Problem 1 (10 points)

Suppose blocks of 1024 bytes are used to store variable-length records. The fixed part of the block header is 24 bytes, and includes a count of the number of records. In addition, the header contains a 4-byte pointer (offset) to each record in the block. The first record is placed at the end of the block, the second record starts where the first one ends, and so on. The size of each record is not stored explicitly, since the size can be computed from the pointers.

(a) How many records of size 50 bytes could we fully fit (no spanning) in such a block? (The block is designed to hold variable size records, but these records by coincidence happen to be all the same size.)

Number of records: $\lfloor \frac{1024-24}{50+4} \rfloor = 18$

(b) Suppose we want to fully store 20 records in the block, again all of the same size. What is the maximum size of such records?

Maximum record size: $\lfloor \frac{1024-24-4*20}{20} \rfloor = 46$

(c) Now assume the block layout is changed to accommodate only fixed size records. In this case, the block header is still 24 bytes and contains the common length of the records in the block.

How many records of size 50 bytes could we fully fit in such a block?

Number of records: $\lfloor \frac{1024-24}{50} \rfloor = 20$

(d) For this part, assume we allow record spanning. Records are fixed size, but since we can have record fragments we need to add a 5-byte header to every fragment to store its length (and other information). (The block header is still 24 bytes.)

Suppose we have 2 blocks of 1024 bytes each, we would like to store 3 records of 600 bytes each. We first allocate as much as we can of the records into the first block, and then use space from the second block. After the 3 records are stored, how many bytes are still available in the second block?

Free bytes available: 180

Record 2 is split into fragments 2-a and 2-b. Block 1 has room for 390 bytes of record 2-a (1024-24-5-600-5). Block 2 needs to store 210 bytes of record 2-b. Block 2 also needs to store block header, record 3, and two record headers (for record 2-b and 3). Therefore, 180 bytes (1024-24-5-210-5-600) are still available in block 2.

Problem 3 (10 points)

(a) Consider an extensible hash structure with the following characteristics:

- Buckets can hold up to two records.
- No overflow blocks are allowed.
- The hash function we use generates $b = 4$ bits total.
- Initially the extensible hash table is empty.

Say we insert X records, where the search key of each record generates a *distinct* 4-bit hash value (no collisions). No records are deleted during this process. We are told that after the X insertions, 4 buckets have been allocated. (Note that the previous sentence does not refer to the size of the directory.)

What is the minimum possible value of X?

Minimum value of X: 3

(b) In the same scenario as part (a), what is the maximum possible value of X?

Maximum value of X: 8

Problem 5 (10 points)

Consider two relations $R(A, B, C)$ and $S(B, C, D)$. We want to estimate the number of tuples and the size of the following expression: $U = \pi_{ACD}[(\sigma_{A=3 \wedge B=5}R) \bowtie S)]$.

We are given the following information:

- $T(R) = 100000$; $V(R, A) = 20$; $V(R, B) = 50$; $V(R, C) = 150$.
- $T(S) = 5000$; $V(S, B) = 100$; $V(S, C) = 200$; $V(S, D) = 30$.
- All attributes are 10 bytes in size.
- We use the "containment of value sets" assumption.
- We assume query values are selected from values in the relations.

(a) First consider the innermost select $W = \sigma_{A=3 \wedge B=5}R$. Compute the following values.

$T(W) = 100 \qquad S(W) = 30$

$V(W, A) = 1 \qquad V(W, B) = 1$

$V(W, C) = 100$

Explanation: $T(W) = \frac{T(R)}{V(R,A)V(R,B)} = \frac{100000}{20*50} = 100$

Note: $V(W, C)$ is a trick question. The maximum possible value for $V(W, C)$ is $T(W)$.

(b) Next consider the join $Y = W \bowtie S$. Compute the following values.

$T(Y) = 25 \qquad S(Y) = 40$

$V(Y, A) = 1 \qquad V(Y, B) = 1$

$V(Y, C) = 25 \qquad V(Y, D) = 25$

Explanation: $T(Y) = \frac{T(W)T(S)}{max(V(W,B),V(S,B))max(V(W,C),V(S,C))} = \frac{100*5000}{100*200} = 25.$

(c) Finally consider the full expression $U = \pi_{ACD}[(\sigma_{A=3 \wedge B=5}R) \bowtie S)]$. Compute the following values.

$T(U) = 25 \qquad S(U) = 30$

$V(U, A) = 1 \qquad V(U, C) = 25 \qquad V(U, D) = 25$

Problem 1 (10 points)

For parts **(a)**, **(b)**, consider a B+ tree of order n (where n is odd) that has a depth $L > 1$.

(a) What is the minimum number of record pointers the tree can contain?

$$2\left(\frac{n+1}{2}\right)^{L-1}$$

(b) What is the maximum number of record pointers the tree can contain?

$$n(n+1)^{L-1}$$

(c) Imagine you have a B+ tree with 3 levels in which each node has exactly 10 keys. There is a record for each key $1, 2, 3, \ldots, N$, where N is the number of records. How many nodes must be examined to find all records with keys in the range $[95, 134]$?

By following the sequence pointers, we only need to look at one node in each of the first two levels. We only need to look at leaf nodes containing a record in this range, of which there are 5 (Node with record # 91-100, 101-110, 111-120, 121-130, 131-140). This means we must read a total of 1 + 1 + 5 = 7 nodes.

(a) Consider **Expression 1:** $\sigma_B(\sigma_{A\wedge C}(R) \bowtie \sigma_D(S))$.

(a1) Rewrite Expression 1 by pushing the selections as far in as possible (smallest possible join(s)).

$$\begin{aligned}
\sigma_B(\sigma_{A\wedge C}(R) \bowtie \sigma_D(S)) &= \sigma_B(\sigma_C(\sigma_A(R) \bowtie \sigma_D(S))) && (0.1)\\
&= \sigma_B(\sigma_{A\wedge C}(R) \bowtie \sigma_{D\wedge C}(S)) && (0.2)\\
&= \sigma_{A\wedge C\wedge B}(R) \bowtie \sigma_{D\wedge C\wedge B}(S) && (0.3)
\end{aligned}$$

$$\boxed{\sigma_{A\wedge B\wedge C}(R) \bowtie \sigma_{B\wedge C\wedge D}(S)}$$

(a2) Rewrite Expression 1 by pushing the selections as far out as possible (largest possible join(s)).

$\sigma_{A\wedge B\wedge C\wedge D}(R \bowtie S)$

(b) Consider: **Expression 2:** $\sigma_B(\sigma_{A\vee C}(R) \bowtie \sigma_D(S))$.

(b1) Rewrite Expression 2 by pushing the selections as far in as possible (smallest possible join(s)).

$$\begin{aligned}
\sigma_B(\sigma_{A\vee C}(R) \bowtie \sigma_D(S)) &= \sigma_B(\sigma_A(R) \bowtie \sigma_D(S)) \cup \sigma_B(\sigma_C(R) \bowtie \sigma_D(S)) && (0.4)\\
&= (\sigma_{A\wedge B}(R) \bowtie \sigma_{D\wedge B}(S)) \cup (\sigma_{C\wedge B}(R) \bowtie \sigma_{D\wedge B}(S)) && (0.5)\\
&= (\sigma_{A\wedge B}(R) \bowtie \sigma_{D\wedge B}(S)) \cup (\sigma_{C\wedge B}(R) \bowtie \sigma_{D\wedge B\wedge C}(S)) && (0.6)
\end{aligned}$$

$$(\sigma_{A\wedge B}(R) \bowtie \sigma_{B\wedge D}(S)) \cup (\sigma_{B\wedge C}(R) \bowtie \sigma_{B\wedge C\wedge D}(S))$$

(b2) Rewrite Expression 2 by pushing the selections as far out as possible (largest possible join(s)).

$\sigma_{B\wedge(A\vee C)\wedge D}(R \bowtie S)$

(c) Consider **Expression 3:** $\pi_{A,B}(\sigma_C(R) \bowtie S)$.

Rewrite Expression 3 by pushing the selection and projection as far in as possible (smallest possible join(s)).

$$\pi_{A,B}(\sigma_C(R) \bowtie S) = \pi_{A,B}(\sigma_C(R) \bowtie \sigma_C(S)) \quad (0.7)$$
$$= \pi_{A,B}(\sigma_C(R) \bowtie \pi_{B,C}(\sigma_C(S))) \quad (0.8)$$

(can project out D as it is not part of the join or final projected view)

$$= \pi_{A,B}(\sigma_C(R) \bowtie \pi_B(\sigma_C(S))) \quad (0.9)$$

(can project out D as σ_C ensures join condition over C)

$$= \pi_{A,B}(\sigma_C(R)) \bowtie \pi_B(\sigma_C(S)) \quad (0.10)$$

(can project out C from R as join no longer uses C)

Note that performing the last two steps (projecting out C from R, S) without first pushing σ_C to S is incorrect, as it could result in a potentially different logical result.

$$\pi_{A,B}(\sigma_C(R)) \bowtie \pi_B(\sigma_C(S))$$

(b) $Y = \pi_{CD}[(\sigma_{B=3\wedge C=5}R_2)]$

Let's call $P = \sigma_{B=3\wedge C=5}R_2$. We need to calculate $T(Y) = \max(V(P,C), V(P,D))$ but we know $V(P,C) = 1$, so we need $V(P,D)$. By containment of value sets, we have that $V(P,D) = \min(T(P), V(R_2,D))$. But note that $T(P) = \frac{T(R_2)}{V(R_2,B)V(R_2,C)} = 10$.

$$T(Y) = \min(T(P), V(R_2,D)) = 10$$
$$S(Y) = 2 \times 10 = 20$$

(c) $U = \sigma_{D=1}[(\sigma_{A=3\wedge B=5}R_1) \bowtie (\sigma_{C=3\wedge D=5}R_2)]$

Because of $\sigma_{C=3\wedge D=5}$, we must have that $(\sigma_{A=3\wedge B=5}R_1) \bowtie (\sigma_{C=3\wedge D=5}R_2)$ only contains tuples with $D = 5$. Applying $\sigma_{D=1}$ to this relation will yield an empty relation. Therefore,

$$T(U) = 0$$

Problem 5 (10 points)

Consider a hard disk with the following specifications:

- 6000 RPM
- 3.5in in diameter
- 250GB usable capacity
- 100 cylinders, numbered from 1 (innermost) to 100 (outermost).
- Takes time 1+(t/100) milliseconds to move the head t across t cylinders.
- 20% overhead between blocks (gaps).
- Block size is 32 MB.
- transfer rate is 16 GB/sec.
- For this problem 1GB is 10^9 bytes, 1MB is 10^6 bytes.

(a) Based on the specifications, calculate the average rotational delay (in milliseconds) of this disk

The average rotational delay is half of the maximum rotational delay.

$$\frac{1}{2} \times \frac{60 \text{ s}}{6000} = 5 \text{ ms}$$

(b) Suppose that we have just finished reading a block at track 50, and we next want to read a block at track 10. What is the total read time (time for the desired block to appear in memory)? You can ignore the delays we ignored in class.

We will only consider seek time, rotational delay, and transfer time:

$$\left(1 + \frac{50 - 10}{100}\right) + (5) + \left(\frac{32 \times 10^6}{16 \times 10^9} \times 10^3\right) = 8.4 \text{ ms}$$

Problem 1 (10 points)

Consider doing a natural join operation on two relations $R(A, B)$ and $S(B, C)$. Assume that the R tuples are stored contiguously on 200 disk blocks (i.e., $B(R) = 200$), while the S tuples are stored contiguously on 1000 blocks (i.e., $B(S) = 1000$). Each block holds 20 tuples (same for R as for S). There are 51 memory blocks available.

Calculate the I/O cost for each of the following join algorithms. Use an analysis similar to the one used in class. Ignore the I/O cost of writing the final join output to disk. Unless stated otherwise, the tuples in the relations are not sorted.

(a) **Iteration Join Algorithm, R First**. In this case, the algorithm reads 50 R blocks into memory, reads all of S (1 block at a time) into memory, joining with R. The process is repeated until all outputs are generated.

Number of IOs: 4200

200 IOs to read R into memory (50*4), then for each set of 50 R blocks we need 1000 IOs to read in R (1000*4). Therefore, the total is 50*4 + 1000*4 = 4200.

(b) **Merge Join, Sorted Relations**. Assume that both R and S are sorted by B.

Number of IOs: 1200

$B(R) + B(S) = 200 + 1000 = 1200$

(c) **Merge Join, Unsorted Relations**. Assume R and S are not sorted. First sort R by B using merge-sort, writing out all tuples in R into a sorted file. Then sort S by B in a similar fashion. Finally, do the merge-join on the sorted relations.

Number of IOs: 6000

Each tuple is read, written, read written. The first read, written is to sort each chunk, the second read, written is to merge sort all of the chunks.

Sort cost R: 4*200 = 800.

Sort cost S: 4*1000 = 4000.

Total cost: sort cost R + sort cost S + join cost = 800 + 4000 + 1200 = 6000

(d) **Index Join**. Assume that there is an index on the B column of S and that the index fits in memory. Furthermore, assume that on average each R tuple matches 4 S tuples. With this strategy, we read a block of R and for each tuple r of R in this block we use the index to find all matching tuples $\{s_1, ..., s_m\}$ of S. Each of these S tuples is read into memory and joined with r. We repeat the process for all R blocks.

Number of IOs: 16200

(4000 tuples in R * 4 matching tuples in S per R tuple) + 200 IOs to read in R = 16200

(e) **Hash Join**. We first hash tuples from R (using the B attribute) into 50 buckets. (No R buckets are kept in memory.) We then hash S into 50 buckets in a similar fashion. Then we join each pair of R_i and S_i buckets.

Number of IOs: 3600

Read and write R to hash, read and write S to hash, then read all buckets to join. That is a read, write, and read for each block in R and S.

Total cost = 3*($B(R)$ + $B(S)$) = 3*(200 + 1000) = 3600

(f) **Hybrid Hash Join**. We first hash tuples from R (using the B attribute) into 20 buckets (i.e., buckets $R_1, R_2, ..., R_{20}$). and keep 3 R buckets in memory (i.e., buckets R_1, R_2, R_3). We then hash S into 20 buckets in a similar fashion (i.e., buckets $S_1, S_2, ..., S_{20}$) but join buckets S_1, S_2, S_3 immediately with buckets R_1, R_2, R_3. Then we join each pair of R_i and S_i buckets for the remaining 17 buckets.

Number of IOs: 3240

Bucketize R = read R + (write 17 buckets * 10 blocks per bucket) = 200 + 17*10 = 370

Bucketize S, only write 17 buckets = read S + (write 17 buckets * 50 blocks per bucket) = 1000 + 17*50 = 1850

Compare join (3 buckets already done) = read 17*10 + 17*50 = 1020

Total cost = 370 + 1850 + 1020 = 3240