Robert Owens (rao7utn)
Social Distancing (Basic HW Divide and Conquer)

This problem requires finding what the smallest spacing is between two occupied picnic tables given the number of people at the park, the number of tables at the park and the arrangement of those tables.

For my approach, I read in input to a list which stores the number associated with each picnic bench. Next, for the divide and conquer algorithm, I divide that spacing between people in the following way. A lower bound of 1, meaning the minimum possible spacing would-be placing people at picnic tables at least 1 level apart and an upper bound of the length of the park divided by the number of people -1 (The -1 s because you should always have someone at the first table). The idea now is the test (the 'walk' function see below) by preforming a binary search between the upper and lower bounds. These bounds form a range of possible integers that could be the minimum distance, where the lower bound is inclusive (will work) and the upper bound is exclusive (will not work). Take the average of the two bounds, test it; if that average is valid (walk function returns true) then set the lower bound equal to the average of the two however if the test fails, set the upper bound equal to the average. When the upper bound and lower bound are with 1 of each other print the lower bound as the solution (since the lower bound must work not necessarily true of upper bound)

I tested whether or not a certain bound would work using my walk method which had 1 parameter, step. Step was the minimum possible distance between tables. Hence, in walk keep a counter of how many people *can* be placed in the park with this given step, then iterate through the list of the park and if the current table minus the previous table where you could place someone is greater than or equal to the step, increment the counter and set this picnic table equal to the pervious table, otherwise continue through he loop. Finally return whether or not the counter is greater than or equal to the number of people in the park.

This algorithm runs in Theta(nlogn) time complexity since the walk function will iterate through in Theta(n) time, and the finding bounds computation will operate in Theta(logn) because it is continuously cutting the universe of possible integer solutions in half, much like binary search. Since the finding bounds computation calls the walk function inside of it, we arrive at Theta(logn)*Theta(n) = Theta(nlogn).

The grader may look at the file with my code so no pseudocode is required here.