Database Systems (H) Assessed Coursework

Team 41:
Karlis Siders (2467273S), Donald MacKenzie (2426230m), Declan McBride (2399448m)

Task 1

1.A

bfr = 5 employees/block
M = 5 buckets
r = 1000 employees

b = r / bfr = 1000 / 5 = 200 blocks

b_b = 200 / 5 = 40 blocks per bucket

<u>Part 1</u>
Method 1:

Average block accesses = $5 \cdot 0.2 \cdot \sum_{i=1}^{40} 0.025i = 20.5$

Method 2 (valid because the data is uniformly distributed):
Worst case (search all 40 blocks): 5(0.2 x 40) = 40 block accesses
Best case (found on first check): 5(0.2 x 1) = 1 block access
Average case: (40+1)/2 = **20.5** block accesses

<u>Part 2</u>
Avg = 5 (0.2 log2(40)) ≈ 5.32 block accesses
Binary search can be used in each linked list, which is why log2 is taken of the elements in the list.

1.B

<u>Part 1</u>
avg = 40 * 5 = **200** block accesses
Because the database cannot know when it has found all departments matching *y* (unless specified in some header/piece of metadata), it will always continue searching until it has exhausted all of the blocks, making 200 the worst and best case (thus, also the average case).

<u>Part 2</u>
Algorithm:

1. Choose a bucket (5 * 0.2)
2. Because the tuples are sorted, binary search can be used (log2(40))
3. The other 9 employees (in 2 blocks on average) can be found surrounding the found tuple (ceil(9/5) = 2)
4. The edges of the 10 employees need to be checked to know that all employees have been found (1 block on average)

Avg = 5 * 0.2 * (log2(40) + 2 + 1) = **8.32** block accesses

Task 2

An Employee tuple = 32 bytes and a Data Block of 64 bytes => **2 tuples per block**

There are 10^8 tuples => 10^8/2 blocks = **5 · 10^7 blocks**

A Tax code attribute V = 5 bytes and a pointer is P = 10 bytes. We know that an Internal Node has *p tree-pointers* and *p-1 key values*. Therefore, an Internal Node will be of:

> 10p + 5(p - 1) bytes
> = **15p - 5 bytes**

A block is 64 bytes, therefore the order of the Internal Node will be the floor value of:

> 15p - 5 = 64
> 15p = 69
> p = 4.6
> floor(p) = 4

Order of Internal Node is p = 4

This means that an internal node will have **(p) 4 tree pointers and (p-1) 3 key values.**

A Leaf Node of order pL contains: **pL key values, pL block pointers and 1 pointer** to the next leaf node in the tree. Therefore, a Leaf Node will be of:

> 5 pL + 10pL + 10 bytes
> = **15pL + 10 bytes**

A block is 64 bytes; therefore, the order of the Leaf Node will be the floor value of:

> 15pL + 10 = 64
> 15pL = 54
> pL = 3.6
> floor(pL) = 3

Order of Leaf Node is pL = 3

This means that a leaf node will have **(pL) 3 key values, (pL) 3 block pointers and 1 pointer to the next leaf node in the tree** (except for the last leaf node, which won't have the additional pointer).

Each leaf node in the tree is **100% full of Tax code values**, so we will need:

Level	Key values	Tree pointers	Leaf node pointers
0	3	4	N/A (potentially 12)
1	4 · 3	4^2	N/A (potentially 48)
2	4^2 · 3	4^3	N/A (potentially 192)
3	4^3 · 4	4^4	N/A (potentially 768)
4	4^4 · 3	4^5	N/A (potentially 3072)
5	4^5 · 3	4^6	12288

Therefore, a B+ tree with 6 levels to store all 12288 TaxCodes.

SQL3: SELECT * FROM EMPLOYEE WHERE TaxCode <= x

Keys are gathered *only* in the B+ Tree leaf nodes, therefore to access the key values you must first take the leftmost path to access the lowest key value.

For a 6 level B+ Tree this will require **t = 6 block accesses.**

The leaf nodes are *linked-list and sorted by key*, so travel right along this data structure until the condition (TaxCode <= x) is no longer valid.

The TaxCode values are uniformly distributed and each node contains 3 data pointers, therefore finding all values that satisfy the condition requires **q = x/3 block accesses**

Per Data block we require access to:

$10^8/12288 \cdot x$ data blocks

Per TaxCode there are **$10^8/12288$ employee tuples.** Each index block of 64 bytes can hold 5 data block pointers ($5 \cdot 10$ bytes) and 1 block pointer to the next index block (10 bytes)

Therefore, for x Tax codes, we need to access:

$10^8/12288 \cdot x/5$ index blocks

Let **$w = x/12288$** be our range ratio

Cost = $t + q + (10^8/12288 \cdot x) + (10^8/12288 \cdot x/5)$

$= 6 + x/3 + (10^8/12288 \cdot x) + (10^8/12288 \cdot x/5)$

$= 6 + (12288w/3) + (10^8/12288 \cdot 12288w) + (10^8/12288 \cdot 12288w/5)$

$C1 = 6 + 4096w + (10^8/5)w + 10^8w$

As there are $5 \cdot 10^7$ blocks, a linear search would require **$C2 = 5 \cdot 10^7$ block accesses**. Therefore for a B+ tree to be more efficient than a linear search:

$$C1 < C2$$
$$6 + 4096w + (10^8/5)w + 10^8w < 5 \cdot 10^7$$

Therefore,

$w < 0.417$

IF range ratio is less than 41.7%, **THEN** use B+ Tree

ELSE use linear scan

Task 3

3.A

p = 30 tree pointers per internal node
pL = 5 pointers per leaf node
r = 4500 employees
n = 4500 distinct SSN values

s = r/n = 1 employee per SSN
Leaf nodes = n / pL = 900

Structure:
Root: 1 node, with 30 tree pointers
L1: 30 nodes, with 900 leaf node pointers
Leaf: 900 nodes, each storing 5 SSN values => 5*900 = 4500 values)

- Each leaf node containing 5 data pointers and 1 sibling pointer

=> Level t = 3

MIN(SALARY) algorithm:

1. Load root node (1 block access)
2. Move to the lowest value Level 1 node and load it (1 block access)
3. Move to the lowest value leaf node and load it (1 block access)
4. Access each data block and note the lowest salary found (5 block accesses)
5. Load the sibling data block (1 block access)
6. Access each data block and update lowest salary if a salary is found that is lower than the current lowest salary (5 block accesses)
7. Repeat step 5 and 6 until all leaf nodes have been checked (6 * 898 block accesses)

Expected cost: 5402 block accesses

AVG(SALARY) algorithm:

1. Load root node (1 block access)
2. Move to the lowest value Level 1 node and load it (1 block access)
3. Move to the lowest value leaf node and load it (1 block access)
4. Access each data block. Track the sum of each salary for each employee accessed, also track the number of employees accessed (5 block accesses)
5. Load the sibling data block (1 block access)
6. Repeat step 4 and 5 until all leaf nodes have been checked (6 * 899) block

Expected cost: 5402 block accesses

SQL Query expected cost:
Keep track of all variables used in AVG and MIN, which makes the search 5402 block accesses.

3.B

p = 5 tree pointers per internal node
pL = 5 pointers per leaf node
r = 4500 employees
n = 625 distinct SALARY values

s = r/n = 7.2 employees/salary
Leaf nodes = n / pL = 125

Structure:

Root: 1 node (p = 5)
L1: 5 nodes
L2: 25 nodes
Leaf: 125 nodes (each with 5 SALARY values => 5*125 = 625 values)

- Each leaf node pointing to 5 blocks of block-pointers
- Each block contains 8 block-pointers
- => 625*8 = 5000 data-blocks (8 data-blocks per SALARY)

=> Level t = 4

MIN(SALARY) algorithm:

1. Go to smallest value in every level of B+ tree
2. The first leaf node in the order will be the smallest one.

Expected cost = t + 1 = 5 block accesses (because each node fits in 1 block and "uniformly distributed" means that 7/8 employees will have even the first SALARY)

AVG(SALARY) algorithm:

1. Go to smallest value in every level of B+ tree
2. Start from smallest SALARY value in the blocks of block-pointers
3. Count how many block-pointers for each SALARY, going up to the biggest SALARY value, while also keeping sum of all distinct SALARIES
4. Divide sum of SALARIES by count of block-pointers

Expected cost: t+1 + 624 = 629 block accesses

SQL Query expected cost:
MIN algorithm can be combined in AVG by checking first data-block: t+1 + 624 = 629 block accesses