Storage and Querying Exercises

1. Consider that we build a B^+ tree for a data file with N many search key values. If we want to do equality search on the search key value in no more than 5 disk accesses (that is, no more than 5 block seeks and 5 block transfers), then explain how to design the B^+ tree?

   Let us design a node with n pointers. Then each internal nodes with have at least $\lceil n/2 \rceil$ children. With height 4, the B^+-tree has at least $\lceil n/2 \rceil^4$ leaf nodes. Each leaf node has at least $\lceil (n-1)/2 \rceil$ search key values. Thus, with height 4, the B^+-tree will have $\lceil n/2 \rceil^4 \cdot \lceil (n-1)/2 \rceil$ search key values.

   $$\lceil n/2 \rceil^4 \cdot \lceil (n-1)/2 \rceil > \lceil (n-1)/2 \rceil^5$$

   Let

   $$\lceil (n-1)/2 \rceil^5 \geq N$$

   We have

   $$n \geq 2\sqrt[5]{\mathrm{N}} + 1$$

   Thus, we can choose the smallest n greater than or equal to $2\sqrt[5]{N} + 1$.

   For Example, when N = 10,000,000, we just need to set n = 52.

2. Assume that we have a B^+tree index for a relation based on a search key A, explain how to perform a range query, that is, to find tuples such that there search key values satisfying $a \leq A \leq b$?

   First, find the smallest search key K value $\geq a$. This leads to a leaf node with the search key value K. From that leaf node, perform linear scan of the leaf nodes till either we reach the end or find the first search key value $> b$.

3. Explain whether you can use a dynamic hash index based on a search key A to perform comparison search, such as, to find all tuples satisfying $A \leq a$?

   Easy, right? :=)

4. What is the worst-case time complexity for the linear search algorithm?

   Recall that we focus on number of disk seeks and number of block transfers.

   Assume the data file is stored in sequential order with b consecutive blocks. We need to 1 disk seek to find the first block of the file, and from there we read the file one block at a time till either reach the last block or we find the search key value. So, the worst time is

   1 seek + b block transfers.

5. Assume that a primary B^+ tree index is available, what is the worst-case time complexity for the equality search on key?

   Assume the B^+ tree index is stored in secondary memory and we need on disk seek and one disk transfer to access a tree node. Assume the tree height is h.

We need (h+1) seeks + (h+1) disk transfers.

6. What is the worst-case time complexity for the merge-join algorithm, assuming that the join is an equi-join or natural join?

   Let r and s be two relations with b_r and b_s blocks, respectively. Assume that both relations are stored sequentially in consecutive blocks.

   Suppose that both relations are sorted. We can load w blocks once from r and s to do merge-join, then the time we need is $b_r + b_s$ block transfers $+\lceil b_r/w\rceil + \lceil b_s/w\rceil$ seeks.

   If r and s are not sorted, and then we need to use external merge-sort to sort them. Suppose each time we can load M blocks to the buffer for merging with w blocks per run to facilitate faster processing. Total number of disk transfers is

   $b_r(2\lceil \log_{\lfloor M/w\rfloor-1}(b_r/M)\rceil + 1)\rceil + b_s(2\lceil \log_{\lfloor M/w\rfloor-1}(b_s/M)\rceil + 1)\rceil$.

   And, the total number of disk seeks is
   $2\lceil b_r/M\rceil + \lceil b_r/w\rceil(2\lceil \log_{\lfloor M/w\rfloor-1}(b_r/M)\rceil -1) + 2\lceil b_s/M\rceil + \lceil b_s/w\rceil(2\lceil \log_{\lfloor M/w\rfloor-1}(b_s/M)\rceil -1)$.

7. What is the worst-case time complexity for the hash-join algorithm, assuming that the join is an equi-join or natural join?

   Let r and s be two relations with b_r and b_s blocks, respectively. Assume that both relations are stored sequentially in consecutive blocks. Assume that a hash function h is used to partition r and s each into n buckets. Now, we just need to join blocks from a bucket of r with blocks of a bucket from s such that both buckets have he same hash value under h.

   For each block of r and s, we need read and write back for hash partitioning, and then read for joining. Thus, the total number of block transfers is $3\ b_r + 3b_s$. There may be some extra blocks added during the hash partitions due to under-full block within each bucket. But these can be ignored.

   Suppose we can load w blocks once into the buffer for faster processing. We need to read and write blocks for hash partitioning. Thus, we need $2\lceil \mathrm{b_r}/w\rceil + 2\lceil \mathrm{b_s}/w\rceil$ seeks to partition r and s. Since we join blocks from two buckets with the same hash value, assume that we can load the smaller buckets entirely into the buffer, then we need only one seek per bucket. Thus, the total number of seeks is $2\lceil \mathrm{b_r}/w\rceil + 2\lceil \mathrm{b_s}/w\rceil + 2n$.

8. Given a relation *instructor(<u>ID</u>, name, department, salary),* devise an algorithm to do

   select department, sum(salary) as departmentTotalSalary
   from instructor
   group by department

   discuss the time complexity of the algorithm.

Think about it. It is not hard to find an algorithm, right? :=)

9. Let r and s be relations with no indices and assume that the relations are not sorted. Assuming infinite memory, what is the lowest-cost way (in terms of I/O operations) to compute r ⋈ s? What is the amount of memory required for this algorithm?

   Easy. :=)

10. Suppose that a B^+-tree index on **building** is available on relation *department*, and that no other index is available. What is the best way to handle the following selections?
    a. $\sigma_{\neg(building < \text{"Watson"})}(department)$
    b. $\sigma_{\neg(building = \text{"Watson"})}(department)$
    c. $\sigma_{\neg(building < \text{"Watson"} \vee \text{budget} < 50000)}(department)$

    Hint:

    For a, we have an equivalent query: $\sigma_{building \geq \text{"Watson"}}(department)$. Can you see the solution now?

    For b, an equivalent query is $\sigma_{building \neq \text{"Watson"}}(department)$. Will the B^+-tree index help you?

    For c, we have an equivalent query: $\sigma_{building \geq \text{"Watson"} \wedge \text{budget} \geq 50000}(department)$. Will the B^+-tree index help you?