CARNEGIE MELLON UNIVERSITY
COMPUTER SCIENCE DEPARTMENT
15-445/645 – DATABASE SYSTEMS (SPRING 2024)
PROF. JIGNESH PATEL

Homework #2 (by Shivang and Avery)

Due: **Friday February 16, 2024 @ 11:59pm**

IMPORTANT:

- Enter all of your answers into **Gradescope by 11:59pm on Friday February 16, 2024.**
- **Plagiarism**: Homework may be discussed with other students, but all homework is to be completed **individually**.

For your information:

- Graded out of **100** points; **4** questions total
- Rough time estimate: ≈4-6 hours (1-1.5 hours for each question)

Revision : 2024/02/05 15:08

Question	Points	Score
Storage Models	24	
Linear Hashing and Cuckoo Hashing	24	
Extendible Hashing	20	
B+Tree	32	
Total:	100	

Question 1: Storage Models ....[24 points]

Consider a database with a single table `T(song_id,song_name,artist_id,duration number_of_streams)`, where `song_id` is the *primary key*, and all attributes are the same fixed width. Suppose `T` has 10,000 tuples that fit into 500 pages, Ignore any additional storage overhead for the table (e.g., page headers, tuple headers). Additionally, you should make the following assumptions:

- The DBMS does *not* have any additional meta-data (e.g., sort order, zone maps).
- `T` does *not* have any indexes (including for primary key `song_id`).
- None of `T`'s pages are already in the buffer pool. The DMBS has an infinite buffer pool.
- Content-wise, the tuples of `T` will always make each query run the longest possible and do the most page accesses.
- The tuples of `T` can be in any order (keep this in mind when computing *minimum* versus *maximum* number of pages that the DBMS will potentially have to read and think of all possible orderings)

(a) Consider the following query:

```
SELECT MAX(number_of_streams) FROM T
    WHERE duration > 100 AND artist_id == 123321 ;
```

i. **[3 points]** Suppose the DBMS uses the decomposition storage model (DSM) with implicit offsets. How many pages will the DBMS potentially have to read from disk to answer this query? (Keep in mind our assumption about the contents of `T`!)
☐ 1-100 ☐ 101-200 ☐ 201-300 ☐ 301-500 ☐ ≥ 501 ☐ Not possible to determine

ii. **[3 points]** Suppose the DBMS uses the N-ary storage model (NSM). How many pages will the DBMS potentially have to read from disk to answer this query? (Keep in mind our assumption about the contents of `T`!)
☐ 1-100 ☐ 101-200 ☐ 201-300 ☐ 301-500 ☐ ≥ 501 ☐ Not possible to determine

(b) Now consider the following query:

```
SELECT song_name, artist_id FROM T
    WHERE song_id = 15445 OR song_id = 15645 OR song_id = 15721;
```

i. Suppose the DBMS uses the decomposition storage model (DSM) with implicit offsets.

α) **[3 points]** What is the *minimum* number of pages that the DBMS will potentially have to read from disk to answer this query?
☐ 1 ☐ 2-4 ☐ 5-100 ☐ 101-200 ☐ 201-500 ☐ ≥ 501

☐ Not possible to determine

β) **[3 points]** What is the *maximum* number of pages that the DBMS will potentially have to read from disk to answer this query?
☐ 1 ☐ 2-4 ☐ 5-100 ☐ 101-200 ☐ 201-500 ☐ $\geq$ 501
☐ Not possible to determine

ii. Suppose the DBMS uses the N-ary storage model (NSM).

α) **[3 points]** What is the *minimum* number of pages that the DBMS will potentially have to read from disk to answer this query?
☐ 1 ☐ 2-4 ☐ 5-100 ☐ 101-200 ☐ 201-500 ☐ $\geq$ 501
☐ Not possible to determine

β) **[3 points]** What is the *maximum* number of pages that the DBMS will potentially have to read from disk to answer this query?
☐ 1 ☐ 2-4 ☐ 5-100 ☐ 101-200 ☐ 201-500 ☐ $\geq$ 501
☐ Not possible to determine

(c) Finally consider the following query:

```
SELECT song_id, number_of_streams FROM T
    WHERE duration = (SELECT MIN(duration) FROM T);
```

Suppose the DBMS uses the decomposition storage model (DSM) with implicit offsets.

i. **[3 points]** What is the *minimum* number of pages that the DBMS will potentially have to read from disk to answer this query?
☐ 1 ☐ 2-5 ☐ 100-200 ☐ 201-299 ☐ $\geq$ 300 ☐ Not possible to determine

ii. **[3 points]** What is the *maximum* number of pages that the DBMS will potentially have to read from disk to answer this query?
☐ 1 ☐ 2-5 ☐ 100-200 ☐ 201-299 ☐ $\geq$ 300 ☐ Not possible to determine

Question 2: Linear Hashing and Cuckoo Hashing . . . . [24 points]

Consider the following linear probe hashing schema:

1. The table have a size of 4 slots, each slot can only contain one key value pair.
2. The hashing function is
   $h_1(\texttt{x}) = \texttt{x \% 4}$.
3. When there is conflict, it finds the next free slot to insert key value pairs.
4. The original table is empty.

(a) Insert key value pair (3, A) and (4, B). (for (3,A), 3 is the key and A is the value) Select the value in each entry of the resulting table.

i. **[1 point]** Entry 0 (key % 4 = 0) ☐ A ☐ B ☐ Empty
ii. **[1 point]** Entry 1 (key % 4 = 1) ☐ A ☐ B ☐ Empty
iii. **[1 point]** Entry 2 (key % 4 = 2) ☐ A ☐ B ☐ Empty
iv. **[1 point]** Entry 3 (key % 4 = 3) ☐ A ☐ B ☐ Empty

(b) After the changes from part **(a)**, insert key value pair (8, C) and then (9, D). Select the value in each entry of the resulting table.

i. **[1 point]** Entry 0 (key % 4 = 0) ☐ A ☐ B ☐ C ☐ D ☐ Empty
ii. **[1 point]** Entry 1 (key % 4 = 1) ☐ A ☐ B ☐ C ☐ D ☐ Empty
iii. **[1 point]** Entry 2 (key % 4 = 2) ☐ A ☐ B ☐ C ☐ D ☐ Empty
iv. **[1 point]** Entry 3 (key % 4 = 3) ☐ A ☐ B ☐ C ☐ D ☐ Empty

Consider the following cuckoo hashing schema:

1. Both tables have a size of 4.
2. The hashing function of the first table returns the fourth and third least significant bits: $h_1(\texttt{x}) = \texttt{(x >> 2) \& 0b11}$.
3. The hashing function of the second table returns the least significant two bits: $h_2(\texttt{x}) = \texttt{x \& 0b11}$.
4. When inserting, try table 1 first.
5. When replacement is necessary, first select an element in the second table.
6. The original entries in the table are shown in the figure below.

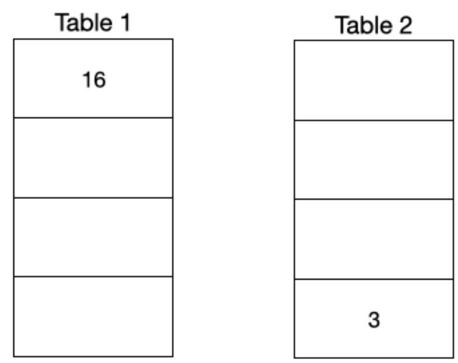


Figure 1: Initial contents of the hash tables.

(a) Insert key 32 and then delete 16. Select the value in each entry of the resulting two tables.

i. Table 1

α) **[1 point]** Entry 0 (`0b00`) ☐ 32 ☐ 16 ☐ Empty
β) **[1 point]** Entry 1 (`0b01`) ☐ 32 ☐ 16 ☐ Empty
γ) **[1 point]** Entry 2 (`0b10`) ☐ 32 ☐ 16 ☐ Empty
δ) **[1 point]** Entry 3 (`0b11`) ☐ 32 ☐ 16 ☐ Empty

ii. Table 2

α) **[1 point]** Entry 0 (`0b00`) ☐ 32 ☐ 3 ☐ Empty
β) **[1 point]** Entry 1 (`0b01`) ☐ 32 ☐ 3 ☐ Empty
γ) **[1 point]** Entry 2 (`0b10`) ☐ 32 ☐ 3 ☐ Empty
δ) **[1 point]** Entry 3 (`0b11`) ☐ 32 ☐ 3 ☐ Empty

(b) After the changes from part **(a)**, insert key 27, then delete 3, then insert key 8. Select the value in each entry of the resulting two tables.

i. Table 1

α) **[1 point]** Entry 0 (`0b00`) ☐ 8 ☐ 32 ☐ 27 ☐ Empty

β) **[1 point]** Entry 1 (`0b01`) ☐ 8 ☐ 32 ☐ 27 ☐ Empty

γ) **[1 point]** Entry 2 (`0b10`) ☐ 8 ☐ 32 ☐ 27 ☐ Empty

δ) **[1 point]** Entry 3 (`0b11`) ☐ 8 ☐ 32 ☐ 27 ☐ Empty

ii. Table 2

α) **[1 point]** Entry 0 (`0b00`) ☐ 32 ☐ 8 ☐ 27 ☐ Empty

β) **[1 point]** Entry 1 (`0b01`) ☐ 32 ☐ 8 ☐ 27 ☐ Empty

γ) **[1 point]** Entry 2 (`0b10`) ☐ 32 ☐ 8 ☐ 27 ☐ Empty

δ) **[1 point]** Entry 3 (`0b11`) ☐ 32 ☐ 8 ☐ 27 ☐ Empty

Question 3: Extendible Hashing .... [20 points]

Consider an extendible hashing structure such that:

- Each bucket can hold up to two records.
- The hashing function uses the lowest g bits, where g is the global depth.
- A new extendible hashing structure is initialized with $g = 0$ and one empty bucket
- If multiple keys are provided in a question, assume they are inserted one after the other from left to right.

(a) Starting from an empty table, insert keys 0, 3.

i. **[1 point]** What is the global depth of the resulting table?
☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ None of the above

ii. **[1 point]** What is the local depth of the bucket containing 0?
☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ None of the above

(b) Starting from the result in **(a)**, you insert keys 23, 33.

i. **[2 points]** What is the global depth of the resulting table?
☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ None of the above

ii. **[1 point]** What is the local depth of the bucket containing 0?
☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ None of the above

(c) Starting from the result in **(b)**, you insert keys 13, 27.

i. **[2 points]** What is the global depth of the resulting table?
☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ None of the above

ii. **[2 points]** What is the local depth of the bucket containing 13?
☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ None of the above

iii. **[1 point]** What is the local depth of the bucket containing 3?
☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ None of the above

iv. **[1 point]** Which value, if inserted, will hash to the same bucket as the bucket containing key 27?
☐ 43 ☐ 4 ☐ 21 ☐ 13 ☐ None of the above

(d) **[3 points]** Starting from the result in **(c)**, which **key(s)**, if inserted next, will **not** cause a split?
☐ 35 ☐ 9 ☐ 17 ☐ 41 ☐ 16 ☐ None of the above

(e) **[3 points]** Starting from the result in **(c)**, which **key(s)**, if inserted next, will cause a split and increase the table's global depth?
☐ 0 ☐ 2 ☐ 51 ☐ 9 ☐ 1 ☐ None of the above

(f) **[3 points]** Starting from an empty table, insert keys 64, 128, 256, 512. What is the global depth of the resulting table?
☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐ 8 ☐ ≥ 9

Question 4: B+Tree . . . . [32 points]

Consider the following B+tree.

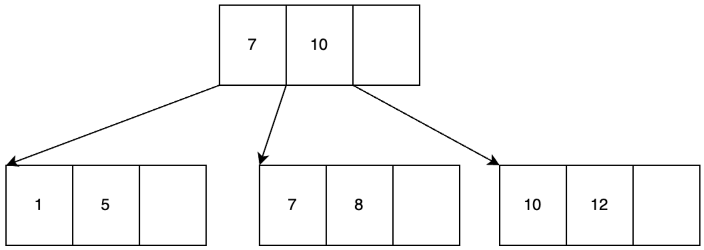


Figure 2: B+ Tree of order $d = 4$ and height $h = 2$.

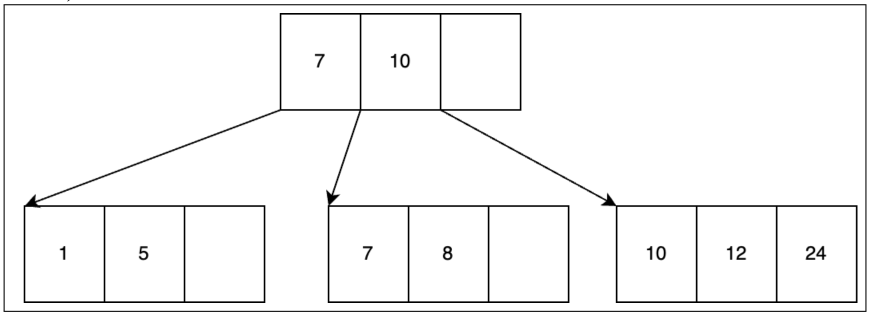
When answering the following questions, be sure to follow the procedures described in class and in your textbook. You can make the following assumptions:

- A left pointer in an internal node guides towards keys $<$ than its corresponding key, while a right pointer guides towards keys $\geq$.
- A leaf node underflows when the number of **keys** goes below $\lceil \frac{d-1}{2} \rceil$.
- An internal node underflows when the number of **pointers** goes below $\lceil \frac{d}{2} \rceil$.

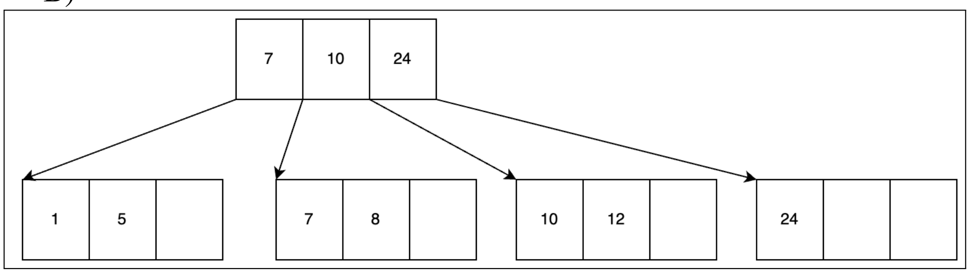
Note that B+ tree diagrams for this problem omit leaf pointers for convenience. The leaves of actual B+ trees are linked together via pointers, forming a singly linked list allowing for quick traversal through all keys.

(a) **[4 points]** Insert 24^* into the B+tree. Select the resulting tree.

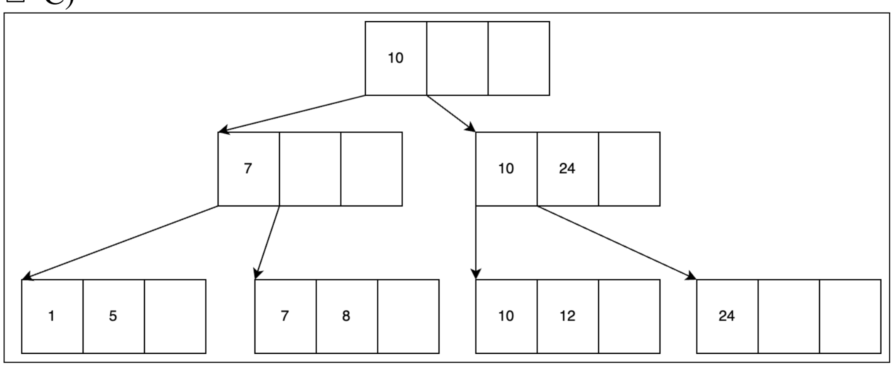
☐ A)



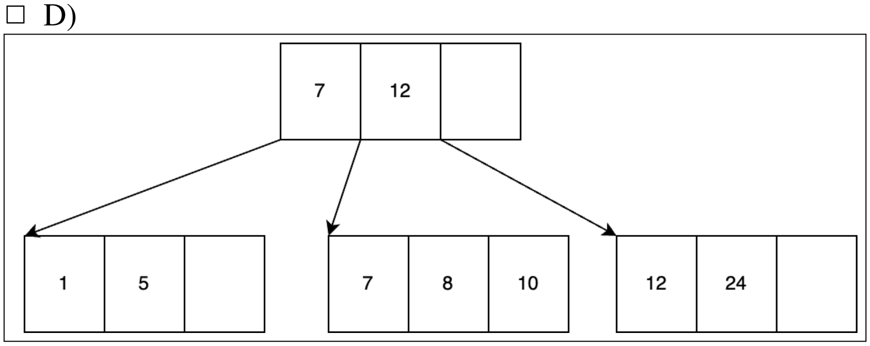
☐ B)



☐ C)

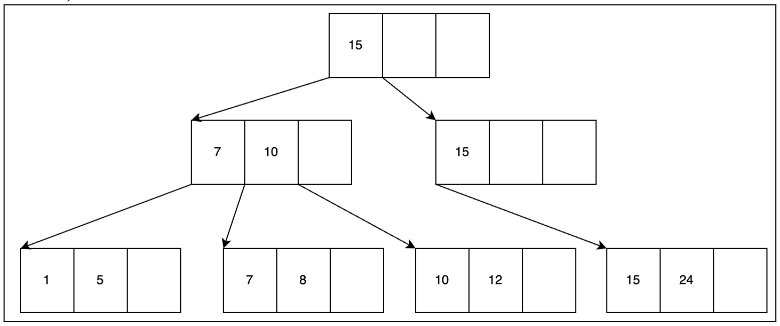


☐ D)

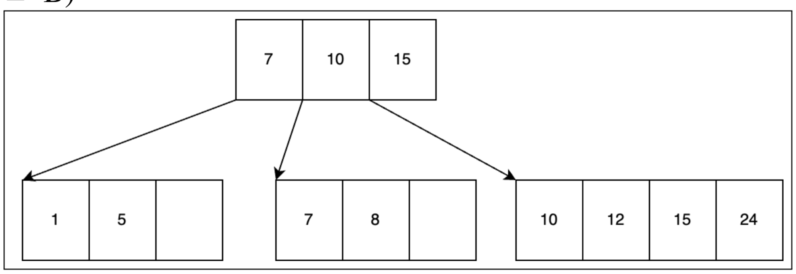


(b) **[4 points]** Starting with the tree that results from **(a)**, insert 15^*. Select the resulting tree.

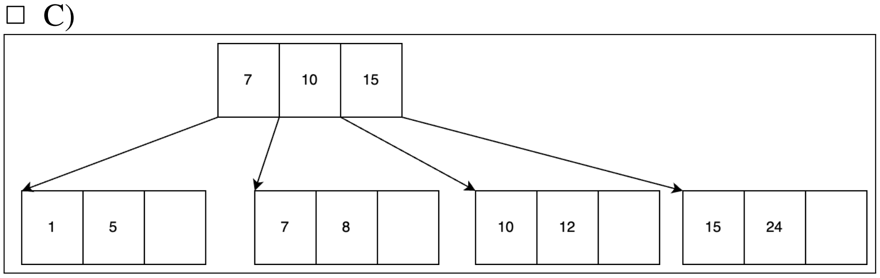
☐ A)



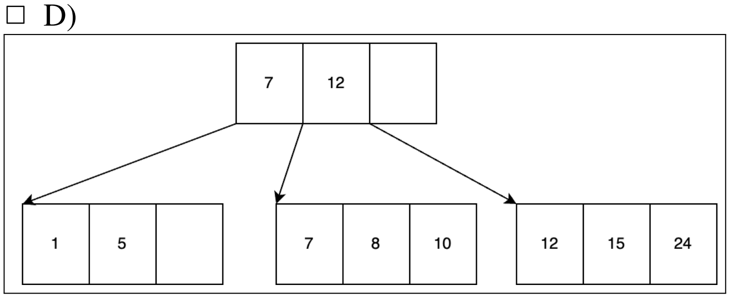
☐ B)



☐ C)

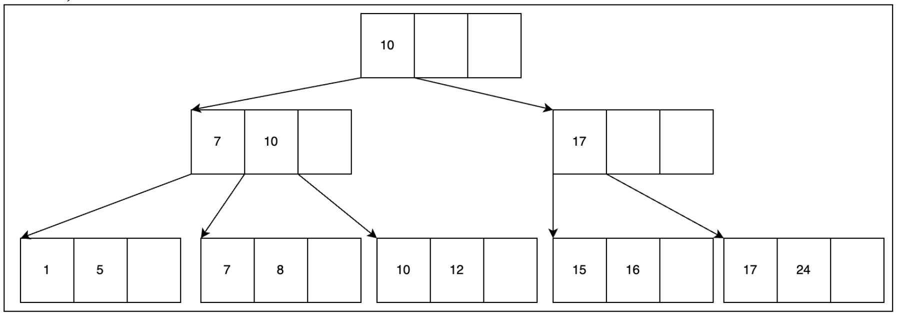


☐ D)

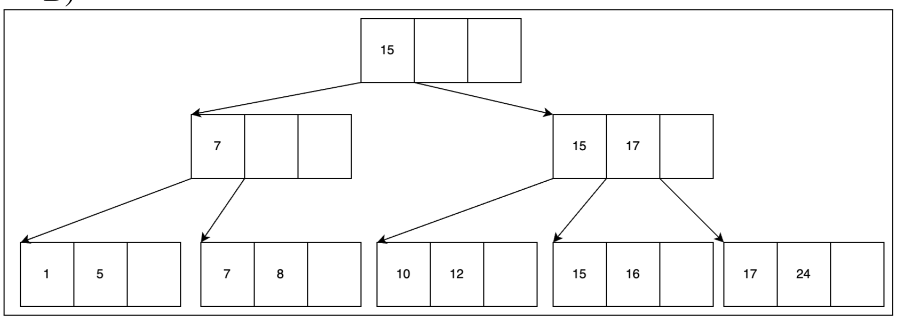


(c) **[5 points]** Starting with the tree that results from **(b)**, insert 16^* and then insert 17^*. Select the resulting tree (the result after inserting 17^*).

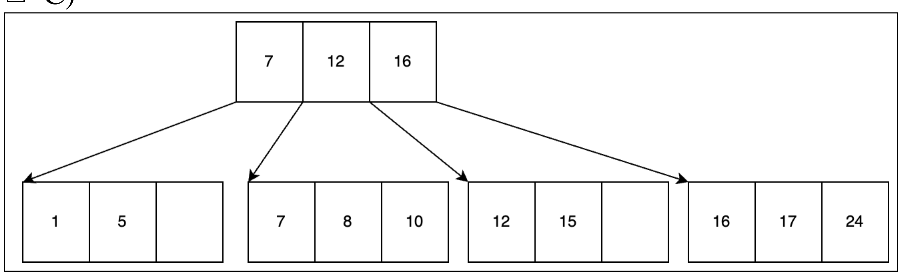
☐ A)



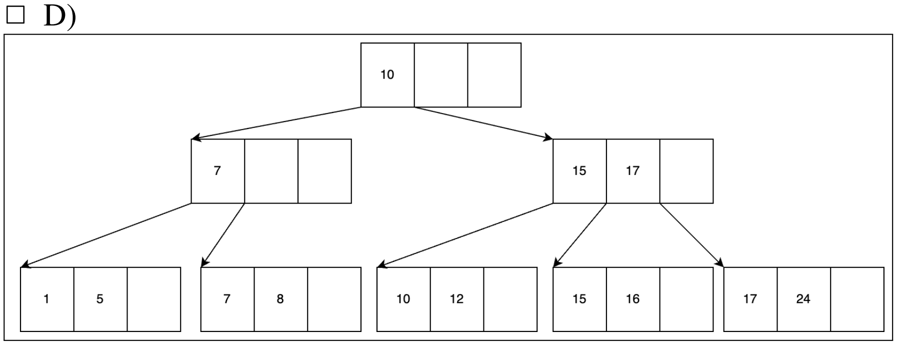
☐ B)



☐ C)

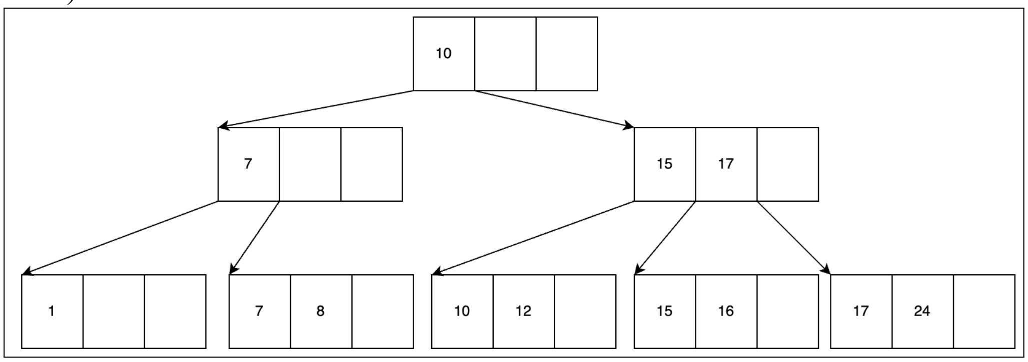


☐ D)

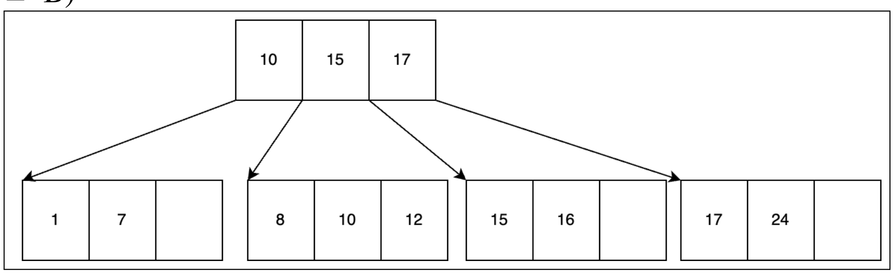


(d) **[5 points]** Starting with the tree that results from **(c)**, deletes 5^*. Select the resulting tree.

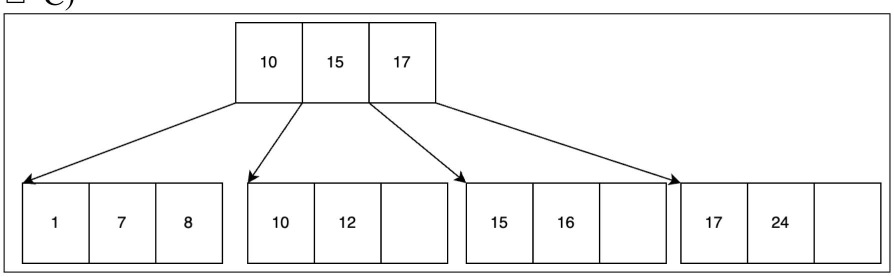
☐ A)



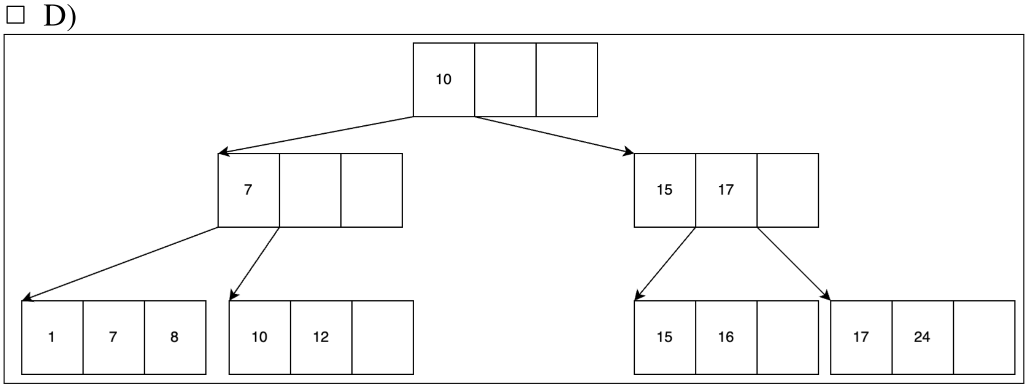
☐ B)



☐ C)



☐ D)



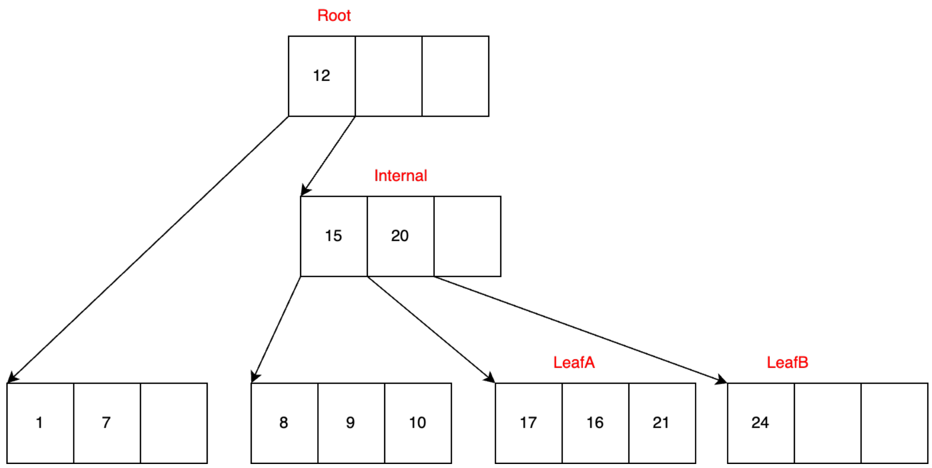


Figure 3: B+tree with violations

(e) The B+Tree shown in Figure 3 is invalid. That is, its nodes violate the correctness properties of B+Trees that we discussed in class. If the tree is invalid, select all the properties that are violated for each node. If the node is valid, then select 'None'. There will be **no** partial credit for missing violations.

Note:

- *If a node's subtrees are not the same height, the balance property is violated at that node only.*
- *If a node's subtrees contain values not in the range specified by the node's separator keys (e.g. internal node has separator keys as 5,10,18, then the first pointer pointed leaf node values should be less than 5, the second one should be in the range of [5, 10), etc. The logic also applies to any upper level internal node or root node. If the values in the leaf node satisfies its parent node's separator key property but violates the root node's, mark root node as separator key violation and not its parent node.), the separator keys property is violated at that node.*

i. **[2 points]** Which properties are violated by **Leaf A**?
☐ Key order property ☐ Half-full property ☐ Balance property
☐ Separator keys ☐ None

ii. **[2 points]** Which properties are violated by **LeafB**?
☐ Key order property ☐ Half-full property ☐ Balance property
☐ Separator keys ☐ None

iii. **[2 points]** Which properties are violated by **Internal Node**?
☐ Key order property ☐ Half-full property ☐ Balance property
☐ Separator keys ☐ None

iv. **[2 points]** Which properties are violated by **Root**?
☐ Key order property ☐ Half-full property ☐ Balance property
☐ Separator keys ☐ None

Question 4 continues. . .

(f) i. **[2 points]** A read-only thread needs to hold at most one latches at the same time.
☐ True ☐ False

ii. **[2 points]** A write thread needs to hold at most two write latches at the same time
☐ True ☐ False

iii. **[2 points]** A write thread must release its latches in the order they were acquired (i.e., FIFO) to prevent concurrency errors.
☐ True ☐ False