1 Algorithm Analysis

The problem prompted in this assignment is to design an algorithm that returns the optimal path through a truffle field, subject to constrained movement through the field. In addition, we are given with the initial knowledge of the number of truffles at each square meter of the field.

To solve this problem we must first read the text file and store the number of truffles in each square meter as a 2-D array. From here, we will need a means of calculating our movement set. We can go down to the left, down the middle, or down to the right. Given this, we can construct three methods: getLeftChild, getMiddleChild, and getRightChild. All three of these methods will work as follows: given the initial 2-D array and an index in the array, these methods will return an integer array of size 3 that stores the number of truffles, the row, and the column of the respective children. If the child does not exist (out of bounds of the 2-D array), we return -1 for all values. Next, we can construct a getMax method that takes all the children as input and returns the child array with the maximum truffle count. In order to keep track of the path for each starting position, we make a $(2 \times width)$ array that stores the row of the path in the first row and the column of the path in the second row. Additionally, we need an integer array to store the truffles grabbed at each step.

From here, we start in the first row and the first column. We set the first path position to be at our starting position and set the first truffle count to be the number of truffles at our starting position. Additionally, we need two counters to keep track of our current row/column. We then call the getChildren/getMax methods to see which child has the largest truffle count, update our current position to that child, then add the information of that child into our arrays. We repeat this method for each column so that we may exhaust every starting position to find the optimal path. In order to keep track of which path is optimal, we can make a final method that calculates the sum of the truffle array and when a truffle array exceeds the previous maximum, we update the optimal truffle array and the optimal path array accordingly. When we have finished the algorithm, the largest truffle count will be stored, along with the optimal path of those truffles.

In this algorithm, given an $n \times m$ matrix, we iterate from 1 to n a total of m times. This is because we always start at the beginning of the field and stop at the end of the field, repeating for each starting position. In the case of a square matrix, this will result in a run-time of $\theta(n^2)$. In general, where the matrix is not square, we will have a lower-bounded run-time of $n \times m$ or $\Omega(n)$. This represents our lower bound because we could have a field with a very small width where we only iterate through n a few times.