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.)

Problem 1. Write the pseudo-code for this modified version of Insertion Sort. Do not use a sentinel.

Problem 2. What input gives the best-case cost?

Problem 3. Analyze the exact best-case cost. Show your work.

Problem 4. What input gives the worst-case cost?

Problem 5. Analyze the exact worst-case cost. Show your work.

Problem 6. Analyze the average-case cost for $n = 3$. Show your work.

Problem 7. **Challenge problem, will not be graded.** Analyze the average-case cost (for general n)?