James Russo
Shiven Srivastava
Jdr289
Ss2654

2.1

a) $Cost = M + (M * p_R) * cost\ of\ finding\ matching\ S\ tuples$ where M is the number of pages in R and p_R is the number of tuples of R divided by the number of pages of R. Due to clustering, we know that cost will equal this because the cost of finding S tuples is typically 1 I/O. Since we are using a B+ index, the cost of finding matching S tuples is 4 I/Os.

b) $Cost \leq M + (M * p_S) * cost\ of\ finding\ matching\ R\ tuples$ where M is the number of pages in S and p_S is the number of tuples of S divided by the number of pages of S. Since this join is unclustered, we know that cost will be less than or equal because the cost of finding R tuples is up to 1 I/O. Since we are using a B+ index, the cost of finding matching R tuples is 4 I/Os.

c) $Cost = M + (M * p_R) * cost\ of\ finding\ matching\ S\ tuples.$ We scan reserves to find that it has M pages and (M * p_R) total tuples. Since we have a hash-based lookup, it takes 1.3 I/Os to find the index page with data entries for each tuple. It takes an additional 1 I/O to get the exactly matching S tuple since we are looking at only one key (A).

d) $Cost = MlogM + NlogN + (M + N)$ where M is the number of pages in R, N is the number of pages in S, and (M + N) is the cost to scan R and S. M log M and N log N are the costs to sort R and S respectively. We know that the cost to scan R and S is M + N ≤ scan cost ≤ M*N, but is normally M + N. To do a Sort-Merge Join we have to scan sorted versions of R and S until the current tuples match up on A.

2.2

$2^k = 30{,}000$
$log_2(30{,}000) \approx 15 = k\ (number\ of\ passes\ in\ Phase\ 2)$
Given that k = 15, we know that it will take 16 passes to produce one sorted run of 30,000 pages. By using the formula $Cost = 2N(\lceil log_2 N \rceil + 1)$ with N = 30,000 we get that the total cost is 953,361 I/Os.
It IS NOT possible to sort this file in just two passes with a TWO-WAY EXTERNAL MERGE SORT because we can only use 3 buffer pages in said sort. If we were using a different sorting method, that could use more buffer pages, then it might be possible.

2.3

a) For this query, accessing the sorted file directly is most likely the fastest method because there is no additional I/O cost associated with traversal or retrieval.

b) For this query, we would use a linear hashed index on attribute *R.a*., assuming that there are not many overflow pages. This method is fast at finding tuples that match an equality constraint. Since B+ tree indices are most useful for comparison operations and accessing the sorted file directly would take time proportional to the file length, this method is the best for this query.

c) For this query, using a B+ tree index on attribute *R.a* is most likely the cheapest because trees best match terms that include attributes in the prefix of the search key only and do not need to involve every attribute of the relation. Since the query is asking for tuples where 216,000 < a < 218,000, we do not need to concern ourselves with attributes *R.b* or *R.c*.

d) Like in part a), the best approach here is to access the sorted file directly. Since we are concerned with an (in)equality operation, we might want to use the linear hashed index, but accessing the file directly eliminates some small I/O costs involved with traversal, etc.

2.4

Assume reduction Factor of 0.1 applies to each condition (not to each part of the condition).
Assume not 100% clustering.
a)
i) For this scenario, the method would be a file scan. None of the available index options use age and are clustered. If we have any scenario where we are not using age, then there is no point in indexing. When we use the unclustered B+ tree on age, we get the following:
The retrieval and index cost will be

$$2\ (navigation) + \ 64000(pages) \times \frac{20(bytes\ per\ index)}{160(bytes\ per\ record)} \times 0.1(R.F.) = \ 802\ I/Os$$

The lookup cost will be

$$64000(pages) \times 16(tuples\ per\ page) \times 0.1(R.F.) = 102400\ I/Os$$

The total cost will then be

$$802 + 102400 = 103202\ I/Os$$

A file scan, alternatively, will have a cost of $64000\ I/Os$, which is significantly less.
ii) In this case, it is better to use an index sorted on state. The clustered B+ tree is best for this scenario because there will not be a massive cost from having to search everywhere for relevant tuples. Additionally, it is grouped on state first, so the age grouping is not relevant here. The index cost would be

$$2\ (navigation) + \ 64000(pages) \times \frac{20(bytes\ per\ index)}{160(bytes\ per\ record)} \times 0.1(R.F.) = \ 802\ I/Os$$

The lookup cost will be

$$64000(pages) \times 0.1(R.F.) = 6400\ I/Os$$

The total cost will then be

$$802 + 6400 = 7202\ I/Os$$

iii) For this condition it makes sense to first generate a table where age > 5 and then find tuple that match category = “Technology”. To do the first step, we follow the same procedure as in i) which has a cost = 64000 I/Os. For the second step, it takes no additional I/Os because the check on category = “Technology” can happen at the same time we are scanning the file for age > 5. Thus, the best method is a direct file scan which has a cost of 64000 I/Os
iv) For this case, we should use a clustered B+ tree once again because we are concerned with state and age (which this index is sorted on). Thus, we get that our index cost is

$$2\ (navigation) + \ 64000(pages) \times \frac{20(bytes\ per\ index)}{160(bytes\ per\ record)} \times 0.1(R.F.) = \ 802\ I/Os$$

For the lookup, adding additional constraints does not increase I/O cost. Thus, the lookup cost will be

$$64000(pages) \times 0.1(R.F.) = 6400\ I/Os$$

The total cost will then be
$802 + 6400 = 7202\ I/Os$

b) The least expensive evaluation method is to use the clustered B+ tree index. For this scenario we can find the relevant indices (i.e. just the indices where state is California) and then just keep a counter of the ages to be summed and divided to find the average. As such, the cost will be

$$2\ (to\ scan) + \ 64000(pages) \times \frac{20(bytes\ per\ index)}{160(bytes\ per\ record)} \times 0.1(R.F.) = \ 802\ I/Os$$