COMP3121: Assignment 1 – Q1

Gerald Huang
*z*5209342

Updated: June 4, 2020

a) Given an array A of size n, sort the elements of the array using merge sort which can be done in $O(n \log n)$ time. For each element in A, compute its square and re-store it in $A[i]$ for all $i \in \mathbb{N} \cup \{0\}$. For example,

$$A[2] = 4 \implies A[2] = 16$$

after computing its square. This process is done in $O(n)$ time. Since each element in A is now composed of the squares of the original array's elements, we only need to consider all of the possible tuples (i, j) and the result of $A[i] + A[j]$. We shall store each of these results in a new array B whose size is $n(n+1)/2$. To discount all of the possible swappings of (i, j) – that is, discarding all of the (j, i) tuples, we can set two indices i, j such that $i < j$. This ensures that we do not count any swaps.

We shall assume that storing our sum of squares is performed in $O(1)$ time. Thus, a new array B is formed in $O(n^2)$ time. However, this new array is unsorted. Fortunately, we can sort this array using merge sort which is $O(n^2 \log(n^2)$ time. But note that

$$n^2 \log(n^2) = 2n^2 \log(n) = O(n^2 \log n).$$

Finally, to determine whether a particular number can be written as a sum of two squares, we just need to iterate over the new sorted array B. If there is a duplicate result, then the algorithm will return true. Otherwise, return false. This is done in $O(n)$ time since we only need to check adjacent entries.

Hence the algorithm runs in $O(n^2 \log n)$ time.

b) Apply the same algorithm as in part (a); that is, sort the array in $O(n \log n)$ time, compute the squares in $O(n)$ time and create a new array B in $O(n^2)$ time. From here, insert each of the elements in its corresponding position on the hash table. If there is a clash, then the algorithm should return true; otherwise, keep inserting the elements until there are no elements left to insert. In this case, return false. Assuming that placing each of the elements into the hash is performed in constant time, this algorithm will run in $O(n^2)$ time since there will be *no more* than n^2 terms.

The **expected** time for the algorithm to execute will be when *no* number can be written as the sum of two squares in two different ways (from the elements in the original array A). This will be terminated after n^2 steps.