1. Recall the problem of finding the number of inversions: You are given a sequence of \( n \) numbers \( a_1, \ldots, a_n \), all distinct, and we define an inversion to be a pair \( i < j \) such that \( a_i > a_j \).

We motivated the problem of counting inversions as a good measure of how different two orderings are. However, one might feel that this measure is too sensitive. Let’s call a pair a significant inversion if \( i < j \) and \( a_i > 2a_j \). You will be designing and implementing an \( O(n \log n) \) algorithm to count the number of significant inversions between two orderings.

(a) Give pseudocode for an algorithm to find significant inversions, and explain briefly why it runs in \( O(n \log n) \) time.

(b) Using the starter code (SignificantInversions.java) on the course webpage, implement your algorithm. Also, for comparison, please implement the naive \( \theta(n^2) \) algorithm which does a comparison between \( a_i \) and \( a_j \) for each pair \( i < j \), and counts the number of significant inversions. In the starter code, it will be clear where to add this. Do not use any external libraries such as Collections.sort.

(c) Report the running times of both the \( \theta(n^2) \) and the \( \theta(n \log n) \) algorithm on the inputs generated by the program. These running times will be generated automatically by the program. Turn in a graph showing the running times of the two algorithms as a function of input size. (Note that this will be a bit tricky since the inputs are powers of 2.) Nonetheless, do the best you can to produce a graph that shows the difference in running time between the naive algorithm and the divide and conquer algorithm.

(d) Upload to canvas the final five numbers output by the program.

2. You are interested in analyzing some hard-to-obtain data from two separate databases. Each database contains \( n \) numerical values – so there are \( 2n \) values total – and you may assume that no two values are the same. You’d like to determine the median of this set of \( 2n \) values, which we will define here to be the \( n^{th} \) smallest value.

However, the only way you can access these values is through queries to the databases. In a single query, you can specify a value \( k \) to one of the two databases, and the chosen database will return the \( k^{th} \) smallest value it contains. Since queries are expensive, you’d like to compute the median using as few queries as possible.

Give an algorithm that finds the median value using at most \( O(\log n) \) queries.