> Note some syntax used:
>
> - `A[i..j]` means a slice of the array from `A[i]` to `A[j]`, inclusive.
> - `a ← b` means assign value `b` to `a` while `a = b` is a boolean expression that compares the value of `a` and `b`. (This is different from the common programming language)
> - `A[1..n] ← []` means the pseudocode declare a 1d array called `A` that is size `n` big. Same for arbitrary n-d array.
> - `A[1..n] ← [c]` means that the pseudocode declare a 1d array called A that is size n big with initial value `c`. Same for arbitrary n-d array.
> - `for i ← m to 1, j ← 1 to m:` is just a syntactic sugar for
>
> ```
> for i ← m to 1:
>     for j ← 1 to m:
> ```
>
> We use this because we don't want too much indent on the code

(a)

We represent the sequence $A = a_1, a_2, \cdots, a_n$ as an array `A[1..n]` in the pseudocode.

To better introduce how we only use $O(n)$ space, we first suppose that there is 2d array called `SPS_memory[1..n, 1..n]` behind the scene (it's not in the actual pseudocode). Where `SPS_memory[i, j]` means the length of shortest palindromic supersequence of the `A[i..j]` minus the length of `A[i..j]` itself. If you see the pseudocode below, the `SPS_memory[i, j]` actually only depends on `SPS_memory[i + 1, j]`, `SPS_memory[i + 1, j - 1]`, `SPS_memory[i, j + 1]`, which means that we only need to store 2 columns of the `SPS_memory` instead of the entire 2d array.

Therefore, within the outer for loop with `i` looping variable, we define `SPS_array[j]` equal to `SPS_memory[i + 1, j]` and `SPS_next_array[j]` equal to `SPS_memory[i, j]`. This will store enough memoization information.

Few things to notice:

- The `i ⩾ j` is a special case, and in this case the `SPS_next_array[j]` need to be defaulted to `0`. (The `SPS_next_array[j]` basically means the length of shortest palindromic supersequence of one single character / empty string minus the length of itself. For single character / empty string, itself is already the shortest palindromic supersequence, therefore no additional length needs to be added).
- We need to evaluate `j` from `1` to `n` since in the pseudocode the `SPS_next_array[j]` depends on `SPS_next_array[j - 1]`. We need to evaluate `i` from `n` to `1` since `SPS_next_array` depends on `SPS_array` (this dependency originate from the fact that `SPS_memory[i, j]` depends on `SPS_memory[i + 1, j]`)
- By definition of `SPS_array`, we will return the `SPS_array[n] + n` as the result.

```
SPS(A[1..n]):
    SPS_array[1..n] ← []
```

```
for i ← n to 1:
    SPS_next_array[1..n] ← []
    for j ← 1 to n:
        if i ⩾ j:
            SPS_next_array[j] ← 0
        else if A[i] = A[j]:
            SPS_next_array[j] ← SPS_array[j - 1]
        else:
            SPS_next_array[j] ← 1 + min(
                SPS_array[j],
                SPS_next_array[j - 1]
            )
    SPS_array ← SPS_next_array

return n + SPS_array[n]
```

We see that there is two for loop in the pseudocode, and the operation in it is $O(1)$ time, therefore the total time complexity is $n^2 \cdot O(1) = O(n^2)$. For space complexity, we see that we only use two array of size n. Therefore, the space complexity is just $2 \cdot O(n) = O(n)$ as we desired.

(b)

We define a 4d array `SCPS_array[1..m, 1..m, 1..n, 1..n]`. The `SCPS_array[i, j, p, q]` means the length of shortest common palindromic supersequence of `X[i..j]` and `Y[p..q]` minus the sum of the length of `X[i..j]` and `Y[p..q]`.

Few things to notice:

- There are few special cases:
    - When `i = j and p = q`, we are considering actually the shortest common palindromic supersequence of two single character, if they are same character, then their shortest common palindromic supersequence is just 1 character, which is 1 character less than their length combined. Otherwise the shortest common palindromic supersequence will be 3 characters, 1 character longer than their length combined
    - When `i > j` and `p > q`, we are considering two empty string, and in this case the shortest common palindromic supersequence could be thought as 0 characters, which is 0 characters longer than the combined length of two empty string.
    - When `i > j and p = q` or `i = j and p > q`, we are considering one single character and one empty string, the shortest common palindromic supersequence is just 1 character long, and 0 character longer than their length combined.
    - When `i > j and p < q` or `i < j and p > q`, we are considering one non-empty string and one empty string, in this case, the shortest common palindromic supersequence is actually just the shortest palindromic supersequence of the non-empty string. (it's similar to part (a))

- The `i` and `p` goes from `m to 1` and `n to 1` respectively, because the `SCPS_array[i, j, p, q]` depends on `SCPS_array[i + 1, j, p, q]` and `SCPS_array[i, j, p + 1, q]`. Similarly, the `j` goes from `1 to m` and `q` goes from `1 to n` because `SCPS_array[i, j, p, q]` depends on `SCPS_array[i, j - 1, p, q]` and `SCPS_array[i, j, p, q - 1]`
- By the definition of `SCPS_array[1..m, 1..m, 1..n, 1..n]`, we will return `m + n + SCPS_array[1, m, 1, n]` in the end

```
SCPS(X[1..m], Y[1..n]):
    SCPS_array[1..m, 1..m, 1..n, 1..n] ← [Infinity]

    for i ← m to 1, j ← 1 to m:
        for p ← n to 1, q ← 1 to n:
            if i = j and p = q:
                if X[i] = Y[p]:
                    SCPS_array[i, j, p, q] ← -1
                else:
                    SCPS_array[i, j, p, q] ← 1
            else if i > j:
                if p > q or p = q:
                    SCPS_array[i, j, p, q] ← 0
                else if Y[p] = Y[q]:
                    SCPS_array[i, j, p, q] ← SCPS_array[i, j, p + 1, q - 1]
                else:
                    SCPS_array[i, j, p, q] ← 1 + min(
                        SCPS_array[i, j, p + 1, q],
                        SCPS_array[i, j, p, q - 1]
                    )
            else if p > q:
                if i > j or i = j:
                    SCPS_array[i, j, p, q] ← 0
                else if X[i] = Y[j]:
                    SCPS_array[i, j, p, q] ← SCPS_array[i, j, p + 1, q - 1]
                else:
                    SCPS_array[i, j, p, q] ← 1 + min(
                        SCPS_array[i + 1, j, p, q],
                        SCPS_array[i, j - 1, p, q]
                    )
            else if X[i] = X[j] = Y[p] = Y[q]:
                SCPS_array[i, j, p, q] ← (-2) + SCPS_array[i + 1, j - 1, p + 1, q
- 1]
            else if X[i] = X[j] = Y[p]:
                SCPS_array[i, j, p, q] ← min(
                    -1 + SCPS_array[i + 1, j - 1, p + 1, q],
                    +1 + SCPs_array[i, j, p, q - 1]
                )
            else if X[i] = X[j] = Y[q]:
                SCPS_array[i, j, p, q] ← min(
                    -1 + SCPS_array[i + 1, j - 1, p, q - 1],
                    +1 + SCPS_array[i, j, p + 1, q]
                )
```

```
            else if X[i] = Y[p] = Y[q]:
                SCPS_array[i, j, p, q] ← min(
                    -1 + SCPS_array[i + 1, j, p + 1, q - 1],
                    +1 + SCPS_array[i, j - 1, p, q]
                )
            else if X[j] = Y[p] = Y[q]:
                SCPS_array[i, j, p, q] ← min(
                    -1 + SCPS_array[i, j - 1, p + 1, q - 1],
                    +1 + SCPS_array[i + 1, j, p, q]
                )
            else if X[i] = X[j]:
                SCPS_array[i, j, p, q] ← min(
                    SCPS_array[i, j, p, q],
                    0 + SCPS_array[i + 1, j - 1, p, q],
                    1 + SCPS_array[i, j, p - 1, q + 1]
                )
            else if Y[p] = Y[q]:
                SCPS_array[i, j, p, q] ← min(
                    SCPS_array[i, j, p, q],
                    0 + SCPS_array[i, j, p + 1, q - 1],
                    1 + SCPS_array[i + 1, j - 1, p, q]
                )
            else if X[i] = Y[p]:
                SCPS_array[i, j, p, q] ← min(
                    SCPS_array[i, j, p, q],
                    0 + SCPS_array[i + 1, j, p + 1, q],
                    1 + SCPS_array[i, j - 1, p, q - 1]
                )
            else if X[j] = Y[q]:
                SCPS_array[i, j, p, q] ← min(
                    SCPS_array[i, j, p, q],
                    0 + SCPS_array[i, j - 1, p, q - 1],
                    1 + SCPS_array[i + 1, j, p + 1, q]
                )
            else if X[i] = Y[q]:
                SCPS_array[i, j, p, q] ← min(
                    SCPS_array[i, j, p, q],
                    0 + SCPS_array[i + 1, j, p, q - 1],
                    1 + SCPS_array[i, j - 1, p + 1, q]
                )
            else if X[j] = Y[p]:
                SCPS_array[i, j, p, q] ← min(
                    SCPS_array[i, j, p, q],
                    0 + SCPS_array[i, j - 1, p + 1, q],
                    1 + SCPS_array[i + 1, j, p, q - 1]
                )
            else:
                SCPS_array[i, j, p, q] ← 1 + min(
                    SCPS_array[i + 1, j, p, q],
                    SCPS_array[i, j, p + 1, q],
                    SCPS_array[i, j - 1, p, q],
                    SCPS_array[i, j, p, q - 1]
```

```
                  )
     return m + n + SCPS_array[1, m, 1, n]
```

We notice that there is four for loop, and the operation in the for loop is $O(1)$, therefore, the time complexity is just $m^2n^2 \cdot O(1) = O(m^2n^2)$. For space complexity, we see that we store a 4d array which need $O(m^2n^2)$ space.