APS Homework 3: Sorting

Problem 1: Target Sum

You are given a list of n real numbers along with a target number x. You want to determine whether there exist 2 distinct elements in your list that sum to x. A naive approach would be to check every possible pair of elements to see if they sum to x, but we can do better.

Problem 1a: In Big-O notation, what is the worst-case time complexity of the naive algorithm, in which you check every possible pair of elements in the list?

Problem 1b: If you were to first sort the list, you can design a more efficient algorithm. Describe an algorithm that, given a *sorted* list, can find a pair of elements that sum to x more efficiently than the naive algorithm.

Problem 1c: In Big-O notation, what is the worst-case time complexity of the sorting step that must precede your algorithm? What about your algorithm itself (after the list has been sorted)?

Problem 2: Comparison of Time Complexities

For each of the functions $f(n)$ in the table below (i.e., the rows), determine the largest problem size n that can be solved in the specified time t (i.e., the columns), assuming the algorithm takes exactly $f(n)$ nanoseconds to run. Hint: Instead of computing them by hand, it may be faster to write a program to compute them for you.

	1 second	**1 minute**	**1 hour**	**1 day**	**1 year**	**1 century**
$\log_2 n$						
$\log_{10} n$						
$\sqrt{n}$						
n						
n^2						
n^3						
n^{50}						
2^n						
$n!$						