Evan Zhang, z5383657

Question 2

2.1

Mergesort the array taking O(nlogn) time.

Starting at the smallest element, add it to every element including itself, taking O(n^2) time. For each of these results, compare it to the middle element.

Since the array is sorted, if the result is smaller than the middle element, recurse on the lower half.

If the result is bigger than the middle element, recurse on the higher half.

If the result is equal to the middle element, return the two integers added and the middle element.

If the result is not equal and the current problem array is n = 1, the result is not in the array and go to the next pair of elements to be added.

Since the array size is halved at each step, it will take O(logn) time to find whether the result is in the array. If no such indices exist, the time taken will be Θ (n^2) times Θ (logn) plus (nlogn), since we first mergesort, then add every element to every element and then take Θ (logn) time to see if the result exists. Therefore, it will take $\Theta(n^2 \log n)$ in the worst case.

2.2

A hashtable using chaining collision resolution has an average insertion complexity of O(1) and so creating one with n elements will take on average O(n) time. Average case for searching for an element will take O(1) time.

Therefore, using a hashtable as the search method instead of a sorted array in the algorithm in 2.1 will result in an average case of $O(n^2) + O(n)$, or $O(n^2)$ time.

2.3

Use a search method that takes O(1) search time in worst case. This can be done by using a hash map that causes no collisions to occur in the hash table. Worst case is when a hash maps every number to the same hash key. Therefore, use a hash map that