CARNEGIE MELLON UNIVERSITY
COMPUTER SCIENCE DEPARTMENT
15-445/645 – DATABASE SYSTEMS (SPRING 2023)
PROF. CHARLIE GARROD

Homework #2 (by Arvin Wu)

Due: **Friday February 17, 2023 @ 11:59pm**

IMPORTANT:

- Enter all of your answers into **Gradescope by 11:59pm on Friday February 17, 2023.**
- **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 : 2023/02/07 15:29

Question	Points	Score
Storage Models	16	
Cuckoo Hashing	27	
Extendible Hashing	22	
B+Tree	35	
Total:	100	

Question 1: Storage Models .... [16 points]

Consider a database with a single table `T(course_id,course_name,instructor,class_size,hrs_per_week)`, where `course_id` is the *primary key*, and all attributes are the same fixed width. Suppose `T` has 5,000 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 `course_id`)
- None of `T`'s pages are already in the buffer pool.
- Content-wise, the tuples of `T` will make each query run the longest possible (this assumption is critical for solving part **(a)**)
- The tuples of `T` can be in any order (this assumption is critical for solving part **(b)** when you compute the *minimum* versus *maximum* number of pages that the DBMS will potentially have to read)

(a) Consider the following query:

```
SELECT MAX(hrs_per_week) FROM T
    WHERE class_size > 10;
```

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

100+100=200

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

500 pages

(b) Now consider the following query:

```
SELECT course_name, instructor, class_size FROM T
    WHERE course_id = 15445 OR course_id = 15645;
```

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

course_id is primary key, so max_num is 2.

1 to find two keys.

3 to find other attributes.

☐ 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 ☐ ≥ 501

☐ Not possible to determine

106 { 100 to find 2 keys.
6 to find other attributes.

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

α) **[2 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

1

β) **[2 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 ☐ ≥ 501

☐ Not possible to determine

500

Question 2: Cuckoo Hashing . . . . [27 points]

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.

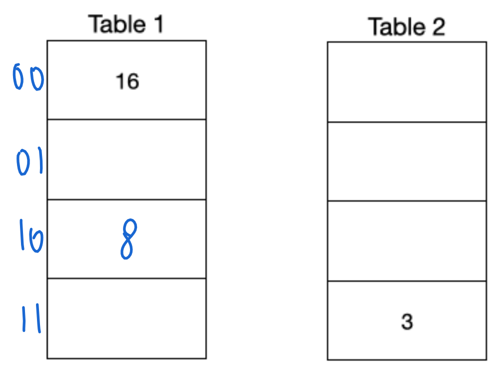


Figure 1: Initial contents of the hash tables.

(a) **[1 point]** Select the sequence of insert operations that results in the initial state.
☑ Insert 16, insert 3 ☐ Insert 3, insert 16 ☐ None of the above

16 = 0b10000

$h_1(16) = 100 \ \& \ 11 = 00$

$h_1(3) = 00 \ \& \ 11 = 00$.

$h_2(3) = 11 \ \& \ 11 = 11$.

(b) Insert key 8 and then delete 16. Select the value in each entry of the resulting two tables.

8 = 0b1000

$h_1(8) = 10 \& 11 = 10$

i. Table 1

α) **[1 point]** Entry 0 (`0b00`) ☐ 3 ☐ 8 ☑ Empty

β) **[1 point]** Entry 1 (`0b01`) ☐ 3 ☐ 8 ☑ Empty

γ) **[1 point]** Entry 2 (`0b10`) ☐ 3 ☑ 8 ☐ Empty

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

ii. Table 2

α) **[1 point]** Entry 0 (`0b00`) ☐ 3 ☐ 8 ☑ Empty

β) **[1 point]** Entry 1 (`0b01`) ☐ 3 ☐ 8 ☑ Empty

γ) **[1 point]** Entry 2 (`0b10`) ☐ 3 ☐ 8 ☑ Empty

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

(c) After the changes from part **(b)**, insert key 27 and then insert 4. Select the value in each entry of the resulting two tables.

i. Table 1

α) **[1 point]** Entry 0 (`0b00`) ☑ 3 ☐ 8 ☐ 27 ☐ 4 ☐ Empty

β) **[1 point]** Entry 1 (`0b01`) ☐ 3 ☐ 8 ☐ 27 ☑ 4 ☐ Empty

γ) **[1 point]** Entry 2 (`0b10`) ☐ 3 ☑ 8 ☐ 27 ☐ 4 ☐ Empty

δ) **[1 point]** Entry 3 (`0b11`) ☐ 3 ☐ 8 ☐ 27 ☐ 4 ☑ Empty

ii. Table 2

α) **[1 point]** Entry 0 (`0b00`) ☐ 3 ☐ 8 ☐ 27 ☐ 4 ☑ Empty

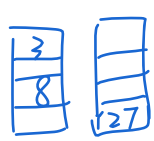
β) **[1 point]** Entry 1 (`0b01`) ☐ 3 ☐ 8 ☐ 27 ☐ 4 ☑ Empty

γ) **[1 point]** Entry 2 (`0b10`) ☐ 3 ☐ 8 ☐ 27 ☐ 4 ☑ Empty

δ) **[1 point]** Entry 3 (`0b11`) ☐ 3 ☐ 8 ☑ 27 ☐ 4 ☐ Empty

$h_1(27) = h_1(0b11011) = 110 \& 11 = 10$

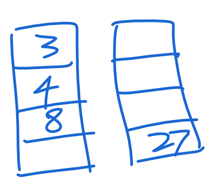
16+8+3



$h_2(27) = 11$

$h_1(3) = 00$

$h_1(4) = h_1(0b100) = 01 \& 11 = 01$



(d) After the changes from parts **(b)** and **(c)**, insert key 19 and then delete 27. Select the value in each entry of the resulting two tables.

i. Table 1

α)	**[1 point]**	Entry 0 (`0b00`)	☑ 3	☐ 8	☐ 4	☐ 19	☐ Empty
β)	**[1 point]**	Entry 1 (`0b01`)	☐ 3	☐ 8	☑ 4	☐ 19	☐ Empty
γ)	**[1 point]**	Entry 2 (`0b10`)	☐ 3	☐ 8	☐ 4	☐ 19	☑ Empty
δ)	**[1 point]**	Entry 3 (`0b11`)	☐ 3	☐ 8	☐ 4	☐ 19	☑ Empty

ii. Table 2

α)	**[1 point]**	Entry 0 (`0b00`)	☐ 3	☑ 8	☐ 4	☐ 19	☐ Empty
β)	**[1 point]**	Entry 1 (`0b01`)	☐ 3	☐ 8	☐ 4	☐ 19	☑ Empty
γ)	**[1 point]**	Entry 2 (`0b10`)	☐ 3	☐ 8	☐ 4	☐ 19	☑ Empty
δ)	**[1 point]**	Entry 3 (`0b11`)	☐ 3	☐ 8	☐ 4	☑ 19	☐ Empty

(e) **[2 points]** What is the smallest key that potentially causes an infinite loop given the table that results from part **(d)**?

☐ 33 ☐ 34 ☑ 35 ☐ 36 ☐ 37 ☐ 51 ☐ None of the above

(d) $h_1(19) = h_1(0b10011) = 100 \& 11 = 00$
$16+3$

$h_2(19) = 11$

3	8
4	
	19

$h_1(27) = 10$

$h_2(8) = 00$

(e) $h_1(33) = 0$ $\quad h_2(33) = 01$ ×
$32+1$
100001

$h_1(34) = 0 \quad h_2(34) = 10$ ×

$h_1(35) = 0 \quad h_2(35) = 11$

Solution: 3, 19, 35 all have $h_1(x) = 0$
$h_2(x) = 3$, which would continue
to replace each other.

Question 3: Extendible Hashing . . . . [22 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

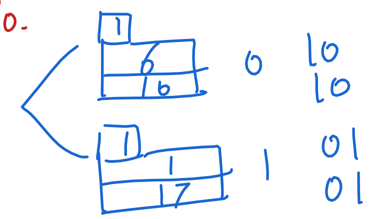
(a) Starting from an empty table, insert keys 6, 1. 110 0

No split has occurred yet because the first bucket (on initialization) can hold 2 arbitrary values. Thus global depth is same as its initial value of 0.

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 6?
☑ 0 ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ None of the above

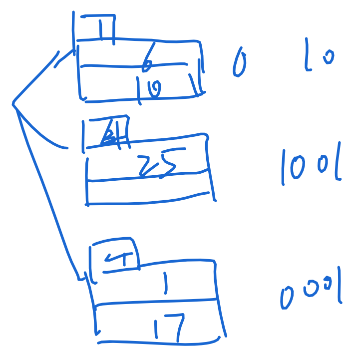
(b) Starting from the result in **(a)**, you insert keys 17, 10. 10001 1010



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 10?
☐ 0 ☑ 1 ☐ 2 ☐ 3 ☐ 4 ☐ None of the above

(c) Starting from the result in **(b)**, you insert key 25. 11001 16+8+1



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

ii. **[3 points]** What is the local depth of the bucket containing 25?
☐ 0 ☐ 1 ☐ 2 ☐ 3 ☑ 4 ☐ None of the above

iii. **[2 points]** What is the local depth of the bucket containing 17?
☐ 0 ☐ 1 ☐ 2 ☐ 3 ☑ 4 ☐ None of the above

iv. **[1 point]** What is the local depth of the bucket containing 6?
☐ 0 ☑ 1 ☐ 2 ☐ 3 ☐ 4 ☐ None of the above

v. **[1 point]** Which value, if inserted, will hash to the same bucket as the bucket containing key 10?
☑ 4 ☐ 7 ☐ 9 ☐ 13 ☐ None of the above

(d) **[2 points]** Starting from the result in **(c)**, which **key(s)**, if inserted next, will cause a split that doubles the table's size? 6→12
☐ 5 ☐ 12 ☐ 24 ☐ 29 ☑ 33 ☐ None of the above

101 1100 16+8+4+1 10000 1

(e) **[2 points]** Starting from the result in **(c)**, which **key(s)**, if inserted next, will cause a split **without** doubling the table's size? double.
☐ 5 ☑ 12 ☑ 24 ☐ 29 ☐ 33 ☐ None of the above

no split 11101 no split

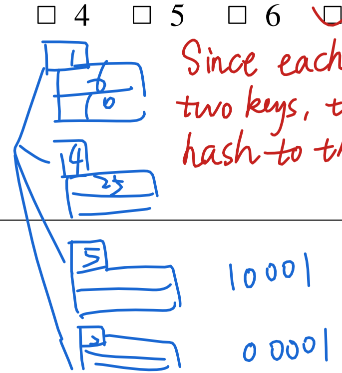
(f) **[3 points]** Starting from an empty table, insert keys 32, 64, 128, 256. What is the global depth of the resulting table?
☐ 4 ☐ 5 ☐ 6 ☑ 7 ☑ 8 ☐ ≥9

$2^5\ 2^6\ 2^7\ 2^8$

Since each buckets can hold at most two keys, three or more keys can-not hash to the same bucket. g=7

2^6

$\{ 2^7, 2^8$



Question 4: B+Tree .... [35 points]

Consider the following B+tree.

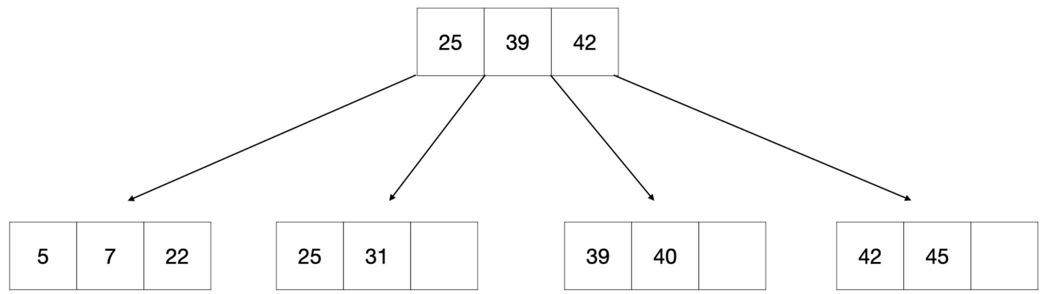


Figure 2: B+ Tree of order $d = 4$ and height $h = 2$.

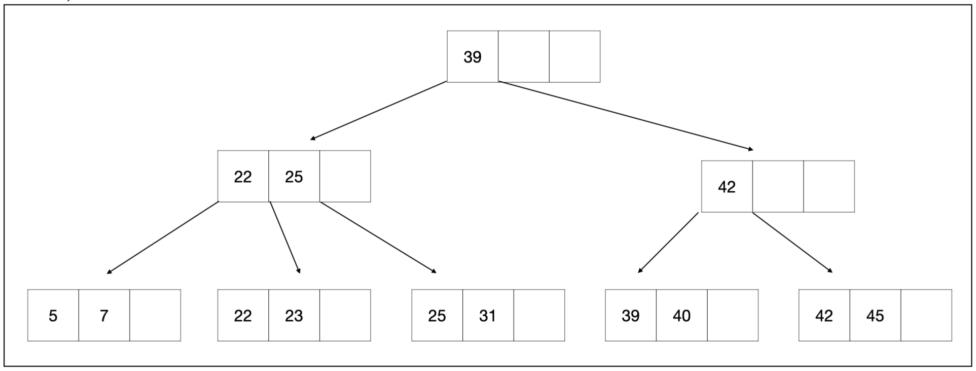
When answering the following questions, be sure to follow the procedures described in class and in your textbook. You can make the following assumptions:

- 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 $\lceil \frac{d}{2} \rceil$.

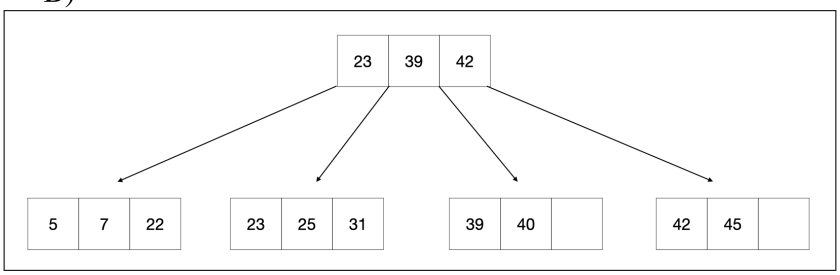
Note that B+ tree diagrams for this problem omit leaf pointers for convenience. The leaves of actual B+ trees are linked together via pointers, forming a singly linked list allowing for quick traversal through all keys.

(a) **[5 points]** Insert 23* into the B+tree. Select the resulting tree.

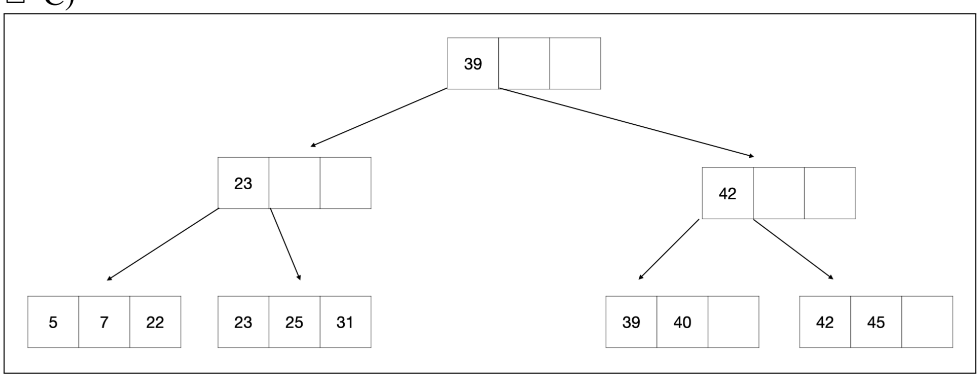
☐ A)



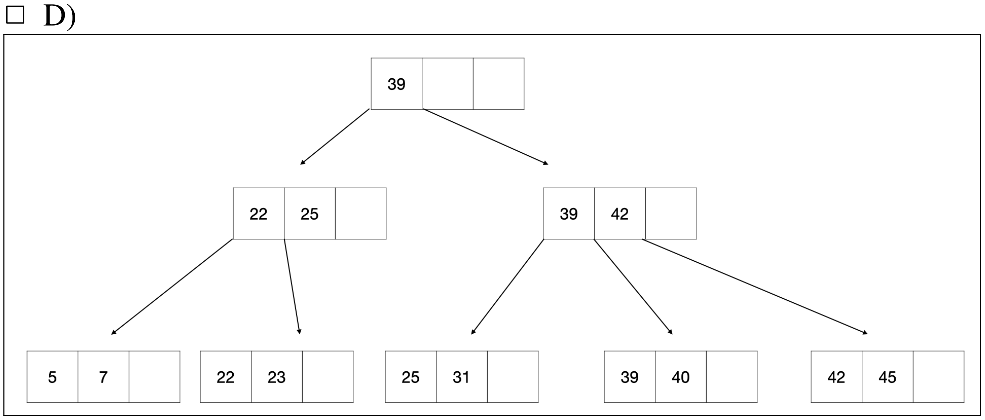
☐ B)



☐ C)

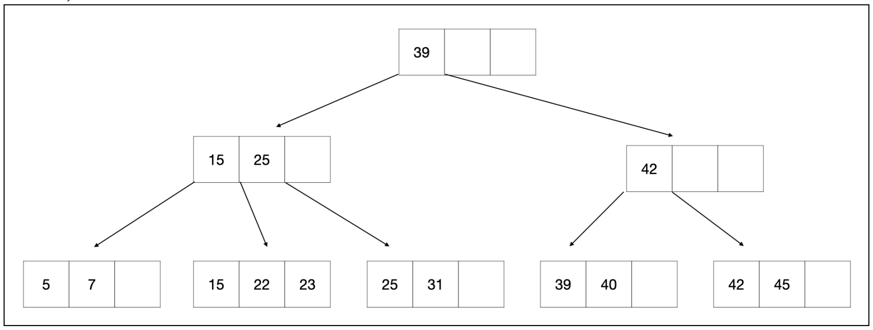


☐ D)

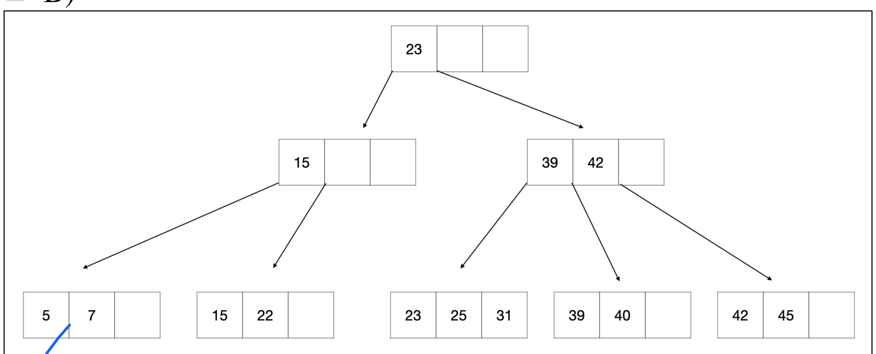


(b) **[5 points]** Starting with the tree that results from **(a)**, insert 15^*. Select the resulting tree.

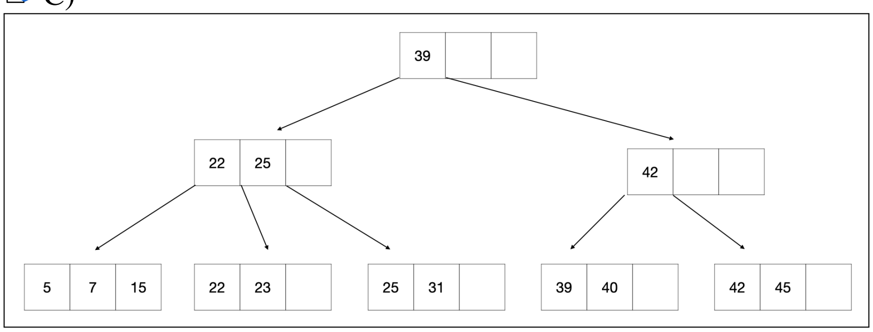
☐ A)



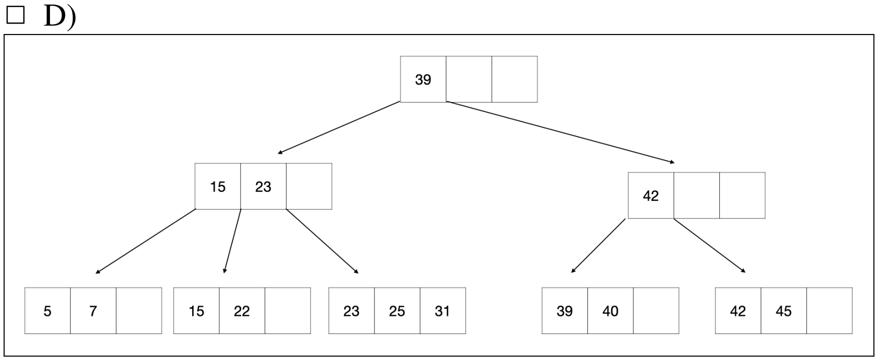
☐ B)



☐ C)

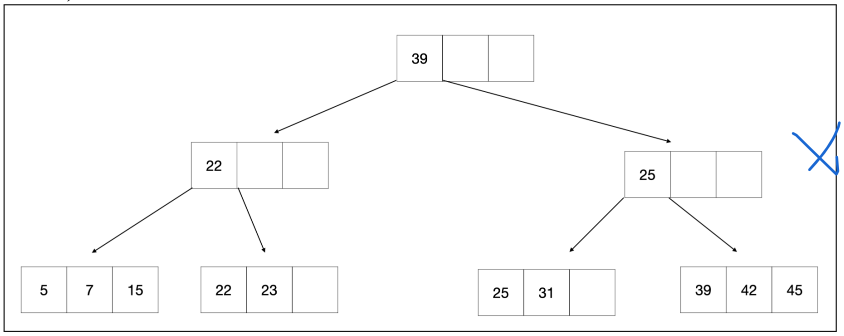


☐ D)

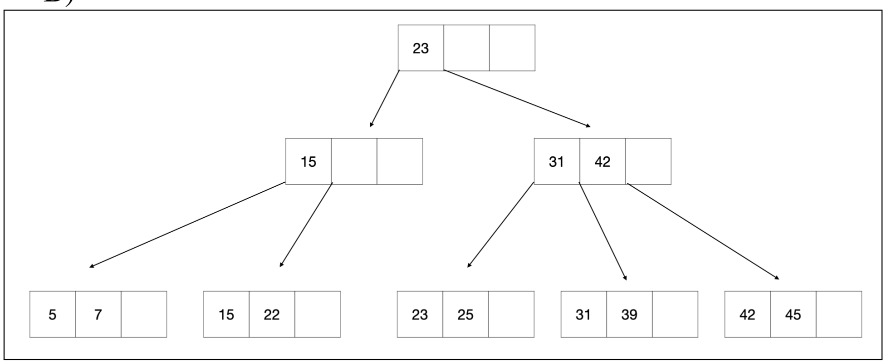


(c) **[5 points]** Starting with the tree that results from **(b)**, delete 40^*. Select the resulting tree.

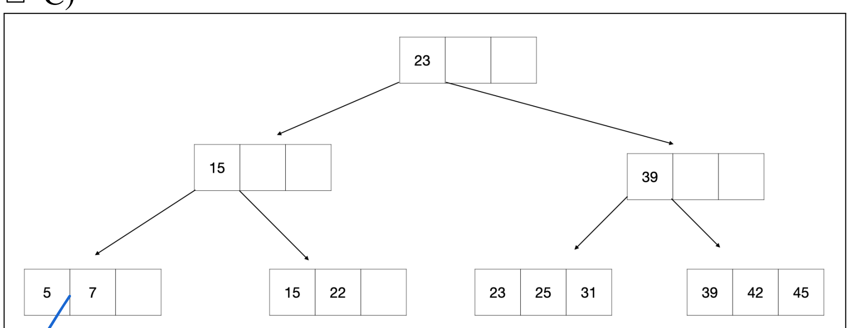
☐ A)



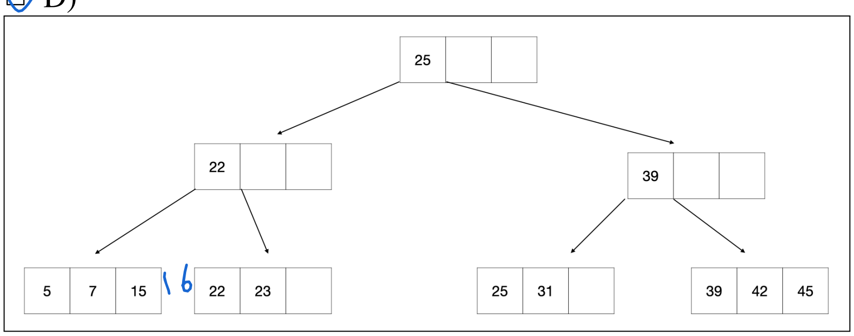
☐ B)



☐ C)

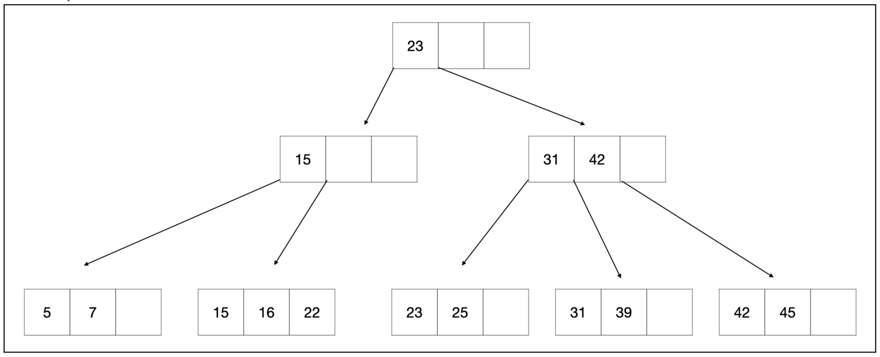


☑ D)

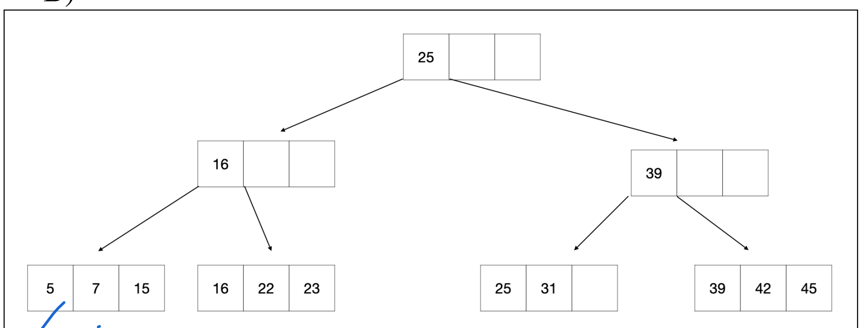


(d) **[5 points]** Starting with the tree that results from **(c)**, insert 16^*. Select the resulting tree.

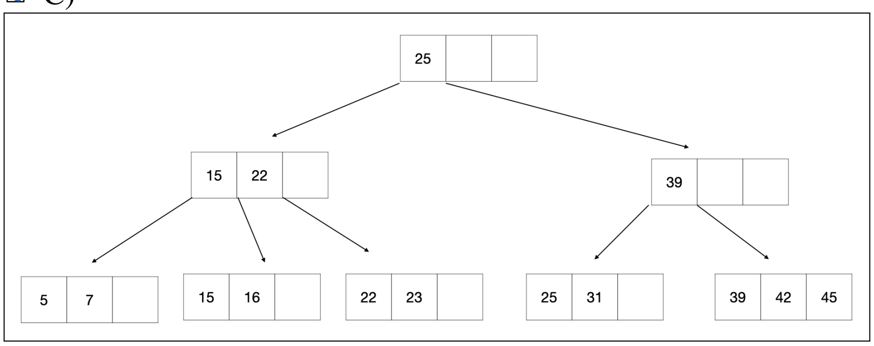
☐ A)



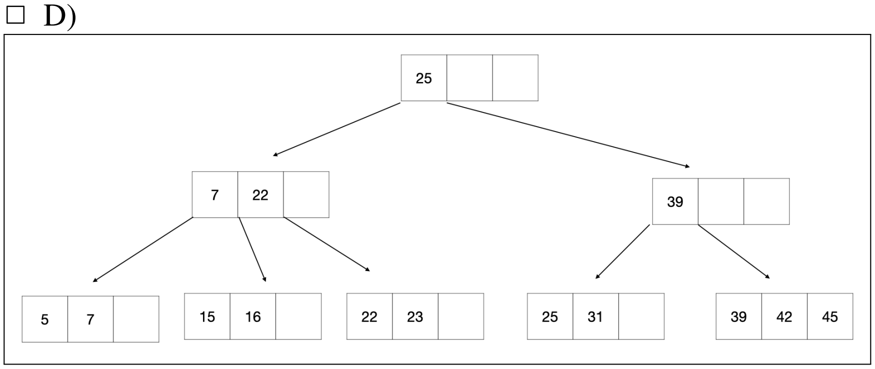
☐ B)



☑ C)

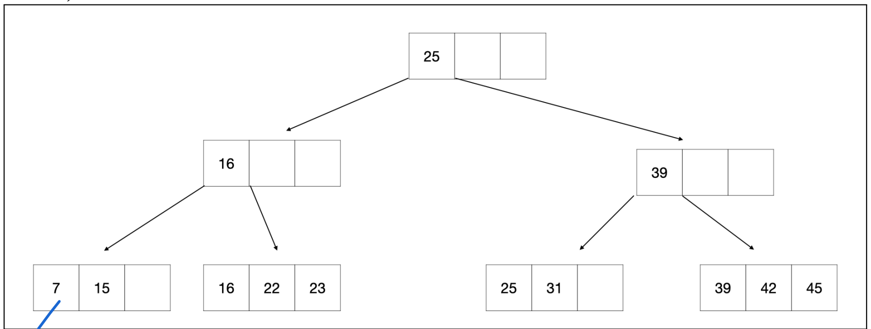


☐ D)

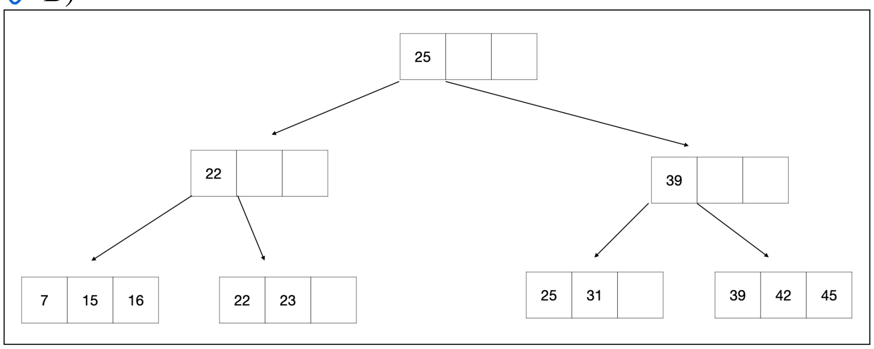


(e) **[5 points]** Finally, starting with the tree that results from **(d)**, delete 5^*. Select the resulting tree.

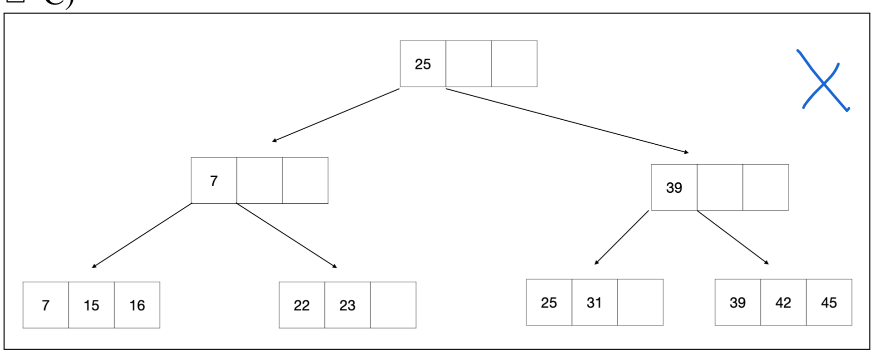
☐ A)



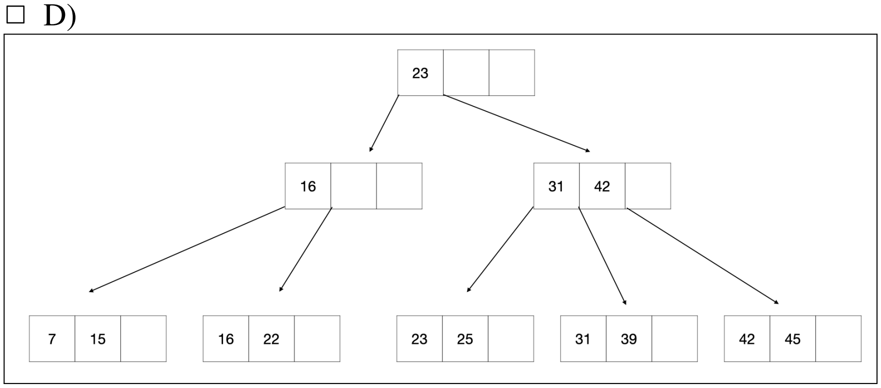
☐ B)



☐ C)



☐ D)



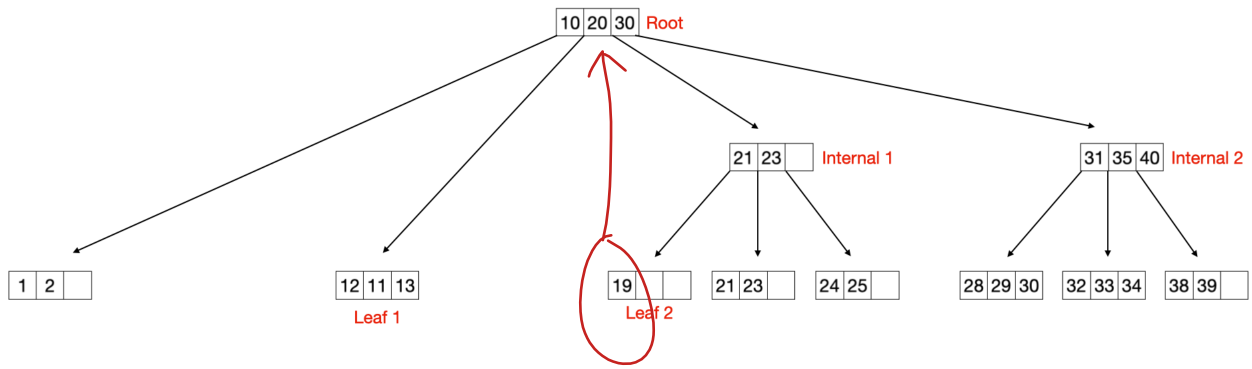


Figure 3: B+tree with violations

(f) 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, the separator keys property is violated at that node.*

i. **[2 points]** Which properties are violated by **Leaf 1**?
☑ Key order property ☐ Half-full property ☐ Balance property
☐ Separator keys ☐ None

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

iii. **[2 points]** Which properties are violated by **Internal Node 1**?
☐ Key order property ☐ Half-full property ☐ Balance property
☑ Separator keys ☐ None

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

There are 3 separator keys, but only 3 children pointers (should be 4).

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

The root is inbalanced, as it has both Leaf 1 and Internal Node 1 as its children. The root's subtree containing Leaf 2 contains the value 19, violating the root's separater keys.