0$: $\frac{3n+4}{n^2+2} \leq C\frac{1}{n}$.

Since $\lim_{n \to \infty} \frac{3n+4}{n^2+2} = \lim_{n \to \infty} \frac{3}{n}$, so effectively $\frac{3}{n} \leq \frac{C}{n}$.

For $C = 3$ and $n_0 = 1$, $\frac{3n+4}{n^2+2} \leq \frac{3}{n}$ for $n \geq 1$.

Since $\frac{3n+4}{n^2+2}$ is bounded by $C\frac{1}{n}$, we can conclude $\frac{3n+4}{n^2+2}$ is $O(\frac{1}{n})$.

Let $F_1(n),\ F_2(n),\ G_1(n),\ G_2(n)$ be positive functions. If $F_1(n)$ is $O(G_1(n))$ and $F_2(n)$ is $O(G_2(n))$, using the definition of "big Oh" show that $F_1(n) \cdot F_2(n)$ is $O(G_1(n) \cdot G_2(n))$.

We know $F_1(n) \leq C_1 \cdot G_1(n)$ for $n \geq n_1$ and $F_2(n) \leq C_2 \cdot G_2(n)$ for $n \geq n_2$.

We multiply the inequalities to get: $F_1(n) \cdot F_2(n) \leq (C_1 \cdot C_2)(G_1(n) \cdot G_2(n))$.

Meaning $F_1(n) \cdot F_2(n)$ is $O(G_1(n) \cdot G_2(n))$ for $C = C_1 \cdot C_2$ and $n_0 = max(n_1, n_2)$.

Use the definition of "big Oh" to prove that 4^n is not $O(2^n)$.

To prove, we need to show no constant $C > 0$ and value $n_0 \geq 1$ exists, such that for all $n \geq n_0$: $4n \leq C \cdot 2^n$.

$4^n = 2^{2n} < C \cdot 2^n$

$2^n < C \cdot 1$

As $n \uparrow$, $2^n \uparrow$ but C remains. So C cannot bound 2^n for sufficiently large n.

Data: String T with n lowercase letters
Result: The number of palindrome substrings in string T

```
c ← 0;
for i ← 0 to n − 1 do
    for j ← i to n − 1 do
        substring ← T[i to j];
        if isPalindrome(substring) then
            c ← c + 1;
        end
    end
end
return c
```

Prove the algorithm terminates.

The outer loop is bounded by n, and will always terminate.

The inner loop is also bounded by n, and will always terminate.
$isPalindrome()$ is assumed to terminate in constant-time.
Thus, for any input string of length n, the algorithm will always terminate after examining all possible substrings of length n.

Show the algorithm always produces the correct answer.

The algorithm iterates through all possible substrings of the string T. The substring is defined by indices i and j, where i is the starting index and j is the ending index of the substring.
For each substring T[i to j], the algorithm calls isPalindrome() to check if the substring is a palindrome.
If the substring is a palindrome, the counter c is incremented.
The algorithm guarantees that all substrings are checked exactly once and that only palindromic substrings are counted.
Thus, for any input string T of length n, the algorithm correctly computes the number of palindromic substrings by ensuring that isPalindrome() is called for all substrings, and it counts correctly based on the function's output.

Compute the time complexity of this algorithm in the worst case (4 marks)

The algorithm has two nested loops (outer: n times, inner: n - i times for each i). As such, the total number of iterations is:
$\sum_{i=0}^{n-1} (n - i) = \frac{n(n+1)}{2}$, which simplifies to $O(n^2)$. Inside the loops, all methods are constant-time, so the time complexity remains.
Thus, the total time complexity of the algorithm, in the worst case (where every possible substring is checked) is $O(n^2)$.

```
Data: Array A storing n integer values
Result: None
i ← 0;
while i < n do
    if i = 0 then
    |   step ← 1
    end
    else if A[i] > 0 then
        A[i] ← −A[i];
        step ← −1;
    end
    i ← i + step;
end
```

Compute the best-case time complexity.

In the best case, $A[i] = 0$ always. Therefore, i is always incremented and the number of iterations will always be n. Therefore, the best-case time complexity is $O(n)$, because a single pass over n elements are performed and all other methods are constant-time.

Compute the worst-case time complexity.

In the worst case, $A[0] = 0$ and $A[i > 0] > 0$. On pass $i > 0, A[i] =- A[i]$, and $step =- 1$. This repeats until $i = 0$, where $step = 1$. When $A[i] < 0$, $step$ is not modified. Therefore, i is decremented until the first element, where the process repeats.
For example, for $A = [0, 1, 1, 1]$. Passing occurs in the following order of indices: 0, 1, 0, 1, 2, 1, 0, 1, 2, 3, 2, 1, 2, 3
The pattern is seen from: n=1, iter=1. n=2, iter=4. n=3, iter=9. n=4, iter=16. n=5, iter=25
Therefore, the worst-case time complexity, is $O(n^2)$.