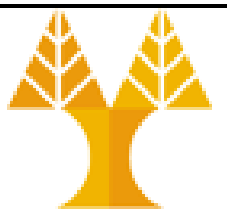

ΕΡΓΑΣΤΗΡΙΟ #5

External Sorting and Evaluation of Relational Operators

1. **(Exercise 13.1)** Suppose you have a file with 10,000 pages and you have three buffer pages. Answer the following questions for each of these scenarios, assuming that our most general external sorting algorithm is used:
   i. A file with 10,000 pages and three available buffer pages.
   ii. A file with 20,000 pages and five available buffer pages.
   iii. A file with 2,000,000 pages and 17 available buffer pages.

   Questions
   i. How many runs will you produce in the first pass?
   ii. How many passes will it take to sort the file completely?
   iii. What is the total I/O cost of sorting the file?
   iv. How many buffer pages do you need to sort the file completely in just two passes?

   Answer:
   i. In the first pass (Pass 0), $\lceil N/B \rceil$ runs of B pages each are produced, where N is the number of file pages and B is the number of available buffer pages:
      a. $\lceil 10000/3 \rceil = 3334$ *sorted runs.*
      b. $\lceil 20000/5 \rceil = 4000$ *sorted runs.*
      c. $\lceil 2000000/17 \rceil = 117648$ *sorted runs.*
   ii. The number of passes required to sort the file completely, including the initial sorting pass, is $\lceil log_{B-1} N1 \rceil + 1$, where $N1 = \lceil N/B \rceil$ is the number of runs produced by Pass 0:
      a. $\lceil log_2 3334 \rceil + 1 = 13$ *passes.*
      b. $\lceil log_4 4000 \rceil + 1 = 7$ *passes.*
      c. $\lceil log_{16} 117648 \rceil + 1 = 6$ *passes*
   iii. Since each page is read and written once per pass, the total number of page I/Os for sorting the file is $2 * N * (\#passes)$:
      *a. 2*10000*13 = 260000.*
      *b. 2*20000*7 = 280000.*
      *c. 2*2000000*6 = 24000000.*
   iv. In Pass 0, $\lceil N/B \rceil$ runs are produced. In Pass 1, we must be able to merge this many runs; i.e., $B - 1 \geq \lceil N/B \rceil$ This implies that B must at least be large enough to satisfy $B * (B - 1) \geq N$; this can be used to guess at B, and the guess must be validated by checking the first inequality. Thus:
      a. With 10000 pages in the file, B = 101 satisfies both inequalities, B = 100 does not, so we need 101 buffer pages.
      b. With 20000 pages in the file, B = 142 satisfies both inequalities, B = 141 does not, so we need 142 buffer pages.
      c. With 2000000 pages in the file, B = 1415 satisfies both inequalities, B = 1414 does not, so we need 1415 buffer pages..

2. **(Exercise 14.3)** Consider processing the following SQL projection query:

SELECT DISTINCT E.title, E.ename FROM Executives E

You are given the following information:
Executives has attributes ename, title, dname, and address; all are string fields of the same length.
The ename attribute is a candidate key.
The relation contains 10,000 pages.
There are 10 buffer pages.

Consider the optimized version of the sorting-based projection algorithm: The initial sorting pass reads the input relation and creates sorted runs of tuples containing only attributes ename and title. Subsequent merging passes eliminate duplicates while merging the initial runs to obtain a single sorted result (as opposed to doing a separate pass to eliminate duplicates from a sorted result containing duplicates).

i. How many sorted runs are produced in the first pass? What is the average length of these runs? (Assume that memory is utilized well and any available optimization to increase run size is used.) What is the I/O cost of this sorting pass?
ii. How many additional merge passes are required to compute the final result of the projection query? What is the I/O cost of these additional passes?
iii.
   a. Suppose that a clustered B+ tree index on *title* is available. Is this index likely to offer a cheaper alternative to sorting? Would your answer change if the index were unclustered? Would your answer change if the index were a hash index?
   b. Suppose that a clustered B+ tree index on *ename* is available. Is this index likely to offer a cheaper alternative to sorting? Would your answer change if the index were unclustered? Would your answer change if the index were a hash index?
   c. Suppose that a clustered B+ tree index on <*ename, title*> is available. Is this index likely to offer a cheaper alternative to sorting? Would your answer change if the index were unclustered? Would your answer change if the index were a hash index?
iv. Suppose that the query is as follows:
   SELECT E.title, E.ename FROM Executives E
   That is, you are not required to do duplicate elimination. How would your answers to the previous questions change?

Answer:

i. The first pass will produce 250 sorted runs of 20 pages each, costing 15000 I/Os. Note: the sorted records have half the size of the original records!

ii. To ***merge*** *the produced runs we need* $\lceil log_{B-1}\lceil N/B \rceil \rceil = \lceil log_9 500 \rceil = 3$ *more passes that cost 2*3*5000 = 30000 I/Os.*

iii.

*a. Using a clustered B+ tree index on title would reduce the cost to single scan, or 12,500 I/Os. An unclustered index could potentially cost more than 2500+100,000 (2500 from scanning the B+ tree, and 10000 * tuples 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.*

iv. *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.)*