Q6.1

6.1. *Subproblems:* Define an array of subproblems $D(i)$ for $0 \leq i \leq n$. $D(i)$ will be the largest sum of a (possibly empty) contiguous subsequence ending exactly at position i.

Algorithm and Recursion: The algorithm will initialize $D(0) = 0$ and update the $D(i)$'s in ascending order according to the rule:

$$D(i) = \max\{0, D(i-1) + a_i\}$$

The largest sum is then given by the maximum element $D(i)^*$ in the array D. The contiguous subsequence of maximum sum will terminate at i^*. Its beginning will be at the first index $j \leq i^*$ such that $D(j-1) = 0$, as this implies that extending the sequence before j will only decrease its sum.

Correctness: The contiguous subsequence of largest sum ending at i will either be empty or contain a_i. In the first case, the value of the sum will be 0. In the second case, it will be the sum of a_i and the best sum we can get ending at $i-1$, i.e. $D(i-1) + a_i$. Because we are looking for the largest sum, $D(i)$ will be the maximum of these two possibilities.

Running Time: The running time for this algorithm is $O(n)$, as we have n subproblems and the solution of each can be computed in constant time. Moreover, the identification of the optimal subsequence only requires a single $O(n)$ time pass through the array D.

Q6.4

6.4. a) *Subproblems:* Define an array of subproblems $S(i)$ for $0 \leq i \leq n$ where $S(i)$ is 1 if $s[1 \cdots i]$ is a sequence of valid words and is 0 otherwise.

Algorithm and Recursion: It is sufficient to initialize $S(0) = 1$ and update the values $S(i)$ in ascending order according to the recursion

$$S(i) = \max_{0 \leq j < i} \{S(j) : \texttt{dict}(s[j+1 \cdots i]) = \texttt{true}\}$$

Then, the string s can be reconstructed as a sequence of valid words if and only if $S(n) = 1$.

Correctness and Running Time: Consider $s[1 \cdots i]$. If it is a sequence of valid words, there is a last word $s[j \cdots i]$, which is valid, and such that $S(j) = 1$ and the update will cause $S(i)$ to be set to 1. Otherwise, for any valid word $S[j \cdots i]$, $S(j)$ must be 0 and $S(i)$ will also be set to 0. This runs in time $O(n^2)$ as there are n subproblems, each of which takes time $O(n)$ to be updated with the solution obtained from smaller subproblems.

b) Every time a $S(i)$ is updated to 1 keep track of the previous item $S(j)$ which caused the update of $S(i)$ because $s[j+1 \cdots i]$ was a valid word. At termination, if $S(n) = 1$, trace back the series of updates to recover the partition in words. This only adds a constant amount of work at each subproblem and a $O(n)$ time pass over the array at the end. Hence, the running time remains $O(n^2)$.

Q6.17, 6.19

6.17. This problem reduces to Knapsack with repetitions. The total capacity is v and there is an item i of value x_i and weight x_i for each coin denomination. It is possible to make change for value v if and only if the maximum value we can fit in v is v. The running time is $O(nv)$.

6.18. Use the same reduction as in 6.17, but reduce to Knapsack without repetition. The running time is $O(nv)$.

6.19. This is similar to 6.17 and 6.18. The problem reduces to Knapsack without repetition with a capacity of v, but this time we have k items of value x_i and weight x_i for each coin denomination x_i, i.e. a total of kn items. The running time is $O(nkv)$.