Suggest up to three different algorithms for computing the join of two tables T1 and T2. Contrast the algorithms in terms of their time complexity and storage requirements.

POSSIBLE ANSWERS:

Algorithm 1: simple nested-loop join

```
for each row R1 in table T1:
  for each row R2 in table T2:
    if R1 and R2 satisfy the joining condition then combine
    R1 and R2 and append to the result table
```

Time complexity: O(|T1|*|T2|)
Storage requirement: essentially none

Algorithm 2: single-sort join

```
sort table T1 on the joining value;
for each row R2 in T2:
  use binary search on sorted T1 to find all matching values and
  add to the result table
```

Time complexity: O(|T1|*log(|T1|)) to sort T1
O(|T2|*log(|T1|) for second step

Algorithm 3: sort-merge join Sort both

```
sort table T1 on the joining value;
sort table T2 on the joining value;
traverse the two tables linearly (with some "backtracking" for
  duplicate values), matching join values and adding joining tuples
  to the result table
```

Time complexity: O(|T1|*log(|T1|)) to sort T1
O(|T2|*log(|T2|)) to sort T2
O(max(|T1|,|T2|) for "merge" phase assuming not too many duplicate joining values
Storage requirement: not much, depends on sorting algorithm used

Algorithm : hash join

```
set up hash table;
for each row R1 in T1:
  hash R1's join value and put R1 in the appropriate hash bucket
for each row R2 in T2:
  hash R2's join value;
  find all matching tuples in the hash bucket and add to the result table
```

Time complexity: |T1|+|T2| assuming well-distributed hash table
Storage requirement: O(|T1|) for hash table