**1. What was the complexity of your function in the Maze assignment that determines the**

**start position of the maze by searching the maze array for the starting ‘x’? Use n equal**

**to the maze width (i.e. the maze is n x n).**

O(N^2), it needs to do n times n operations to find the x. There are n^2 option to be at so its order is.

**2. What data structures did you use (array, vector, list, tree, …), and how did the complexity**

**of the operations influence your design decisions? (For maximum points, the answer to**

**this question should be part of your design decisions, and here you just refer to that…)**

Array for visited list and Vector for worklist. Since the structure of the array is such that we do not need to search for the value that we want(we want to search if a pixel was visited, and we find it by a specific index of our array). And for the vector we also do not need to search since we just use the last element in our algorithm.

**3. Explain the order of complexity for the scanning algorithm and the marching squares**

**algorithm, in terms of N for a screen size of N x N pixels.**

The order of the scanning algorithm is constant by N^2. Since we always must do that many operations. The order of the marching squares depends on the position of the field, and on the search algorithm, mostly. The algorithm for coloring in the contour is quite fast and depends on the size of the contour (which varies). Worst case scenario there is no vector field, or at the spot your search algorithm finds it last and then marching squares is basically as slow as scanning.

**Design choices:**

For the scanning algorithm the simple approach of looping through all pixels horizontally was chosen. If and only if one pixel is over the threshold, draw the pixel white. This was realized using two forloops.

For the worklist a vector was used and by scanning using a similar method as in the first assignment a pixel that is part of the contour is found and added to the worklist. Then for the visited-list, an array of Booleans was created that contains as many elements as pixels. Since every pixel maps to one element in the array, we set each element to true that has been visited. Also, the data structure map was used. But this turned out to be slower, because we must insert new elements all the time. This approach however uses more memory, also it needs to be initialized to be zero. Then by constantly checking if pixels are on the contour and adding their neighbors the worklist its constantly filled up until the entire contour is drawn and it becomes empty again since all pixel already have been visited.

For the fast and improved marching squares we do the same, but scan two times by from the other direction, since we know there are only two vector fields we will always encounter them or find the same field twice. To speed up the scanning we scan the field while skipping some horizontal line to speed up the scanning.

For the multithreaded scanning we make a maps of all pixel similar as used in the visited list, and then let N threads each scan a separate part of the field and they put their data in the list. Since each thread has its own area of the “screen” they never write to the same data. Then later the vector field is drawn, using only the main thread.