ASSIGNMENT QUESTIONS:

1. Write in pseudo code a binary search algorithm that works over an array. Use the iterative method. Comment to explain the algorithm.

The pseudo code is in the HW1\_pseudo code.py file.

2. Write in the programming language of your choice a binary search algorithm that works over an array. Comment to explain the algorithm.

The main code is in the HW1\_main code.py file.

3a. What is the position of 51216352 in the array? How many operations did it take to find?

index 499 with 2 operations.

3b. What is the position of 198313119 in the array? How many operations did it take to find?

index 1980 with 11 operations.

3c. What is the position of 196614208 in the array? How many operations did it take to find?

no index’s element matched 196614208 with 11 operations.

3d. What is the worst case time complexity? Evaluate line by line, create a time complexity function and then define its Big O value.

Check the HW1\_time complexity.py file. In general it is T(n) = O(log n) or the big O is O(log n). it is logarithmic time.

3e. What would the worst case time complexity be if we have 4000 entries instead of 2000?

The worst case for 2000 is 11 operations as log\_2(2000) = 11 but if we take log\_2(4000) it equals about 12 so the worst case would be 12 operations.

3f. What do you think the average case time complexity is for binary search? Explain your reasoning.

each time the loop is run its cutting the search 'area' in half causing a logarithmic increase in runtime so the average time to search an array with n index's is log\_2(n)

4a.

Check the HW1\_time complexity.py file. Its time complexity is linear time as the number of operations are equal to the length of the input array. Its big O value is then O(n). the math for this is T(n) = O(1) + O(1) + O(1) + O(1)\*n + O(1) => T(n) = O(1)\*n => T(n) = O(n).

4b.

Check the HW1\_time complexity.py file. Its time complexity is quadratic time as the number of operations is the length of the input array squared. It big O value is then O(n^2). The math for this is T(n) = O(1)\*n^2 + O(1)\*n + O(1) => T(n) = O(1)\*n^2 => T(n) = O(n^2).