a) I concur with Alice. Line(6) is within the inner-most for loop of the algorithm, and it's the "hot spot" in the algorithm that meaningfully characterizes its running-time. In this case, Alice's mindset causes us to gloss over the time it takes to execute Line(6), which has to do with the size of A[k-1], which is a component of the size of the input. So, the number of executions of Line(6) has nothing to do with the size of A[k-1], for any k. Rather, it's a function of n, the number of entries in the array, only.

b) I do agree. So, just characterize the number of executions of Line(6). We can observe that the outer **foreach** loop runs for n-1 iterations. The second **foreach** loop runs for i-1 iterations. And the inner-most **foreach** loop runs for i-j iterations, starting at i, down to j+1, only if the if evaluates to true at Line(3), and this is true for at most one value of j. For the outer **foreach** loop, i is incremented by 1 in each iteration. For the second **foreach** loop, j is incremented by 1 in each iteration, and for the inner-most **foreach** loop, k is incremented by 1 in each iteration. Thus, the maximum number of times Line(6) is executed is:

$$\sum_{i=2}^{n}\sum_{j=1}^{i-1}\sum_{k=j+1}^{i} 1 = \sum_{i=2}^{n}\sum_{j=1}^{i-1}(i-j)$$

$$= \sum_{i=2}^{n} i(i-1) - \sum_{i=2}^{n}\frac{i(i-1)}{2}$$

$$= \left(\frac{n^3-n}{3}\right) - \left(\frac{n^3-n}{6}\right)$$

$$= \frac{n^3}{6} - \frac{n}{6}$$

Thus, the running time of this algorithm can be characterized as $O(n^3)$. To prove the worst-case running time is $\Theta(n^3)$, suppose $T(n)$ is the worst-case running-time of this algorithm. We have shown that $T(n) = O(n^3)$. It remains to be shown that $T(n) = \Omega(n^3)$. That is, in the worst-case, the number of times Line(6) runs is lower-bounded asymptotically by n^3. This can be proved by construction, by producing an input array of n entries for which we are guaranteed that Line(6) is guaranteed to run i-j times, and no fewer, for each value of i. And this is the case when j is minimized, so that the number of times the inner-most foreach loop of Line(5) would be maximized. And this is exactly the case if A comprises distinct entries that are sorted in reverse. Thus, $T(n) = \Omega(n^3)$. Hence, the number of times Line(6) is executed in the worse-case is $\Theta(n^3)$.

c) So we assume the underlying computer uses base-2 for its arithmetic, and we assume that each basic operation takes 1 time-unit. So we need to consider: n, the number of entries in the array A, which is bounded by a constant, such that $1 \leq n \leq 10^6$. And the size of each entry of A[·]. When we perform assignments, e.g., *tmp* $\leftarrow$ *A[i], A[k]* $\leftarrow$ *A[k-1], A[j]* $\leftarrow$ *tmp.* It's reasonable to say that for *tmp* $\leftarrow$ *A[i]*, the assignment takes time |A[i]| for A[i] being |A[i]| bits long. Similarly, when we compare A[i] < A[j], the worst-case time is something like 1+min{|A[i]|, |A[j]|}, because a natural algorithm is to compare the two values bit-by-bit, and if all the bits of the two that has fewer bits are the same, we have an additional step to return **true** or **false**.

Suppose the maximum size to encode any of the entries in the input array A is s bits. So, then comparison in Line(3), and the assignments in Line(4), (6), and (7) would all take *s* bits. In the worst case, the outer **foreach** loop runs n-1 times, the second **foreach** loop runs i-1 times, the inner-most **foreach** loop gets to run for i-j times only if the if statement evaluates to be true, and this is true for at most one j. For the worst-case, we need to minimize j, as that would maximize the number of iterations for the **foreach** loop on Line(5), and this is for j=1, and in the worst case, this **foreach** loop would run for every i value, and this would make the time efficiency for this **foreach** loop $O(n^2)$. Now, suppose we make the simplifying assumption that, for each of the **foreach** loop, every assignment is for the maximum size i , j, and k could possibly have. As we assume binary encoding, this is then $1 + \lfloor log_2 10^6 \rfloor$.

Thus, the worst-case running-time is

$(n-1)[\,1 + \lfloor log_2 10^6 \rfloor + (i-1)[\,1 + \lfloor log_2 10^6 \rfloor + (s+1) + 2s + (i-j)[\,1 + \lfloor log_2 10^6 \rfloor + s]]]$.

With these been said, a meaningful characterization of the worst-case time-efficiency can be represented by the inner-most foreach loop on Line(5) and (6). So the worst-case time-efficiency could be represented as $O(n^2 s)$. But since n is bounded, and it's a finite constant, the worst-case running time, then, can be shown as $\Theta(s)$.