Problem 1. Comparing Sorting Algorithms
Consider the following sorting algorithms: bubble, selection, insertion, heap, BST (unbalanced), BST (splay), and merge. For each of the following inputs, state the \( \Theta \) (tight bound) complexity for every one of the above algorithms. Assume in each case that the array contains \( N \) items. You do not need to show your work, but it may be helpful for partial credit.

(a) input is sorted already (smallest to largest).

(b) input is reverse sorted (largest to smallest).

Problem 2. Nonrecursive Mergesort
Mergesort can be done without using recursion. As noted in lecture, non-recursive mergesort should merge into the auxiliary buffer and then into the original array and so on. Write the corresponding algorithm in pseudocode. Your method should alternate between merging into the auxiliary array and merging into the original array, and should avoid excessive copying. For the actual merging step, you may use the merge method shown in the text; however, you must note any changes needed to make it work with your algorithm. (If these changes are extensive, then you should just write out your new version of the merge method.)

Note: please write pseudocode that actually resembles code (e.g., Java) – it doesn’t need to compile, but it should be easy for a programmer to read. And, a gentle reminder: looking up the answer to this on the web is not allowed.

Problem 3. Some searching, some sorting, and some sums
Suppose you are given as input \( n \) positive integers and a number \( k \). Our goal is to develop an algorithm to determine if there are any four of them, repetitions allowed, that sum to \( k \). As an example, if \( n = 7 \), the input numbers are 6, 1, 7, 12, 5, 2, 14 and \( k = 15 \), the answer should be YES because \( 6 + 5 + 2 + 2 = 15 \).

(a) First solve the simpler problem of determining if there are any two numbers in a list that sum to \( k \) (repetitions allowed). Describe, in words, an algorithm that will achieve this goal in \( O(n \log n) \) worst case time. Explain how your algorithm achieves this complexity bound.

(b) Now observe that the sum of four numbers is the sum of two pairs of numbers. Carefully explain how we can solve the original problem (i.e., whether a four number sum exists). For full credit, show that your algorithm runs in worst case time \( O(n^2 \log n) \).