Problem 1. We would like to take into account *memory hierarchies* when analyzing Insertion Sort. Assume that your machine takes time $f(i)$ to access memory location i, for some nondecreasing function $f(i)$. Most of the time, when doing algorithmic analysis, we assume $f(i)$ is a constant. In reality it is a slow growing function. For this problem, to keep things simple, we will assume that $f(i)$ grows linearly, which is pretty drastic. Also, to keep things simple, we will only count the time for comparisons. To compare two elements in locations i and j takes time $i + j$.

We modify Insertion Sort to take the memory access cost into account. The array to be sorted is in $A[1, \ldots, n]$. To insert the ith element of A into its proper location, put it into $A[0]$, where it is cheap to access. Then compare the elements of A to this value starting at location 1 and going up, rather starting at location i - 1 and going down. (Note that if we were going to be really efficient, we could move an element into location 1 before comparing it the element being inserted, so the cost would be just $0 + 1 = 1$. This is not in the spirit of the problem. If you must do this, then, to be fair, you should also keep track of the cost of moving items.)

(a) Write the pseudo-code for this modified version of Insertion Sort. Do not use a sentinel.

(b) What input gives the best-case cost?

(c) Analyze the exact best-case cost. Show your work.

(d) What input gives the worst-case cost?

(e) Analyze the exact worst-case cost. Show your work.

(f) Analyze the average-case cost for $n = 3$. Show your work.

Problem 2. A company wants to determine the highest floor of its n- story building from which a gadget can fall with no impact on the gadgets functionality. The company has two identical gadgets to experiment with. Design an efficient algorithm to solve this problem. There is not need to write the pseudo-code. Explanation in simple English with clear steps would be sufficient. Report the asymptotic runtime of the most efficient algorithm.

Problem 3. Consider the following "comparison" model. There are n red and n blue water jugs for $n \geq 1$. All jugs are of different size and hold different amounts of water. For every red jug, there is a blue jug that holds the same amount of water, and vice versa.

Your task is to find a grouping of the jugs into pairs of red and blue jugs that hold the same amount of water. To do so, you may perform the following operation: pick a pair of jugs in which one is red and one is blue, fill the red jug with water, and then pour the water into the blue jug. This operation will tell you whether the red or blue jug can hold more water, or that they have the same volume. Assume that such a comparison takes one time unit. Your job is to find an algorithm that makes a minimum number of comparisons to determine the grouping. Remember that you may not directly compare two red jugs or two blue jugs.

Write pseudo code for an efficient brute force algorithm that sorts all the n pairs without doing any unnecessary comparisons. Show the exact worst-case analysis of the runtime for this algorithm.