1º)

```
subsequencia_crescente(A, n_A, b, n_B):
    crio uma matriz 2D "dp" de tamanho n_A x n_B com zeros
    para cada i em A e j em B:
        se A[i] == B[j]:
            dp[i][j] = dp[i-1][j-1] + 1
        senão:
            dp[i][j] = max(dp[i-1][j], dp[i][j-1])

    o valor dp[n_A - 1][n_B - 1] dá o comprimento da subsequeência
    crescente comum mais longa.
```

O loop é executado n_Axn_B vezes, e cada operação é O(1) interna.
Portanto, o total é O(n_A * n_B).

2º)

crio uma matriz "dp" de tamanho nxn, onde n é o número de cartas.
pra cada gap de 1 até n:
    para i = 0 até n-gap:
        j = i + gap - 1
        escolha entre a carta do início e a carta do fim
        dp[i][j] = max(min(dp[i+1][j-1], dp[i+2][j]) + carta[i], min(dp[i][j-2], dp[i+1][j-1]) + carta[j])

A matriz dp vai dar a melhor jogada pra qualquer situação.
O loop é executado n^2 por conta do aninhamento, e por isso O(n^2).

3º)

```
palindromes(S, n_S):
    crio uma matriz "dp" de tamanho n_S x n_S com zeros.
    para cada caractere em S:
        dp[i][i] = 1
    para cada gap de 1 até n_S:
        j = i+ gap
        se s[i] == s[j]:
            dp[i][j] = dp[i+1][j-1]+2
        senao:
            dp[i][j] = max(dp[i+1][j], dp[i][j-1])

    o valor dp[0][n_S-1] vai ser o comprimento da maior
    subsequencia que forma um palindromo
```

O tempo de execução é O(n_S^2).