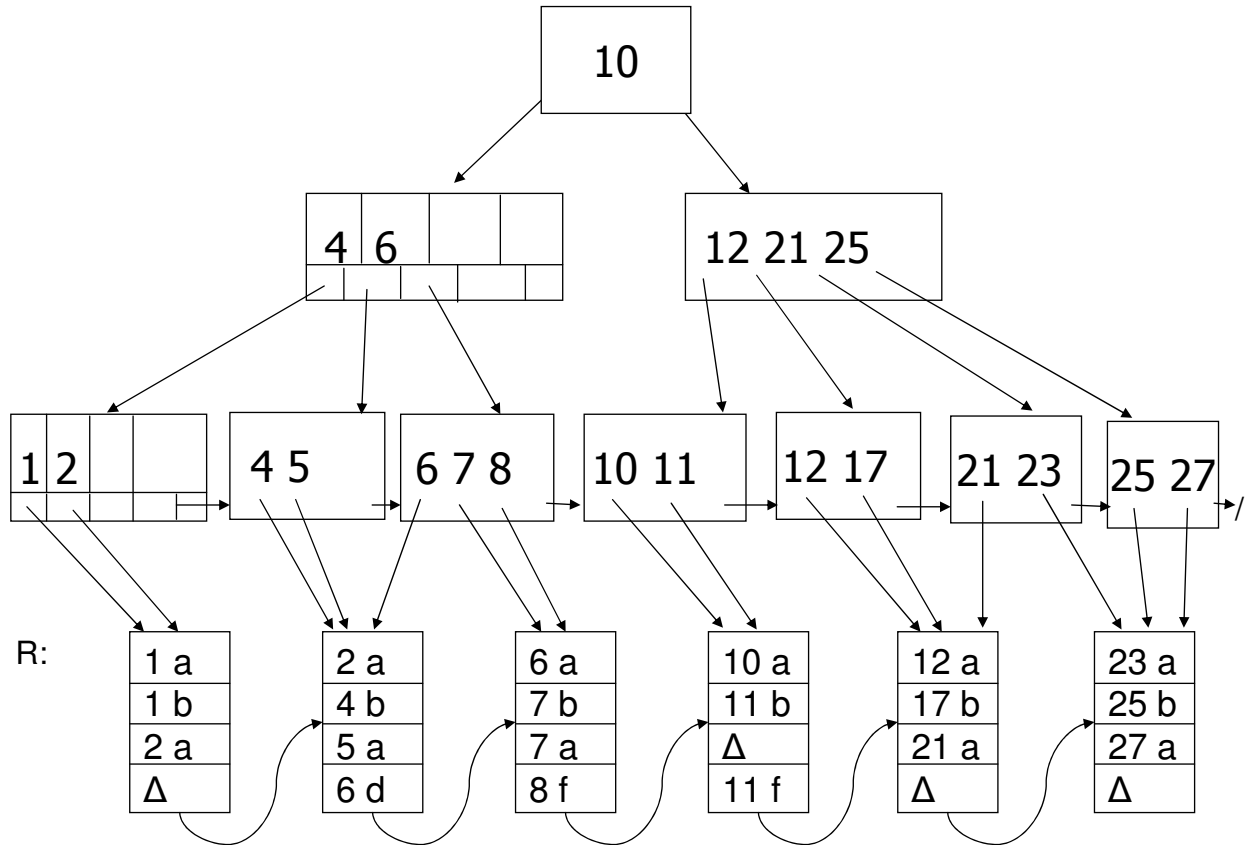


## 1 Indexes (30 minutes)

Given the following index sequential data file  $R(v,x)$  and B+ tree indexing  $v$ :



Assume the number of keys per block  $N = 4$ . You may assume that deletions in the data file are marked with a special value  $\Delta$  that indicates the item is deleted. Records marked with  $\Delta$  can be reused by a later insert.

Answer the following:

### 1.1 Deletion (7 minutes, 6 points)

Show the results of the following SQL transaction on the index and data file:

```
/* Transaction  $T_1$  starts here */
delete from R where  $v = 2$ ;
commit /*  $T_1$  */;
```



### 1.3 Locking (18 minutes)

Assume record-level locking, with two lock types (Shared and eXclusive). Assume 2-phase locking.

#### 1.3.1 The wrong way (11 minutes)

Since I have an index on  $R(v)$ , I can go directly to the records with  $v=2$ . Thus I only need two locks: eXclusive locks on the items with  $v=2$ .

1. **Demonstrate that this is wrong (6 minutes, 3 points)**

The preceding statement ( $T_1$  only needs two locks) is incorrect. Give a transaction  $T_2$  (sequence of reads and writes) and sequence of locks/unlocks for both  $T_1$  and  $T_2$  that does not violate 2-Phase Locking, but is not conflict serializable.

2. **Fix the problem (4 minutes, 2 points)**

What locks should  $T_1$  have?

#### 1.3.2 Index locking (6 minutes, 2 points)

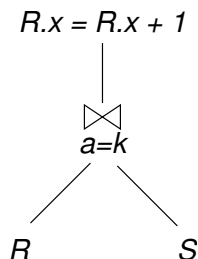
Does the presence of an index affect the locks we need? Briefly explain your answer (3-5 sentences.)

## 2 Query Plan (45 minutes)

Use the following two relations:

$R(a, x)$	$S(\underline{k}, z)$
$\text{Blocks}(R) = 10$	$\text{Blocks}(S) = 100$
$\text{Tuples}(R) = 100$	$\text{Tuples}(S) = 2000$
$\text{Values}(R,a) = 10$	$\text{Values}(S,k) = 2000$
$\text{Values}(R,x) = 5$	$\text{Values}(S,z) = 100$

The following query plan contains an update ( $x = x + 1$ ) that increments the value of  $x$  in relation  $R$  for the tuples that are returned by the result of the join.



Assume that an ID of each tuple is passed through the query. The ID enables direct access to the tuple. If a tuple contains values from two relations (e.g., a cross product or join), the ID of both tuples is included. Thus when we get to the update, we can directly access the tuple that needs to be updated. The cost of storing the ID is negligible (i.e.,  $R$  with tuple IDs attached still takes 10 blocks.)

You should also assume that buffer space is constrained: You can only hold 5 blocks in memory.

### 2.1 Nested Loop Join (10 minutes, 4 points)

Assume that we perform a nested loop join, where the outer loop is relation  $R$  (we choose an item from  $R$ , then run through all tuples in  $S$ ). Estimate the I/O cost (number of blocks read or written). If you just estimate sizes of intermediate results (as opposed to total I/Os), you will get partial credit.

## 2.2 Alternate query processing (20 minutes, 8 points)

Give way of processing the query (e.g., different query plan, different join algorithm, etc.) that you think is better, and estimate the I/O cost for your approach. Assume no indexes (unless you build them, in which case you must include the cost of building indexes in your I/O cost).

## 2.3 SQL update (10 minutes)

### 1. SQL update (5 minutes, 2 points)

Give an SQL query that would be correctly answered by the above query plan.

### 2. Correctness (5 minutes, 1 point)

Would your query plan still perform the correct updates for your SQL query if  $S(k)$  was not a key?

### **3 Undo vs. Redo Logging (40 minutes)**

Undo logging requires that the log record must be written to disk before the database is changed on disk, and all database changes must be written before the commit. Redo logging requires that the log records and commit be written to disk before any database changes are written to disk.

Keeping these in mind, answer the following. For each, give a brief (two to three sentence) justification of your answer. Remember that you are comparing undo and redo logging – not undo/redo logging.

#### **3.1 Fast Response Time (5 minutes, 3 points)**

Which logging method is best if the goal is a fast response time. Response time is defined as the time from when the transaction first reaches the system until it becomes persistent, i.e., the first point at which a crash followed by recovery will result in a system that shows the results of the transaction.

#### **3.2 High Throughput (5 minutes, 3 points)**

Which logging method would give the highest throughput, i.e., the most transactions per second?

#### **3.3 High Availability (5 minutes, 3 points)**

Which method would give the highest system availability? Assume failures are “fail-stop”, e.g., a power outage, and that the actual time when the computer is not operating is the same for each logging method. Availability is the time that the computer is ready to process transactions, which may be smaller than the time when the power is on. In other words, which approach gives the smallest recovery time.

### **3.4 Demonstrate your answer (15 minutes, 6 points)**

For **one** of questions 3.1, 3.2, or 3.3, give a detailed example demonstrating the correctness of your answer. In other words, provide a sequence of transactions, reads/writes performed by those transactions, I/Os required, and expected time for those I/Os that demonstrates conditions under which your answer is correct.