A useful operation in databases is the ***natural join***. If we view a database as a list of *ordered*pairs of objects, then the natural join of databases *A* and *B*is the list of all ordered triples (*x*,*y*, *z*) such that the pair (*x*,*y*) is in

*A*and the pair (*y*, *z*) is in *B*. Describe and analyse an efficient algorithm for computing the natural join of a list *A*of *n*pairs and a list *B*of *m*pairs.

Given that A and B have already been pre-sorted for us, we can use algorithms to take this into account and move from one side of our list to the other. We can build up two counters, one that iterates through the items of a and one that iterates through the elements of B, to accomplish this rather than simply verifying every element each time. Every time I reach the size it is now, the length of A is being reached. We stop, and you complete B with the remaining components. Have you reached the conclusion of the first, too? then we move forward. Any remaining parts of A after jay reaches length B. We can start at the beginning of our list and add to it. Both counts are pointing at the first element, or the element that is less important. Here, BV is less than zero. We proceed and include that component in her to-return list. And add one to the J. counter. By continuing to compare a zero and a one in this manner, we may see that one is less than three. Therefore, we'll go ahead and include the one in our list before increasing the I counter. Once more, we'll need to add to the list in three. The I counter is increased by three. Is that three of us and five? If so, add three and increase J. Now we have components that match. Therefore, in this specific instance, the answer would be A 2 and B 2. Therefore, for each of these, we just went forward by one without adding the element. However, take note that just A I and B J need to be on an equal footing; I N J need not. Because when we get to this portion here, we can see that six is going to be less than seven, so we add the sixth increment J, but in this case we're going to implement both by one and not add the seventh to the list and we keep going and continuing until either I reaches length A or J reaches length B. Because that indicates that we have reached the end of one of our lists, we proceed by simply adding counters one at a time, based on the lesser of the two elements, until we reach the end of either one or both of our lists.