1 Algorithm Analysis

The problem prompted in this assignment is to design an algorithm that returns the optimal path from castle a_0 to a_n where there are n is the number of castles. The problem is that the castles can be any arbitrary distance apart but we must travel exactly m miles per day or suffer a penalty of $(x - m)^2$ where x is the miles traveled in a particular day.

To solve this problem we must first store the optimal miles per day as a variable and store an integer array of mile-markers representing the locations of the castles. Then we must construct two new arrays: an array to store the minimal penalties and an array to store the optimal path. We initialize the minimal penalty array at each index i to be $(distances[i] - m)^2$ where *distances* is an array storing the locations of each castle and m is the optimal distance to travel each day. Since we are starting at index 0, our penalties array will give us the minimum penalty at index 1 since it adjacent to our starting index. Any index after 1 in the penalties array will likely not be a minimum because we are taking the distance from our starting castle to a non-adjacent castle. After initializing all these arrays, we must iterate through the distances array and for any index k in the array, we must compare the penalty from k to every other index in the array and find the minimum among that group. This takes the form of a doubly nested for-loop. In our first loop, we iterate from 1 to n. In our nested for-loop, we iterate backwards from i to 0. At each step in the inner for-loop, we must construct the penalty by subtracting the value of our current mile-marker in our outer for-loop from the current mile-marker in our inner for loop. Then we subtract our optimal distances per day from this value and square the result. This, in turn, will give the penalties from the current index of our outer loop to each index preceding it. We then check if any of these new penalties we construct is less than the current penalty of our penalty array. If true, we will update our penalty array at the index of our outer loop to be the currently penalty. Additionally, we update the optimal path at the index of our outer loop to be the index of our inner loop, because that is the index in which we found the minimum penalty. We must then iterate through the optimal path array and print the resulting optimal path. Lastly, our minimum penalty is stored as the last index of our penalty array because at that point, our algorithm has gone through all the combinations of paths and determined the final penalty.

To determine the run-time of this algorithm, we note that in our first loop we iterate from 1 to n, and in the inner for loop we iterate from 0 to i. This means that each iteration in our inner for-loop will look like $0, 1, 2, ..., i$ where $1 \leq i \leq n$. This gives $\sum_0^n i$ because we do this loop a total of n times. This sum gives the close form of $\frac{n(n+1)}{2}$ which can be distrubuted to get $\frac{n^2+n}{2}$. Since we are looking at the time complexity, we can drop the coefficients and look at the highest term polynomial. Doing so results in a run-time of $\Theta(n^2)$.