Hash Version of the Algorithm:

1. New version of the Algorithm called Hash version of the Join is discussed, this involves the same approach as previous one like loading the fragments with disk blocks, initialize the Hash for the relation, Scan the blocks to load the fragments, scan the tuples in the respective blocks and insert it into the hash for the relation.
2. Nextly, the block of the other relation is loaded and for each tuple in it we perform the lookup activity with the Hash. If any match is found, then we concatenate and return it as output and release the Hash.
3. This version of the algorithm enhances the usage of the CPU efficiently.
4. We can also Achieve the parallelism with this approach unless the Hash is changed.

Indexed Based Nested Loop Join:

5. A key observation we can make with the previous approach is that, Hashing and Indexing is same thing performed on the relation.
6. We intend to build an External Index on the relation and use it instead.
7. The process goes same as the Index is built on top of the relation and each tuple is scanned and index lookup is done to find the matching instance and return it.
8. The Index is a pre-built and must be maintained, as it is expensive and necessary actions to use it efficiently. Any index occupies some space on the disk.
9. Several scenarios where the building hash on the fly or choosing index cannot be detailed out. As to answer it we need the low-level Query Cost Estimation which helps to identify which one is better.

Sort Based Algorithm:

10. There are scenarios where the data where we need to work can scale up to a large extent, and we can use the external algorithm which works irrespective of how big the data is and even sufficient memory is not available.
11. As a part of the Classical theory for Rational Databases, there are two classes of algorithms exists for external algorithms they are opposite to each other.
    a. Distributive Sort Based
    b. External Sort
12. Sort helps us find the chunks of the data which has similar key values in it. This eases the lookup of the data with the other relations. Basically, the data inside a relation is sorted out either in an ascending or the descending order to perform the necessary operations. Also, we need both the relations which are involving the operations needed to be sorted out.

Sort Join:

13. We have two relations R and S which are joined together on an element R.a = S.a, we sort both the relations R and S to perform the above operation. This creates the blocks of data that confines the interaction between the relations.
14. The whole idea revolves around detecting the blocks in sorted order and use them to determine the interactions among them.
15. This also demands the simultaneous scanning both the arguments with the coordinated scans.
16. Sort based on distinct where we consider all the attributes and with this all the duplicates end up sorted together.
17. Sort Based on group, in which the sort of key is the set of attributes, and all the members show up consecutively in the sorted order.
18. Next, Sort based intersection, where all the attributes of relations R and S are sort keys, and also assume there are no duplicates and if there are any, we can run the duplicate elimination operator to remove them.

External Merge Sort:

19. One advantage of the external sort is we can handle the large volumes of the data for the performing the various operations. It is a technique in which the data is stored in intermediate files and then each intermediate files are sorted independently and then combine to get sorted data.
20. This sorting algorithm takes M disk blocks.

```
sort A[i;j]
L = sort A[i:j/2]
R = sort A [j/2+1, j]
Return merge (L, R)
```

21. Two ways to merge lists, it takes $B_R passes to merge the data. We can parallelize the process using in-memory and using the single-thread algorithm.