Assignment 9

1. (a) Assume that we have a primary index on the primary key of a table of N records which is maintained as a B$^+$-tree of order n.

   i. Argue why the search time to a record is $O(\log_n(N))$.

   ii. Argue why the insert time of a record is $O(\log_n(N))$.

   iii. Argue why the delete time of a record is $O(\log_n(N))$.

   Solutions: For the solutions, see the last several slides of the lecture notes on B$^+$-trees.

   (b) Consider the following parameters:

block size	=	4096 bytes
block-address size	=	9 bytes
block access time	=	10 ms (micro seconds)
record size	=	200 bytes
record key size	=	12 bytes

   Assume that there is a B$^+$-tree, adhering to these parameters, that indexes $100 = 10^8$ million records on their primary key values.

   Show all the intermediate computations leading to your answers.

   i. Specify (in ms) the minimum time to retrieve a record with key k in the B$^+$-tree provided that there is a record with this key.
   Solution: We first need to determine the order n of the B$^+$-tree. This can be done by finding the largest n such that

$$9(n+1) + 12n \leq 4096$$

   I.e., $n \leq \frac{4087}{21}$. Thus $n = 194$.
   The minimum time will be $(\lceil log_{195}(10^8)\rceil + 1 = 4+1) * 10ms = 50ms$. The $+1$ occurs because we need one more block access to the record in the data file.
   Observe that the minimum time is determined by the largest possible fanout of the nodes in the B$^+$-tree. This fanout is maximally $n + 1 = 194 + 1 = 195$.

   ii. Specify (in ms) the maximum time to retrieve a record with key k in the B$^+$-tree.

Solution: The maximum time results when we have the minimum branching factor, i.e. $\lceil \frac{194}{2} \rceil + 1 = 98$.
Since the minimum branching factor at the root is 2, we must determine the height of a tree that has half of the nodes 2, i.e., $\frac{10^8}{2} = 5 \times 10^7$ nodes.
The height of such a the tree will be $\lceil log_{98}(5 \times 10^7) \rceil = 4$.
We then also need one more block access to get the record, so the maximum time is $(4 + 1 + 1)10ms = 60ms$.
So we note that the minimum and maximum times are off by just 1 block access.

iii. How many records would there need to be indexed to increase the maximum time to retrieve a record with key k in the B$^+$-tree by at least 20 ms?
Solution: This would happen if the height of the tree increases by 2. So we would then need $98^2 * 10^8$ records.

iv. How would your answer to question 1(b)ii change if the block size is 8192 bytes.
Solution: In this case, $n \leq \lceil \frac{8183}{21} \rceil = 389$. Doing the same calculation as in question 1(b)ii, we would get $\lceil \frac{389}{2} \rceil + 1 = 196$.
We must then consider $\lceil log_{196}(5 \times 10^7) \rceil = 4$.
So the answer $(4 + 1 + 1)10ms = 60ms$.
This suggest that doubling the size of a block does not necessarily decrease the search time.

(c) Consider the following B$^+$-tree of order 2 that holds records with keys 2, 7, 9, and 11. (Observe that (a) an internal node of a B$^+$-tree of order 2 can have either 1 or 2 keys values, and 2 or 3 sub-trees, and (b) a leaf node can have either 1 or 2 key values.)

```
          -----------
          | /  9  \ |
          -----------
           /       \
          /         \
 ----------     ------------
 | 2   7 |--->| 9     11 |
 ----------     ------------
```

i. Show the contents of your B^+-tree after inserting records with keys 6, 10, 14, and 4, in that order.

ii. Starting from your answer in question 1(c)i, show the contents of your B^+-tree after deleting records with keys 2, 14, 4, and 10, in that order.

2. (a) Describe how overflow is handled in extendible hashing.

Solution: See textbook, Section 14.3.6.

(b) Describe how underflow is handled in extendible hashing.

Solution: Underflow may occur when records are deleted. Underflow is handled as the inverse of overflow.

When a deletion yields an underflow, the bucket where the underflow occurs will be merged with its corresponding buddy. The bucket that underflowed will be deleted and the buddy's local depth will be decreased by 1. When this decrease leads to a situation where **all** local depths are strictly less that the global depth, then the directory is halved and the global depth will be decreased by 1.

(c) Consider an extensible hashing data structure wherein (1) the initial global depth is set at 1 and (2) all directory pointers point to the same **empty** block which has local depth 0. So the hashing structure looks like this:

```
global-depth 1       local-depth 0
    ------         -----------------
  0 |  --|-------->|               |
    ------         |               |
  1 |  --|-------->|               |
    ------         -----------------
```

Assume that a block can hold at most two records.

i. Show the state of the hash data structure after each of the following insert sequences:[1]

   A. records with keys 2 and 6.

   B. records with keys 1 and 7.

   C. records with keys 4 and 8.

   D. records with keys 0 and 9.

ii. Starting from the answer you obtained for Question 2(c)i, show the state of the hash data structure after each of the following delete sequences:

   A. records with keys 1 and 2.

   B. records with keys 6 and 7.

   C. records with keys 0 and 9.

(d) Note: This question is independent of question 2(c)i and question 2(c)ii. Give an example where the insertion of a record in an extensible hash data structure can result in the increase of the global depth by 3.

You can assume that a block can only hold 2 records.

[1]You should interpret the key values as bit strings of length 4. So for example, key value 7 is represented as the bit string 0111 and key value 2 is represented as the bit string 0010.

3. Let $R(A, B)$ and $S(B, C)$ be two relations and consider their natural join $R \bowtie S$.

   Assume that R has 1,500,000 records and that S has $5,000$ records.

   Furthermore, assume that 30 records of R can fit in a block and that 10 records of S can fit in a block.

   Assume that you have a main-memory buffer with 101 blocks.

   (a) How many block IO's are necessary to perform $R \bowtie S$ using the block nested-loops join algorithm? Show your analysis.

   Solution: The complexity is $B(S) + \frac{B(R)B(S)}{100}$.

   $$\begin{array}{rcl} B(R) & = & \frac{1500000}{30} = 50000 \\ B(S) & = & \frac{5000}{10} = 500 \end{array}$$

   So $B(S) + \frac{B(R)B(S)}{100} = 500 + 5.50000 = 250500$.

   (b) How many block IO's are necessary to perform $R \bowtie S$ using the sort-merge join algorithm? Show your analysis.

   Solution In this case, we first need to determine how much time is involved in sorting R and S.

   External sorting R takes $2B(R)\log_{100}(B(R))$. This is $2 \cdot 50000\lceil \log_{100}(5.10^4)\rceil =$ 300000.

   Sorting S takes $2B(S)log_{100}(B(S))$. This is $2 \cdot 500\lceil log_{100}(500)\rceil =$ 2000.

   The merge phase of these sorted files is $B(R) + B(S)$ which is $50000 + 500 = 50500$.

   So the total is $300000 + 2000 + 50500 = 352500$.

   (c) Repeat question 3b under the following assumptions.

   Assume that there are p different B-values and that these are uniformly distributed in R and S.

   Observe that to solve this problem, depending on p, it may be necessary to perform a block nested-loop join per occurrence of a B-value..

   Solution:

   The time to sort R and S remains the same. So this account for $300000 + 2000 = 302000$.

- Consider the case $p = 1$. Clearly, neither R nor S fit in the buffer. So, we do a block nested-loop join. Since S is the smaller relation and since $p = 1$, this requires $B(S) + \frac{B(R)B(S)}{100}$ block accesses. So, in total, an additional 250500 block accesses.
- Consider the case $p = 2$. Since now we have 2 B-values, we need 2 block nested-loop joins between a file of size $\frac{50000}{2}$ and a file of size $\frac{500}{2}$. Since S is the smaller relation, we can accomplish each of these in time $250 + \frac{250 \cdot 25000}{100} = 62750$. Does together we have $2 \cdot 62750 = 125500$ block accesses.
- Consider the case $p = 3$. Since now we have 3 B-values, we need 3 block nested-loop joins between a file of size $\frac{50000}{3}$ and a file of size $\frac{500}{3}$, so approximately between a file of size 17000 and one of size 170. Since S is the smaller relation, we can accomplish each of these in time $170 + \frac{170 \times 17000}{100} = 29070$. Thus, we need $3 \times 29070 = 87210$ block accesses.
- We leave the case for $p = 4$ as an exercise.
- Consider the case $p = 5$. Since now we have 5 B-values, we need 5 block nested-loop joins between a file of size $\frac{50000}{5} = 10000$ and a file of size $\frac{500}{5} = 100$. Since now each such chunk of S fits in memory, we get, for each such chunk $100 + 10000 = 10100$ block accesses Since $p = 5$, we get an extra 50500 block accesses.

  The cases for $p > 5$ are similar and we leave these as an exercise.

(d) How many block IO's are necessary to perform $R \bowtie S$ using the hash-join algorithm? Show your analysis.

Solution: Note: For questions 3b and 3d you can assume that the buffer is sufficiently large to hold all records from R and S for any give B-value.

The complexity is $3(B(R) + B(S)) = 3(50000 + 50) = 150150$.

(e) Repeat question 3d under the following assumptions.

Assume that there are p different B-values and that these are uniformly distributed in R and S.

Observe that to solve this problem, depending on p, it may be necessary to perform a block nested-loop join per different B-value.

Solution:

The time to hash R and S is $2(B(R) + B(S)) = 101000$.

The rest of the analysis for different p values is as in question 3c.

4. State which of the following schedules S_1, S_2, and S_3 over transactions T_1, T_2, and T_3 are conflict-serializable, and for each of the schedules that is serializable, given a serial schedule with which that schedule is conflict-equivalent.

   (a) $S_1 = R_1(x)R_2(y)R_1(z)R_2(x)R_1(y)$.

   This schedule consists of just read operations. Thus, there are no conflicting pairs and thus all T_1 operations can be swapped with all T_2 operations and as such the schedule S_1 can be transformed to the serial schedule $R_1(x)R_1(z)R_1(y)R_2(y)R_2(x)$, i.e. the schedule $T_1; T_2$.

   (b) $S_2 = R_1(x)W_2(y)R_1(z)R_3(z)W_2(x)R_1(y)$.

   The precedence graph $P(S_2) = \{(T_1, T_2), (T_2, T_1)\}$ which is cyclic. Thus, S_2 is not serializable.

   (c) $S_3 = R_1(z)W_2(x)R_2(z)R_2(y)W_1(x)W_3(z)W_1(y)R_3(x)$.

   The precedence graph $P(S_3) = \{(T_1, T_3), (T_2, T_1), (T_2, T_3)\}$ which is acyclic. Thus, S_3 is serializable.

   Using topological sort on this graph, S_3 is equivalent with the serial schedule $T_2; T_1; T_3$.

5. Given 3 transactions T_1, T_2, T_3 and a serializable schedule S on these transactions whose precedence graph (i.e. serialization graph) consists of the edges (T_1, T_2) and (T_1, T_3). Give 2 serial schedules that are conflict-equivalent with S.

   One of these serial schedules is $T_1; T_2; T_3$. Another is $T_1; T_3; T_2$.

6. Give 3 transactions T_1, T_2, T_3 and a schedule S on these transactions whose precedence graph (i.e. serialization graph) consists of the edges (T_1, T_2), (T_2, T_3), and (T_1, T_3).

$$\begin{aligned} T_1 &= R_1(x)R_1(z) \\ T_2 &= W_2(x)W_2(y) \\ T_3 &= W_3(y)W_z(z) \end{aligned}$$

   $S = R_1(x)R_1(z)W_2(x)W_2(y)W_3(y)W_3(z)$.

7. Give 3 transactions T_1, T_2, T_3 and a schedule S on these transactions whose precedence graph (i.e. serialization graph) consists of the edges (T_1, T_2), (T_2, T_1), (T_1, T_3), (T_3, T_2).

   $$\begin{array}{lcl} T_1 & = & R_1(u)R_1(x)R_1(z) \\ T_2 & = & W_2(u)W_2(x)W_2(w) \\ T_3 & = & W_3(w)W_3(z) \end{array}$$

   $S = W_2(u)R_1(u)R_1(x)R_1(z)W_2(x)W_3(w)W_3(z)W_2(w)$.

8. Give 3 transactions T_1, T_2, and T_3 that each involve read and write operations and a schedule S that is conflict-equivalent with **all** serial schedules over T_1, T_2, and T_3.

   We could pick the following transactions

   $$\begin{array}{lcl} T_1 & = & R_1(x)W_1(x) \\ T_2 & = & R_2(y)W_2(y) \\ T_3 & = & R_3(z)W_3(z) \end{array}$$

   In this example there are no conflicting pairs and therefore any schedule over T_1, T_2, and T_3 can be transformed into any serial schedule using appropriate swap operations of non-conflicting operations.

9. Give an example of a serializable schedule that is conflict-equivalent with three different serial schedules but not equivalent with at least 3 serial schedules. (Hint: Consider a schedule with at least 3 transactions.)

   I decided to drop this question.

10. Consider the following transactions:

```
T1:  read(A);
     read(B);
     if A = 0 then B := B+1;
     write(B).

T2:  read(B);
     read(A);
     if B = 0 then A := A+1;
     write(A).
```

Let the consistency requirement be $\texttt{A} = 0 \vee \texttt{B} = 0$, and let $\texttt{A} = \texttt{B} = 0$ be the initial values.

(a) Show that each serial schedule involving transaction T1 and T2 preserves the consistency requirement of the database.

Solution:

For the serial schedule $T_1; T_2$ the value for $A = 0$ and that for $B = 1$.

For the serial schedule $T_1; T_2$ the value for $A = 1$ and that for $B = 0$.

(b) Construct a schedule on T1 and T2 that produces a non-serializable schedule.

Solution:

T1	T2
R(A)	
	R(B)
	R(A)
	if B= 0 then A := A+1
	W(A)
R(B)	
if A = 0 then B := B+1	
W(B)	

The precedence graph consists of the edges (T1,T2) and (T2,T1). This is a cyclic graph so this schedule is not serializable.

(c) Is there a non-serial schedule on T1 and T2 that produces a serializable schedule. If so, give an example.

Solution:

The answer is no.

Assume that the schedule begins with R1(A). Observe that this action conflicts with W2(A).

Assume that we consider the partial schedule R1(A) R2(B). Observe that R2(B) conflicts with W1(B). So if we start the schedule with R1(A)R2(B) we get a cyclic precedence graph. Consequently, if we begin the schedule with R1(A) then the next action must be R1(B). So we must have R1(A)R1(B). We could now consider R2(B). But this will again create a cycle in the precedence graph. Thus we are required to do "R1(A)R1(B) if A = 0 then B:=B+1". Given this if we consider R2(B) we again have a problem since we will get a cyclic precedence graph. Thus we conclude that if we begin our schedule with R1(A), then we must execute the serial schedule T1;T2.

Now assume that we begin the schedule with R2(B). A similar analysis as above shows that if we want a serializable schedule, then we need to execute the serial schedule T2;T1.

(d) i. Add lock and unlock instructions to T1 and T2, so that they observe the two-phase locking protocol, but in such a way that interleaving between operations in T1 and T2 is still possible.
Solution By the previous argument, this can not be done.

ii. Can the execution of these transactions result in a deadlock? If so, give an example.
Solution.
Yes this can happen. Notice the following initial segment of a possible schedule $l_1(A)l_2(B)r_1(A)r_2(B)$ at this point (1) T_1 wants to read B but T_2 has a lock on B and (2) T_2 wants to read A but T_1 has a lock on A. So this schedule deadlocks.