Problem 1:

Idea:
Compare letters in Y with letters in X one by one. The order of letters in Y cannot be changed when match with letters in X.

Pseudocode:
Let X[n] represents sequence X;
Let Y[m] represents sequence Y;
if m>n
    return false;
index=0;
for i=1 to n
    if y[index]== x[i]
        index++;
    if index>m
        return true;
return false;

Time complexity is O(n).

Problem 2:

Idea:
Use greedy algorithm. Sort competitors by sum of running time and biking time in descending order. The competitor with larger running and biking time starts first.

Pseudocode:
Let S[n],R[n],B[n] represent swimming, running and biking time of n competitors;
list= Quicksort(R[n]+ B[n]) in descending order;
return list;

Time complexity is O(nlogn).

Proof:
Consider two competitors A and B:
Ta=Sa+Ra+Ba;
Tb=Sb+Rb+Bb;
And (Ra+Ba)>(Rb+Bb)
Case 1: If competitor A starts first, then total time for A and B is
If (Ra+Ba)>(Sb+ Rb+Bb)
    T1= Sa+ Ra+Ba;
else
    T1= Sa+ Sb+ Rb+Bb;
Case 2: If competitor B starts first, then total time for A and B is
T2= Sb+ Sa+ Ra+Ba;
We can see T2>T1.
So competitor with larger (running and biking) time should go first.

**Problem 3:**

**Idea:**
Use dynamic programming.
Denote total payment of i weeks when choosing ‘stressful’ or ‘ordinary’ or ‘nothing’ in i-th week by S[i], O[i], N[i] respectively. Then the recursive function is as follows:

- \( S[i] = N[i-1] + x_i; \)
- \( O[i] = \max\{S[i-1], O[i-1], N[i-1]\} + y_i; \)
- \( N[i] = \max\{S[i-1], O[i-1], N[i-1]\}; \)

The three functions should be saved separately.

**Pseudocode:**

\[
S[1] = x_1, \\
O[1] = y_1; \\
N[1] = 0;
\]

For \( i=2 \) to \( n \)
\[
S[i] = N[i-1] + x_i; \\
O[i] = \max\{S[i-1], O[i-1], N[i-1]\} + y_i; \\
N[i] = \max\{S[i-1], O[i-1], N[i-1]\};
\]

The maximum payment for \( n \) weeks is
\[
\max\{S[n], O[n], N[n]\}
\]

**Time complexity** is \( O(n) \).