1. Palindrome substring.

(a) $T(i,j)$ indicates whether or not the substring starting at i and ending at j is a palindrome. If that substring is a palindrome, then $T(i,j)$ is the length of that palindrome (i.e., $j-i+1$). If the substring is not a palindrome, then $T(i,j)$ is zero.

(b)

$$T(i,j) = \begin{cases} j-i+1 & \text{if } T(i+1,j-1) > 0 \text{ and } x_i = x_j \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

(c)

Algorithm 1 Palindrome($x_1, x_2, \ldots, x_n$)

```
//Initialization
for i = 1 to n do
  T(i, i) ← 1
for i = 1 to n − 1 do
  if x_i == x_{i+1} then
    T(i, i + 1) ← 2
  else
    T(i, i + 1) ← 0
//Recursion
for s = 2 to n − 2 do
  for i = 1 to n − s do
    if T(i + 1, i + s − 1) > 0 and x_i == x_{i+s} then
      T(i, i + s) ← s + 1
    else
      T(i, i + s) ← 0
//Find the best in the table
Best ← 0
for i = 1 to n do
  for j = i to n do
    if Best < T(i, j) then
      Best ← T(i, j)
return Best
```

(d) There are $O(n^2)$ iterations, each of which takes constant time. It takes $O(n^2)$ time to search for the best in the table

2. Maximum Product.

(a) $T(i,k)$ is the maximum product achievable given the prefix $z_1 z_2 ... z_i$ and using exactly k multiplications.

(b)

$$T(i,k) = \max_{k \leq \ell < i} \{T(\ell, k-1) \times (z_{\ell+1} \dots z_i)\} \tag{2}$$

(c)

Algorithm 2 MaxProd$(z_1, z_2, \dots, z_n, k)$

```
//Initialization
for i = 1 to n do
  T(i, 0) ← z_1 ... z_i //i.e., the number represented by this string
//Recursion
for k' = 1 to k do
  for i = k' + 1 to n do
    T(i, k') ← max_{k' ≤ ℓ < i} {T(ℓ, k' − 1) × (z_{ℓ+1} ... z_i)}
return T(n, k)
```

(d) The loop has $O(nk)$ iterations, and each iteration takes $O(n)$ time, for a total of $O(n^2 k)$ time.

3. Coin changing variant.

(a) $T(i,b)$ is the minimum number of coins needed to make change for b (the table is initially all infinity).

(b)

$$T(i,b) = \begin{cases} \min\{T(i-1,b), T(i-1,b-x_i)+1\} & x_i \leq b \\ T(i-1,b) & x_i > b \end{cases}$$

(c)

Algorithm 3 Change($x_1, x_2, \ldots, x_n, B, k$)

```
//Initialization
Initialize the whole table to positive infinity.
//Base case
T(0,0) ← 0
//Recursion
for i = 1 to n do
  for b = 1 to B do
    if x_i ≤ b then
      T(i,b) ← min{T(i−1,b), T(i−1,b−x_i)+1}
    else
      T(i,b) ← T(i−1,b)
return True iff T(n,B) <= k
```

(d) There are nB iterations, each of which takes constant time, so the total time is $O(nB)$.

Note: A slower alternate solution defines T as a three-dimensional table.

Slower Solution:

(a) $T(i, b, k')$ is TRUE if we can make change for the value b using at most k' coins out of the coins $\{x_1, ..., x_i\}$, otherwise $T(i, b, k')$ is FALSE.
(b) Then the recursion is:

$$T(i, b, k') = \begin{cases} T(i-1, b, k') \text{ OR } T(i-1, b-x_i, k'-1) & x_i \leq b \\ T(i-1, b, k') & x_i > b \end{cases}$$

(c)

Algorithm 4 Change($x_1, x_2, \ldots, x_n, B, k$)

```
//Initialization
Initialize the whole table to FALSE.
//Base case
for i = 0 to n do
  for k' = 0 to k do
    T(i, 0, k') ← TRUE
//Recursion
for i = 1 to n do
  for b = 1 to B do
    for k' = 1 to k do
      if x_i ≤ b then
        T(i, b, k') ← T(i − 1, b, k') OR T(i − 1, b − x_i, k' − 1)
      else
        T(i, b, k') ← T(i − 1, b, k')
return T(n, B, k)
```

(d) There are nBk iterations, each of which takes constant time, so the total time is $O(nBk)$.