per page, which I just assumed to be 10). Thus, an unclustered index would not be cheaper. Whether or not to use a hash index would depend on whether the index is clustered. If so, the hash index would probably be cheaper.

(b) Using the clustered B+ tree on *ename* would be cheaper than sorting, in that the cost of using the B+ tree would be 12,500 I/Os. Since *ename* is a candidate key, no duplicate checking need be done for $< title, ename >$ pairs. An unclustered index would require 2500 (scan of index) + 10000 * tuples per page I/Os and thus probably be more expensive than sorting.

(c) Using a clustered B+ tree index on $< ename, title >$ would also be more cost-effective than sorting. An unclustered B+ tree over the same attributes would allow an index-only scan, and would thus be just as economical as the clustered index. This method (both by clustered and unclustered) would cost around 5000 I/O's.

4. Knowing that duplicate elimination is not required, we can simply scan the relation and discard unwanted fields for each tuple. This is the best strategy except in the case that an index (clustered or unclustered) on $< ename, title >$ is available; in this case, we can do an index-only scan. (Note that even with `DISTINCT` specified, no duplicates are actually present int he answer because *ename* is a candidate key. However, a typical optimizer is not likely to recognize this and omit the duplicate elimination step.)

Exercise 12.4 Consider the join $R \bowtie_{R.a=S.b} S$, given the following information about the relations to be joined. The cost metric is the number of page I/Os, unless otherwise noted, and the cost of writing out the result should be uniformly ignored.

Relation R contains 10,000 tuples and has 10 tuples/page.
Relation S contains 2,000 tuples and also has 10 tuples/page.
Attribute b of relation S is the primary key for S.
Both relations are stored as simple heap files.
Neither relation has any indexes built on it.
52 buffer pages are available.

1. What is the cost of joining R and S using the page-oriented Simple Nested Loops algorithm? What is the minimum number of buffer pages required in order for this cost to remain unchanged?

2. What is the cost of joining R and S using the Block Nested Loops algorithm? What is the minimum number of buffer pages required in order for this cost to remain unchanged?

3. What is the cost of joining R and S using the Sort-Merge Join algorithm? What is the minimum number of buffer pages required in order for this cost to remain unchanged?

4. What is the cost of joining R and S using the Hash-Join algorithm? What is the minimum number of buffer pages required in order for this cost to remain unchanged?

5. What would the lowest possible I/O cost be for joining R and S using *any* join algorithm, and how much buffer space would be needed to achieve this cost? Explain briefly.

6. How many tuples will the join of R and S produce, at most, and how many pages would be required to store the result of the join back on disk?

7. Would your answers to any of the previous questions in this exercise change if you are told that $R.a$ is a foreign key that refers to $S.b$?

Answer 12.4 Let $M = 1000$ be the number of pages in R, $N = 200$ be the number of pages in S, and $B = 52$ be the number of buffer pages available.

1. Basic idea is to read each page of the outer relation, and for each page scan the inner relation for matching tuples. Total cost would be

$$\#pagesinouter + (\#pagesinouter * \#pagesininner)$$

which is minimized by having the smaller relation be the outer relation.

$$TotalCost = N + (N * M) = 200,200$$

The minimum number of buffer pages for this cost is 3.

2. This time read the outer relation in *blocks*, and for each block scan the inner relation for matching tuples. So the outer relation is still read once, but the inner relation is scanned only once for each outer block, of which there are $\lceil \frac{\#pagesinouter}{B-2} \rceil = \lceil 200/50 \rceil = 4$.

$$TotalCost = N + M * \lceil \frac{N}{B-2} \rceil = 4,200$$

If the number of buffer pages is less than 52, the number of scans of the inner would be more than 4 since $= \lceil 200/49 \rceil$ is 5. The minimum number of buffer pages for this cost is therefore 52.

3. Since $B > \sqrt{M} > \sqrt{N}$ we can use the refinement to Sort-Merge discussed on pages 254-255 in the text.

$$TotalCost = 3 * (M + N) = 3,600$$

NOTE: if $S.b$ were not a key, then the merging phase could require more than one pass over one of the relations, making the cost of merging $M * N$ I/Os in the worst case.

The minimum number of buffer pages required is 25. With 25 buffer pages, the initial sorting pass will split R into 20 runs of size 50 and split S into 4 runs of size 50 (approximately). These 24 runs can then be merged in one pass, with one page left over to be used as an output buffer. With fewer than 25 buffer pages the number of runs produced by the first pass over both relations would exceed the number of available pages, making a one-pass merge impossible.

4. The cost of Hash Join is $3 * (M + N)$ if $B > \sqrt{f * N}$ where f is a 'fudge factor' used to capture the small increase in size involved in building a hash table, and N is the number of pages in the smaller relation, S (see page 258). Since $\sqrt{N} \approx 14$, we can assume that this condition is met. We will also assume uniform partitioning from our hash function.

$$TotalCost = 3 * (M + N) = 3,600$$

Without knowing f we can only approximate the minimum number of buffer pages required, and a good guess is that we need $B > \sqrt{f * N}$.

5. The optimal cost would be achieved if each relation was only read once. We could do such a join by storing the entire smaller relation in memory, reading in the larger relation page-by-page, and for each tuple in the larger relation we search the smaller relation (which exists entirely in memory) for matching tuples. The buffer pool would have to hold the entire smaller relation, one page for reading in the larger relation, and one page to serve as an output buffer.

$$TotalCost = M + N = 1,200$$

The minimum number of buffer pages for this cost is $N + 1 + 1 = 202$.

6. Any tuple in R can match at most one tuple in S because $S.b$ is a primary key (which means the $S.b$ field contains no duplicates). So the maximum number of tuples in the result is equal to the number of tuples in R, which is 10,000.

The size of a tuple in the result could be as large as the size of an R tuple plus the size of an S tuple (minus the size of the shared attribute). This may allow only 5 tuples to be stored on a page. Storing 10,000 tuples at 5 per page would require 2,000 pages in the result.

7. The foreign key constraint tells us that for every R tuple there is *exactly* one matching S tuple (because $S.b$ is a key). The Sort-Merge and Hash Joins would not be affected, but we could reduce the cost of the two Nested Loops joins. If we make R the outer relation then for each tuple of R we only have to scan S until a match is found. This will require scanning only 50% of S on average.

   For Page-Oriented Nested Loops, the new cost would be

   $$TotalCost = M + (M * \frac{N}{2}) = 101,000$$

   and 3 buffer pages are still required.

   For Block Nested Loops, the new cost would be

   $$TotalCost = M + (\frac{N}{2}) * \lceil \frac{M}{B-2} \rceil = 3,000$$

   and again this cost can only be achieved with 52 available buffer pages.

Exercise 12.5 Consider the join of R and S described in Exercise 12.4.

1. With 52 buffer pages, if unclustered B+ indexes existed on $R.a$ and $S.b$, would either provide a cheaper alternative for performing the join (using the *Indexed Nested Loops* algorithm) than a Block Nested Loops Join? Explain.
   (a) Would your answer change if only five buffer pages were available?
   (b) Would your answer change if S contained only 10 tuples instead of 2,000 tuples?
2. With 52 buffer pages, if *clustered* B+ indexes existed on $R.a$ and $S.b$, would either provide a cheaper alternative for performing the join (using the *Indexed Nested Loops* algorithm) than a Block Nested Loops Join? Explain.
   (a) Would your answer change if only five5 buffer pages were available?
   (b) Would your answer change if S contained only 10 tuples instead of 2,000 tuples?
3. If only 15 buffers were available, what would the cost of Sort-Merge Join be? What would the cost of Hash Join be?
4. If the size of S were increased to also be 10,000 tuples, but only 15 buffer pages were available, what would the cost of Sort-Merge Join be? What would the cost of Hash Join be?

5. If the size of S were increased to also be 10,000 tuples, and 52 buffer pages were available, what would the cost of Sort-Merge Join be? What would the cost of Hash Join be?

Answer 12.5 Assume that it takes 3 I/Os to access a leaf in R, and 2 I/Os to access a leaf in S. And since S.b is a primary key, we will assume that every tuple in S matches 5 tuples in R.

1. The Index Nested Loops join involves probing an index on the inner relation for each tuple in the outer relation. The cost of the probe is the cost of accessing a leaf page plus the cost of retrieving any matching data records. The cost of retrieving data records could be as high as one I/O per record for an unclustered index.

   With R as the outer relation, the cost of the Index Nested Loops join will be the cost of reading R plus the cost of 10,000 probes on S.

   $$TotalCost = 1,000 + 10,000 * (2 + 1) = 31,000$$

   With S as the outer relation, the cost of the Index Nested Loops join will be the cost of reading S plus the cost of 2000 probes on R.

   $$TotalCost = 200 + 2,000 * (3 + 5) = 16,200$$

   Neither of these solutions is cheaper than Block Nested Loops join which required 4,200 I/Os.

   (a) With 5 buffer pages, the cost of the Index Nested Loops joins remains the same, but the cost of the Block Nested Loops join will increase. The new cost of the Block Nested Loops join is

   $$TotalCost = N + M * \lceil \frac{N}{B-2} \rceil = 67,200$$

   And now the cheapest solution is the Index Nested Loops join with S as the outer relation.

   (b) If S contains 10 tuples then we'll need to change some of our initial assumptions. Now all of the S tuples fit on a single page, and it will only require a single I/O to access the (single) leaf in the index. Also, each tuple in S will match 1,000 tuples in R.

   Block Nested Loops:

   $$TotalCost = N + M * \lceil \frac{N}{B-2} \rceil = 1,001$$

Index Nested Loops with R as the outer relation:

$$TotalCost = 1,000 + 10,000 * (1 + 1) = 21,000$$

Index Nested Loops with S as the outer relation:

$$TotalCost = 1 + 10 * (3 + 1,000) = 10,031$$

Block Nested Loops is still the best solution.

2. With a clustered index the cost of accessing data records becomes one I/O for every 10 data records.

   With R as the outer relation, the cost of the Index Nested Loops join will be the cost of reading R plus the cost of 10,000 probes on S.

   $$TotalCost = 1,000 + 10,000 * (2 + 1) = 31,000$$

   With S as the outer relation, the cost of the Index Nested Loops join will be the cost of reading S plus the cost of 2000 probes on R.

   $$TotalCost = 200 + 2,000 * (3 + 1) = 8,200$$

   Neither of these solutions is cheaper than Block Nested Loops join which required 4,200 I/Os.

   (a) With 5 buffer pages, the cost of the Index Nested Loops joins remains the same, but the cost of the Block Nested Loops join will increase. The new cost of the Block Nested Loops join is

   $$TotalCost = N + M * \lceil \frac{N}{B-2} \rceil = 67,200$$

   And now the cheapest solution is the Index Nested Loops join with S as the outer relation.

   (b) If S contains 10 tuples then we'll need to change some of our initial assumptions. Now all of the S tuples fit on a single page, and it will only require a single I/O to access the (single) leaf in the index. Also, each tuple in S will match 1,000 tuples in R.

   Block Nested Loops:

   $$TotalCost = N + M * \lceil \frac{N}{B-2} \rceil = 1,001$$

   Index Nested Loops with R as the outer relation:

   $$TotalCost = 1,000 + 10,000 * (1 + 1) = 21,000$$

Index Nested Loops with S as the outer relation:

$$TotalCost = 1 + 10 * (3 + 100) = 1,031$$

Block Nested Loops is still the best solution.

3. SORT-MERGE: With 15 buffer pages we can sort R in three passes and S in two passes. The cost of sorting R is $2 * 3 * M = 6,000$, the cost of sorting S is $2 * 2 * N = 800$, and the cost of the merging phase is $M + N = 1,200$.

$$TotalCost = 6,000 + 800 + 1,200 = 8,000$$

HASH JOIN: With 15 buffer pages the first scan of S (the smaller relation) splits it into 14 buckets, each containing about 15 pages. To store one of these buckets (and its hash table) in memory will require $f * 15$ pages, which is more than we have available. We must apply the Hash Join technique again to all partitions of R and S that were created by the first partitioning phase. Then we can fit an entire partition of S in memory. The total cost will be the cost of two partioning phases plus the cost of one matching phase.

$$TotalCost = 2 * (2 * (M + N)) + (M + N) = 6,000$$

4. SORT-MERGE: With 15 buffer pages we can sort R in three passes and S in three passes. The cost of sorting R is $2 * 3 * M = 6,000$, the cost of sorting S is $2 * 3 * N = 6,000$, and the cost of the merging phase is $M + N = 2,000$.

$$TotalCost = 6,000 + 6,000 + 2,000 = 14,000$$

HASH JOIN: Now both relations are the same size, so we can treat either one as the smaller relation. With 15 buffer pages the first scan of S splits it into 14 buckets, each containing about 72 pages, so again we have to deal with partition overflow. We must apply the Hash Join technique again to all partitions of R and S that were created by the first partitioning phase. Then we can fit an entire partition of S in memory. The total cost will be the cost of two partioning phases plus the cost of one matching phase.

$$TotalCost = 2 * (2 * (M + N)) + (M + N) = 10,000$$

5. SORT-MERGE: With 52 buffer pages we have $B > \sqrt{M}$ so we can use the "merge-on-the-fly" refinement which costs $3 * (M + N)$.

$$TotalCost = 3 * (1,000 + 1,000) = 6,000$$

HASH JOIN: Now both relations are the same size, so we can treat either one as the smaller relation. With 52 buffer pages the first scan of S splits it into 51 buckets, each containing about 20 pages. This time we do not have to deal with partition overflow. The total cost will be the cost of one partioning phase plus the cost of one matching phase.

$$TotalCost = (2 * (M + N)) + (M + N) = 6,000$$

Exercise 12.6 Answer all the questions—if some question is inapplicable, explain why—in Exercise 12.4, but using the following information about R and S:

Relation R contains 200,000 tuples and has 20 tuples/page.
Relation S contains 4,000,000 tuples and also has 20 tuples/page.
Attribute a of relation R is the primary key for R.
Each tuple of R joins with exactly 20 tuples of S.
1002 buffer pages are available.

Answer 12.6 Let M = 10,000 be the number of pages in R, N = 200,000 be the number of pages in S, and B = 1002 be the number of buffer pages available.

1. Basic idea is to read each page of the outer relation, and for each page scan the inner relation for matching tuples. Total cost would be

   $$\#pagesinouter + (\#pagesinouter * \#pagesininner)$$

   which is minimized by having the smaller relation be the outer relation.

   $$TotalCost = M + (M * N) = 2,000,010,000$$

   The minimum number of buffer pages for this cost is 3.

2. This time read the outer relation in *blocks*, and for each block scan the inner relation for matching tuples. So the outer relation is still read once, but the inner relation is scanned only once for each outer block, of which there are $\lceil \frac{\#pagesinouter}{B-2} \rceil$.

   $$TotalCost = M + N * \lceil \frac{M}{B-2} \rceil = 2,010,000$$

   The minimum number of buffer pages for this cost is 1002.