6.1 Given an array $a_1, \cdots, a_n$, to compute the longest contiguous subsequence sum, we first consider the best sum, s_b and the current sum at that variable, s_c, and set them respectively to $s_b = -\infty, s_c = 0$, additionally we define the start and end indices of the ideal sum to be and the current sum as i_s, i_e, c_s, c_e. Then we iterate through the array, where for each $i \in [n]$, we first check if $a_i > a_i + s_c$, if this statement is true we set $c_s = i$ and $s_c = a_i$. If $a_i = a_i + s_c$ we set $c_s = i$. If $a_i < a_i + s_c$ we set $s_c = s_c + a_i$ and $c_e = i$. Next, if $s_c \geq s_b$ is true then we set $s_b = s_c$ and $i_e = i, i_s = c_s$. Note the algorithm mentioned before trivially runs in $O(n)$ time, and it will arrive at the best solution because in every moment it trades up to the maximum current sum/element, which will identify when we need to look at a new contiguous array, then after it will swap up to the best sum.

6.2 To figure out the best journey, predicated on for x miles driven in a day $(200 - x)^2$ to be the penalty. For the array of $a_1, \cdots, a_n$ hotel distances, we will define $a_0 = 0$, and the arrays $dp, prev$, where $dp[i]$ is the optimal penalty up to hotel i, and $prev[i]$ is the optimal last hotel. We start by setting $prev[0] = 0$ and $dp[0] = 0$. Then for each $i \in [n]$, we compute $dp[i] = \min(dp[1] + (200 - (a_i - a_1))^2, dp[2] + (200 - (a_i - a_2))^2, \cdots, dp[i-1] + (200 - (a_i - a_{i-1}))^2)$, then store $prev[i]$ based on which hotel was picked previously. This will find the least penalty to get to hotel i. By returning the $prev$ array and dp we can do $dp[n]$ to get the best penalty to hotel n and then all the path took via looking up indicies of the last hotel in prev recursively.

6.3 Similarly as before, one has the dp array which is maximum profit including the i-th location. This is done via first setting $dp[1] = \max_{i \in \mathbb{N}} p_i$, then iterating through for each $i \in [n] \backslash \{1\}$ computing $dp[i] = \max(p_i, \{p_j + dp[j] : m_j - m_i > k, j \in [n] \backslash [i]\})$, where the previous most profitable location is stored in a seperate prev array. Finally one returns the $prev$ and dp arrays, where one can extract the most profitable wackdonalds locations for n potential locations and their positions via $dp[n]$ and working backwards through the $prev$ array.