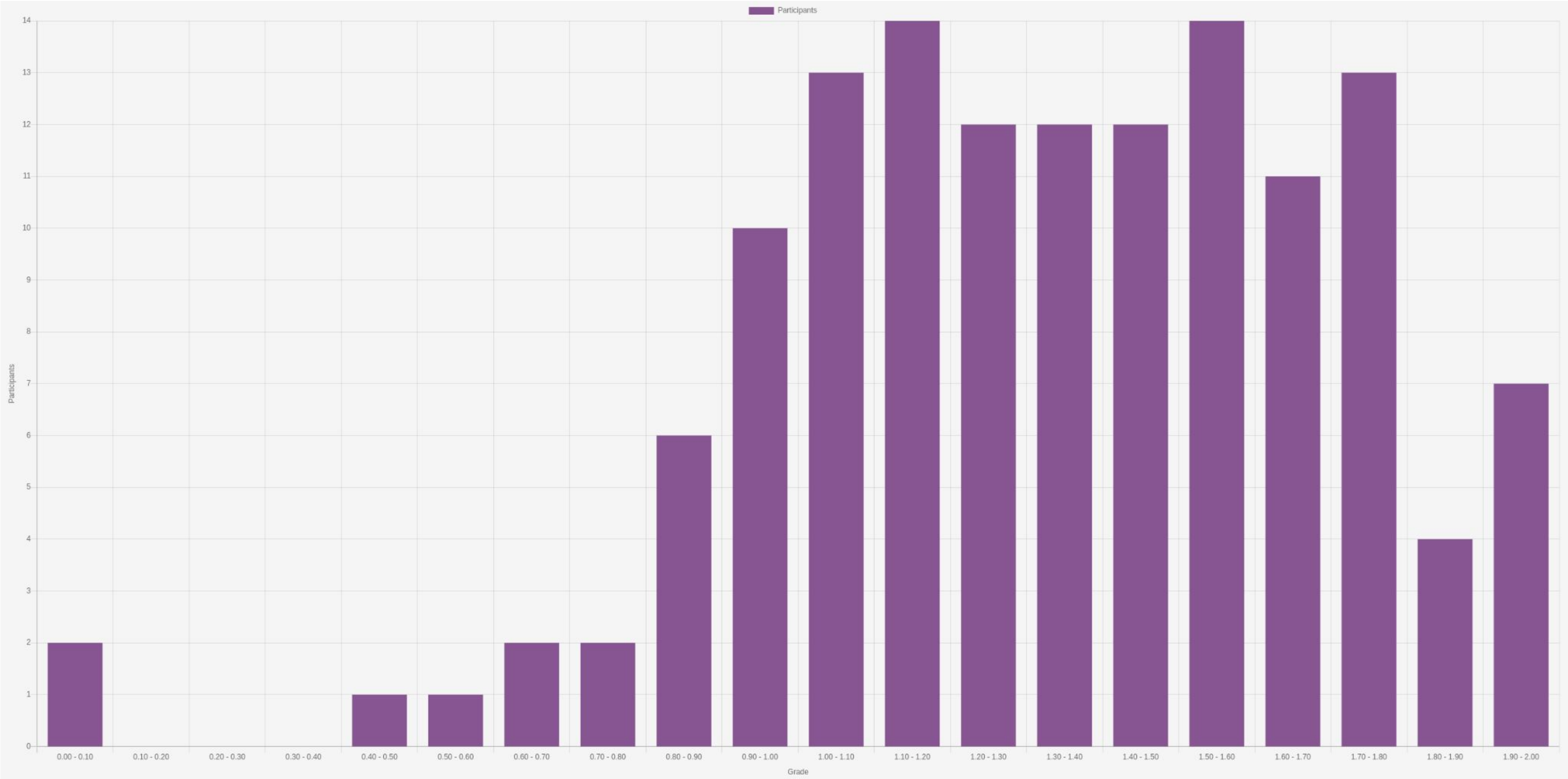
Database Technology

Exercise 4: Review


Participants
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
Participants
0.00 - 0.10
0.10 - 0.20
0.20 - 0.30
0.30 - 0.40
0.40 - 0.50
0.50 - 0.60
0.60 - 0.70
0.70 - 0.80
0.80 - 0.90
0.90 - 1.00
1.00 - 1.10
1.10 - 1.20
1.20 - 1.30
1.30 - 1.40
1.40 - 1.50
1.50 - 1.60
1.60 - 1.70
1.70 - 1.80
1.80 - 1.90
1.90 - 2.00
Grade

Q1: In a B+ tree, all paths from the root to a leaf have the same length.

True

Q2: Select the type of queries which are efficiently supported by a B+ tree index on an attribute ***a***.

True	False	
✓		Range queries over the attribute ***a***.
	✗	Point queries for a single value of another attribute.
	✗	Range queries over another attribute.
✓		Point queries for a single value of the attribute ***a***.

Q3(a): How many I/O operations does it take to execute a range query on “a” using the B+ tree that returns 320 records?

Given the following properties of a disk, a clustered data file that is sorted on an attribute ***a***, and a B+ tree index over the attribute ***a***.

Disk properties:

- The block size is 32768 bytes.

Data file properties:

- The data file contains 100000 tuples.
- The size of record representing a tuple is 512 bytes.
- There are no block headers.

Index properties of a dense B+ tree on the attribute a.

- The average fill rate of the nodes (except the root) of the B+ tree is 70%.
- The size of a search key is 4 bytes.
- The size of a pointer is 20 bytes.
- The root (and only the root) of the B+ tree is cached in main memory.

Number of Tuples Per Block = Floor(32768/512) = 64 Tuples per block
Total blocks to represent 320 tuples = Ceil(320/64) = 5 block
Num of Index tuples = 100000
1 Index Tuple size = 24 bytes
TotalIndexTuplePerBlock= Floor((BlockSize-SizeOfThePointerToTheNextBlock)/IndexTupleSize)
= Floor((32768-20)/24) = 1364
Total Index Tuple with 70% fill rate per block = Ceil(1364*.7) = 955
Total blocks needed to represent Index Tuple = Ceil(100000/955) = 105
Total Leafs nodes in B+ tree = 105
Intermediate B+ nodes required to represent 105 leaf nodes = Ceil(105/955) = 1
We can therefore represent all keys for 105 leaf nodes in 1 node that can be a root node as well.
Depth of the Tree = 1

Total I/O Operations:
Condition: when the first record located at the beginning of the bock
1 I/O to access the B+ tree node pointing to the first record satisfying the range criteria.
5 I/O to fetch the blocks representing 320 records satisfying the range.
1 I/O to fetch the block representing record that doesn’t satisfy the upper bound of the range and terminates the query.

Condition: when the first record located at the end of the bock
1 I/O to access the B+ tree node pointing to the first record satisfying the range criteria.
6 I/O to fetch the blocks representing 320 + some additional records that do not satisfy the range.

Total I/O operation = 7

Q3(b): How many I/O operations does it take to execute a range query on “a” using the B+ tree that returns 40 records?

Given the following properties of a disk, a clustered data file that is sorted on an attribute **a**, and a B+ tree index over the attribute **a**.

disk properties:

- The block size is 16384 bytes.

Data file properties:

- The data file contains 400 million tuples.
- The size of record representing a tuple is 8192 bytes.
- There are no block headers.

Index properties of a dense B+ tree on the attribute **a**.

- The average fill rate of the nodes (except the root) of the B+ tree is 70%.
- The size of a search key is 8 bytes.
- The size of a pointer is 12 bytes.
- The root (and only the root) of the B+ tree is cached in main memory.

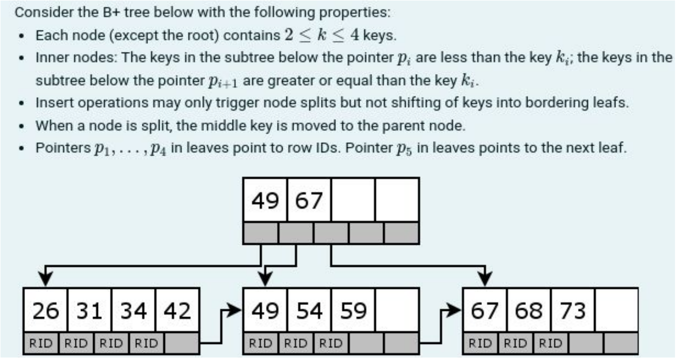
Number of Tuples Per Block = Floor(16384/8192) = 2 Tuples per block
Total blocks to represent 40 tuples = Ceil(40/2) = 20 block
Num of Index tuples = 400X10^6
1 Index Tuple size = 8+12 =20 bytes
TotalIndexTuplePerBlock= Floor((BlockSize-SizeOfThePointerToTheNextBlock)/IndexTupleSize)
= Floor((16384-12)/20) = 818
Total Index Tuple with 70% fill rate per block = Ceil(818*.7) = 573
Total blocks needed to represent Index Tuple = Ceil(400*10^6/573) = 698081
Total B+ Leafs nodes = 698081
Intermediate B+ nodes required to represent 698081 leaf nodes = Ceil(698081/573) = 1219
Intermediate B+ nodes required to represent 1219 intermediate nodes = Ceil(1219/573) = 3
Intermediate B+ nodes required to represent 3 intermediate nodes = Ceil(3/573) = 1
Therefore, Depth of the Tree = 3

Total I/O Operations:
Condition: when the first record located at the beginning of the bock
3 I/O to access the B+ tree node pointing to the first record satisfying the range criteria.
20 I/O to fetch the blocks representing 40 records satisfying the range.
1 I/O to fetch the block representing record that doesn’t satisfy the upper bound of the range and terminates the query.

Condition: when the first record located at the end of the bock
3 I/O to access the B+ tree node pointing to the first record satisfying the range criteria.
21 I/O to fetch the blocks representing 40 + some additional records that do not satisfy the range.

Total I/O operation = 24

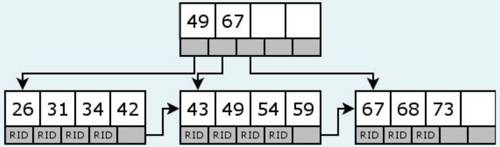
Q4: Which of the following choices represents the B+-Tree after insertion of the key 43?



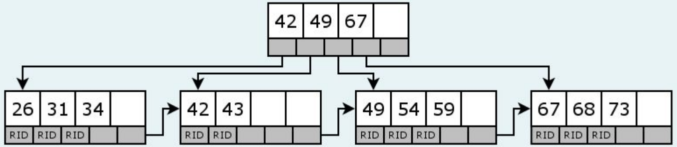
Consider the B+ tree below with the following properties:

- Each node (except the root) contains $2 \leq k \leq 4$ keys.
- Inner nodes: The keys in the subtree below the pointer p_i are less than the key k_i; the keys in the subtree below the pointer p_{i+1} are greater or equal than the key k_i.
- Insert operations may only trigger node splits but not shifting of keys into bordering leafs.
- When a node is split, the middle key is moved to the parent node.
- Pointers $p_1, \ldots, p_4$ in leaves point to row IDs. Pointer p_5 in leaves points to the next leaf.

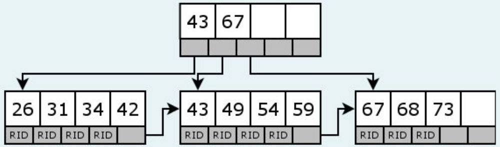
○ a.



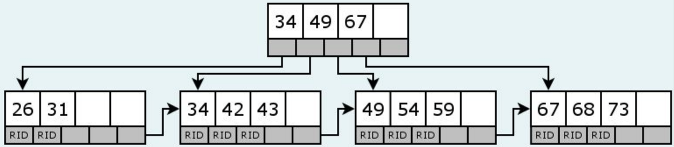
○ b.



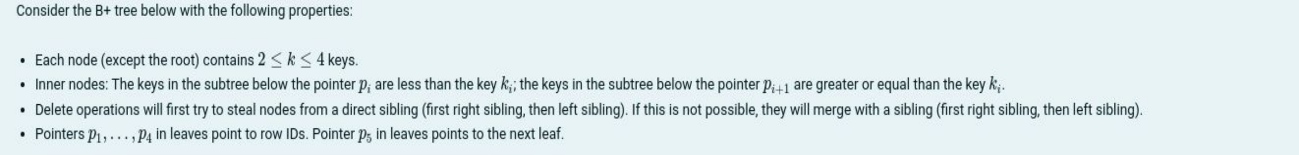
○ c.



[✓ ○ d.]

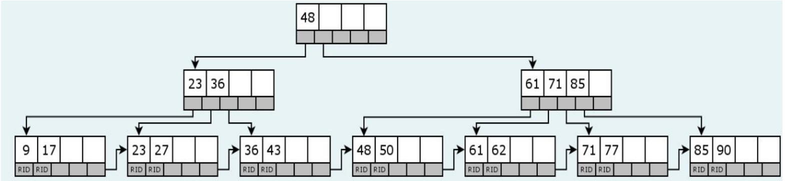


Q5: Which of the following choices represents the B+ tree after deletion of the key 36?

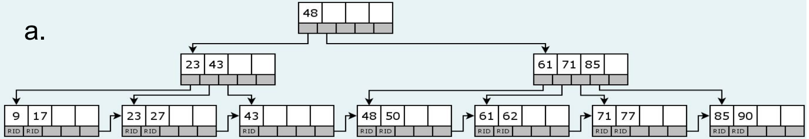


Consider the B+ tree below with the following properties:

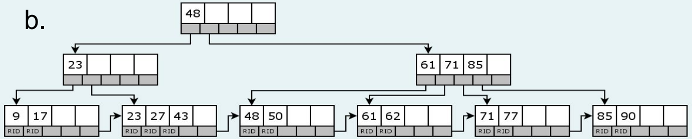
- Each node (except the root) contains $2 \leq k \leq 4$ keys.
- Inner nodes: The keys in the subtree below the pointer p_i are less than the key k_i; the keys in the subtree below the pointer p_{i+1} are greater or equal than the key k_i.
- Delete operations will first try to steal nodes from a direct sibling (first right sibling, then left sibling). If this is not possible, they will merge with a sibling (first right sibling, then left sibling).
- Pointers $p_1, \ldots, p_4$ in leaves point to row IDs. Pointer p_5 in leaves points to the next leaf.



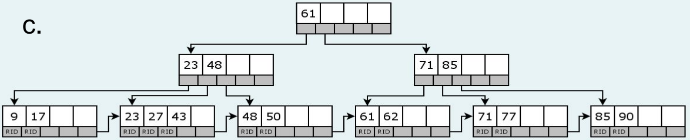
a.



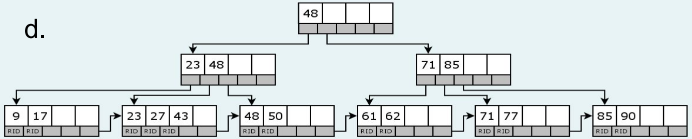
b.



c.



d.

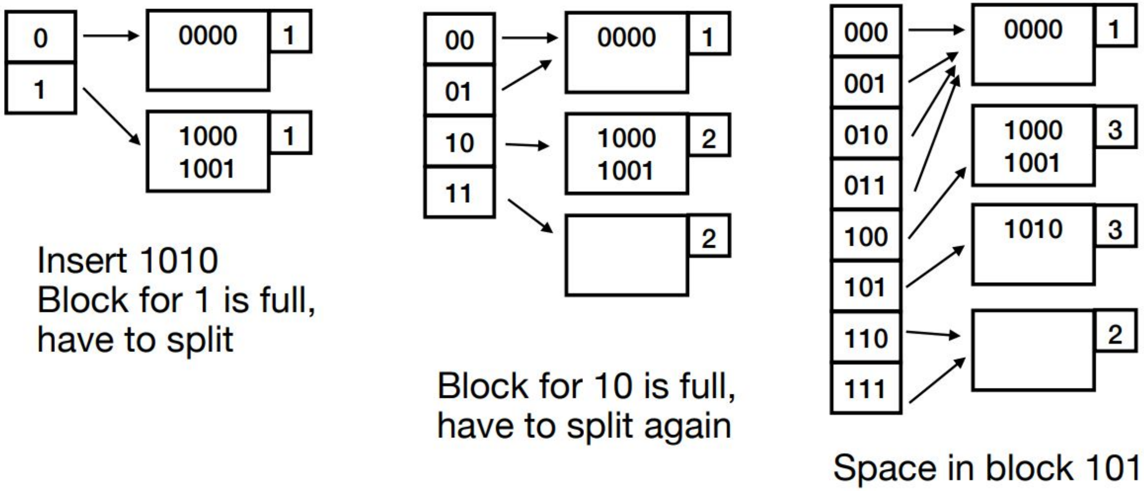


Q6: Select the type of queries which are efficiently supported by a hash index on an attribute ***a***.

True	**False**	
	✗	Range queries over the attribute ***a***.
	✗	Point queries for a single value of another attribute.
	✗	Range queries over another attribute.
✓		Point queries for a single value of the attribute ***a***.

Q7: When a block of an extensible hash table is split and the keys are distributed into new blocks, one of the new blocks can be empty.

True



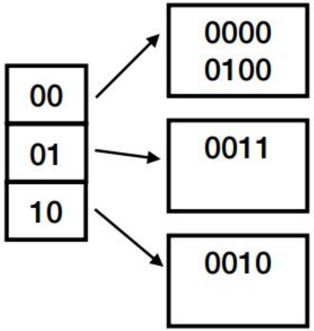
Insert 1010
Block for 1 is full,
have to split

Block for 10 is full,
have to split again

Space in block 101

Q8: Adding a key to an overflow block of a linear hash table can trigger changes in the contents of another block.

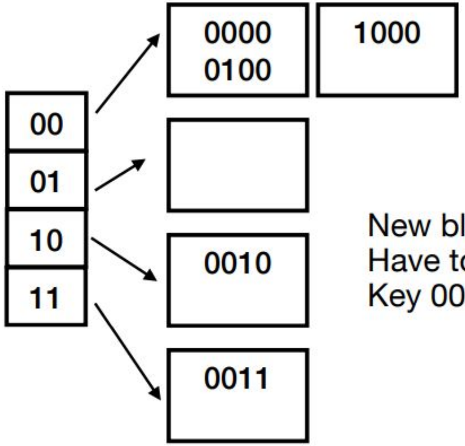
True



Num. of records per bucket is 2

i = 2, n = 3, r = 4
75% max fill grade, r/n <= 1.5

Insert 1000 into overflow of 00
Now r/n > 1.5, have to add block 11

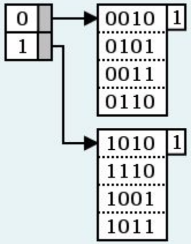


New block is 11
Have to rehash block 01
Key 0011 moves to block 11

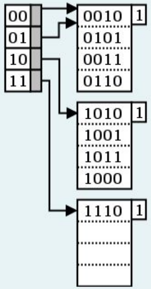
The new bucket will always be 1xxx
We always rehash bucket 0xxx

Q9: How does the hash table look like after the insertion of the key 1000?

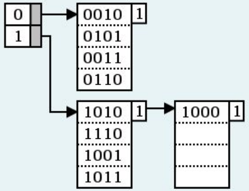
Consider the following extensible hash table that can store 4 records per block:



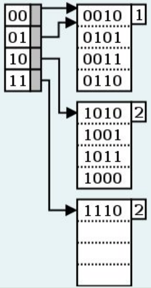
○ a.



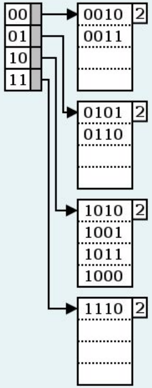
○ b.



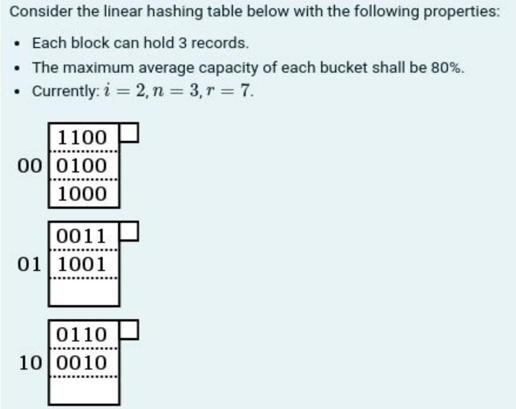
[✓ c.]



○ d.

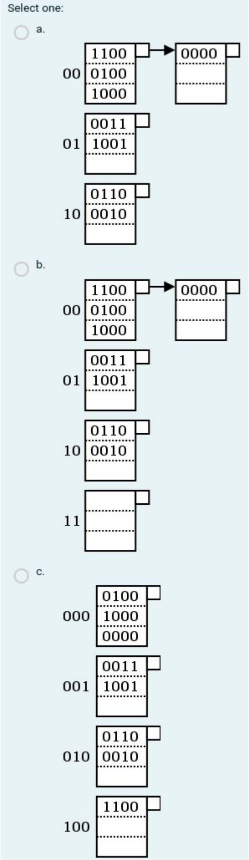


Q10: How does the hash table look like after the insertion of key 0000?



Consider the linear hashing table below with the following properties:

- Each block can hold 3 records.
- The maximum average capacity of each bucket shall be 80%.
- Currently: $i = 2, n = 3, r = 7$.

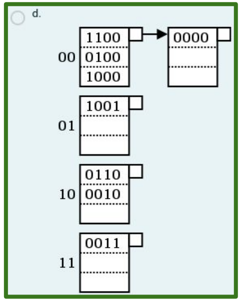


Select one:

○ a.

○ b.

○ c.



○ d.