## Sysc 2100 – Assignment 4

The search method, first found the middle element of the two dimensional array and compared it with the desired item. This divides the two dimensional array into four subarrays: the top left, the top right, the bottom left, and the bottom right. This takes away only a quarter of the array away. If the desired item is less than the middle element, than I know for sure that the value is not in the right side of the 2D array. Thus I take the top left, top right starting from row zero and ending at row of current minus one, and bottom left subarrays and search through it recursively until the problem becomes a one by one array where I just compare the desired value to the value inside the one by one array.

If the desired item is greater than I do the same thing but take the right side of the 2D array. I take top right, bottom left and bottom right and search through it recursively until the problem becomes a one by one array where I just compare the desired value to the value inside the one by one array.

As I am searching, if I find the desired value I return true and quit, otherwise I continue and if and only if I have searched all three parts

Below is a table comparing the number of steps required to solve the search using an algorithm with O(n), binary search with O(log2n) and the search algorithm I implemented. The number of steps were calculated by using the formula pi – int(pi / k) and repeating it over and over again until a constant appeared as many times as the value of the constant in a row, each time taking subtracting int(pi/k) from previous pi, k was either a 2 for binary or a 4 for the search algorithm while the initial pi was n^2.

|  |  |  |  |
| --- | --- | --- | --- |
| n | # of steps O(n) | # of steps search() | # of steps O(log2n) |
| 2 | 2 | 4 | 3 |
| 3 | 3 | 8 | 5 |
| 4 | 4 | 10 | 5 |
| 5 | 5 | 12 | 6 |
| 10 | 10 | 17 | 8 |
| 15 | 15 | 20 | 10 |
| 20 | 20 | 22 | 10 |
| 30 | 30 | 25 | 11 |
| 40 | 40 | 27 | 12 |
| 50 | 50 | 28 | 13 |
| 60 | 60 | 29 | 13 |
| 100 | 100 | 33 | 15 |
| 200 | 200 | 38 | 17 |
| 350 | 350 | 42 | 18 |
| 500 | 500 | 44 | 19 |
| 750 | 750 | 47 | 21 |
| 1000 | 1000 | 49 | 21 |

The worst case time complexity of the search algorithm I implemented as shown by the table above is O(log2n) < f(x) < O(n), where f(x) is the time complexity of the search algorithm.