ASSIGNMENT 1 .

1)
Consider a database with a file containing 30,000 employee records of fixed length. Each record has two fields : Name (30 bytes) and city (9 bytes). A total of 1000 distinct cities are associated with employees. The file is ordered by the non-key field 'City' and you need to create an index on the 'City' using block anchors. The disk has a block size of 512 bytes, with a block pointer of 6 bytes and a record pointer of 7 bytes.
(i)What is the index blocking factor ?

(Ans) The size of each record is 39 bytes (30 bytes for name and 9 bytes for city). Therefore, each block can store 512/39 = 13 records. The size of each index entry is 9 bytes (for the city name) plus 7 bytes (for the record pointer), which is 16 bytes. Thus, each block can hold 512/16 = 32 index entries. Therefore, the index blocking factor is 32.

(ii) How many index entries are there ?

(Ans) There are 1000 distinct cities, so there will be 1000 index entries.

(iii)How many levels are needed to create a multi-level index, such that the highest level can be fit into one block ?

(Ans)the first level will have , record pointer and key values so size of 1 index is 9 + 7 = 16 bytes , so total entries in one block is 512/16 =32 , now total blocks in 1st layer are 1000/32 = 31 blocks,

At the second level we will have key and block pointer so 9 + 6 =15 bytes of index, so we can have 512/15 = 34, entries in one block , but we only needed for 31 blocks. So only 2 levels .

(iv)What is the total number of blocks required by the multi-level index ?

(Ans) total number of blocks will be blocks in first layer + blocks in second layer which is 1 + 31 = 32.

ASSIGNMENT 2

(2)(i) Suppose we are building a B+ Tree on the EMP_ID attribute of the Employee relation. Assume that all employee names are of length 10 bytes. Disk block size is 512 bytes. Record pointer is 7 bytes. Block pointer is 6 bytes.

SINCE SIZE OF EMP_ID IS REQUIRED , AND IT IS NOT GIVEN I AM ASSUMING IT TO BE EQUAL TO 10 BYTES.

(a) What is the order of internal nodes of the B+ Tree ?
To calculate the order of internal nodes , we have in a internal node there are n block pointer ,n-1 key values
So 6*n + 10*(n-1) = 512 bytes.
n= 33.
Order is 33.

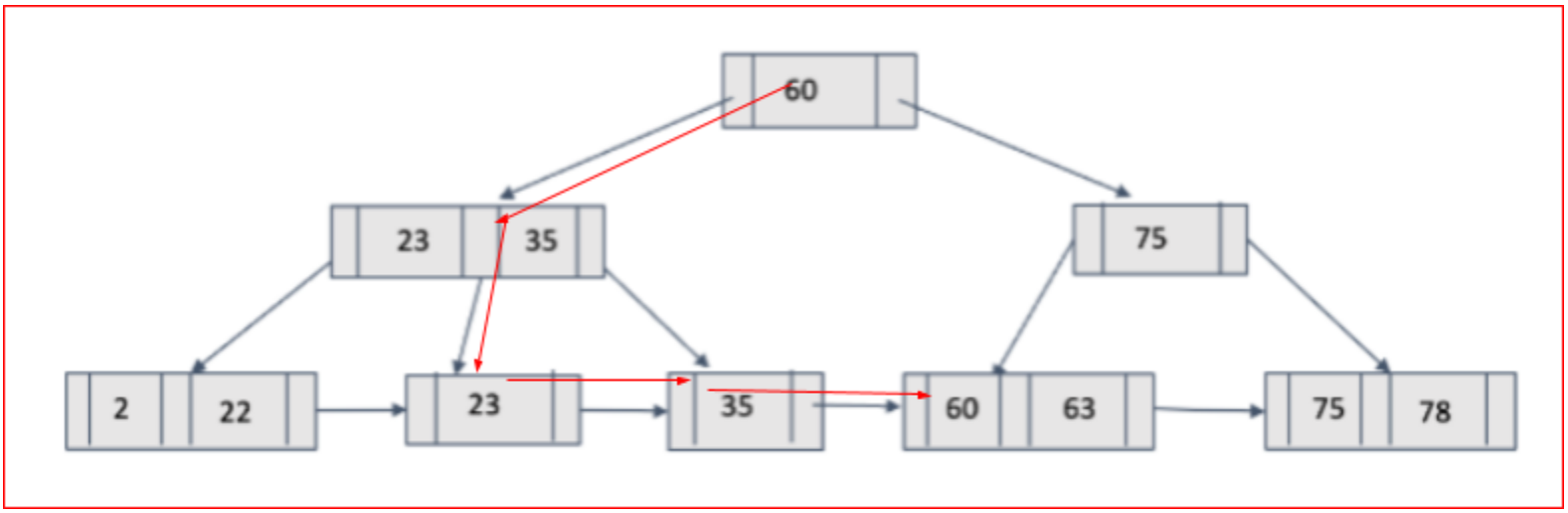
(b) What is the required number of elements in the leaf node of this Tree ?

A leaf node will have record pointers(7 bytes) and emp_id(10 bytes) and block pointer(6 bytes) to next leaf .

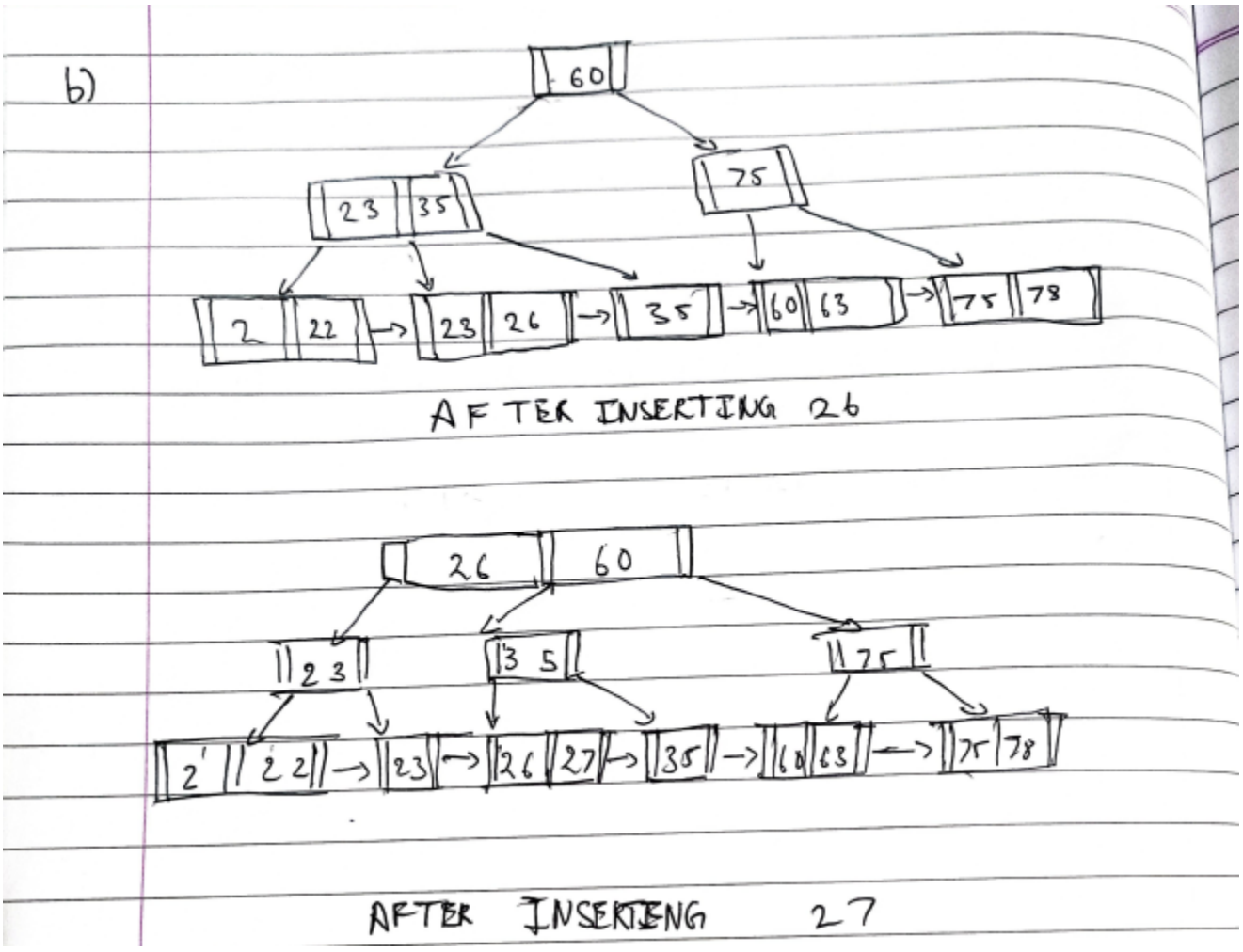
So 512 - 6 = n*(7 + 10)
n = 30.

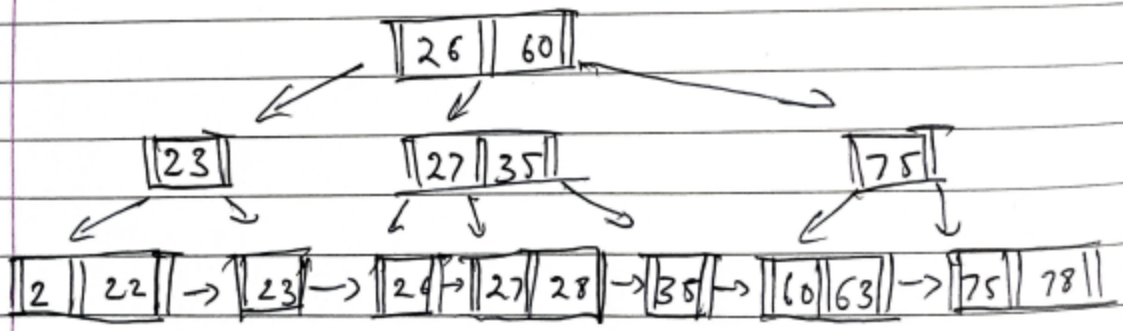
(ii) Consider the following B+ Tree.
(a) What is the minimum number of nodes needed to be accessed (including the Root node) to retrieve all the records with a search key greater than or equal to 25 and less than 37".

Ans -> 5

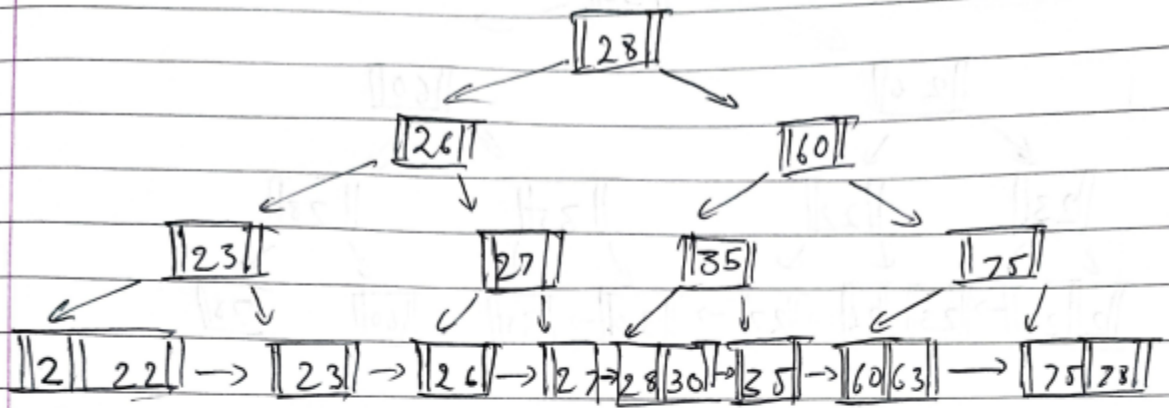


(b) What is the height of the tree after the sequence of insertions 26, 27, 28, 30. Show intermediate results.
(c) On the final tree Delete 28, 63, 75. Show intermediate results.



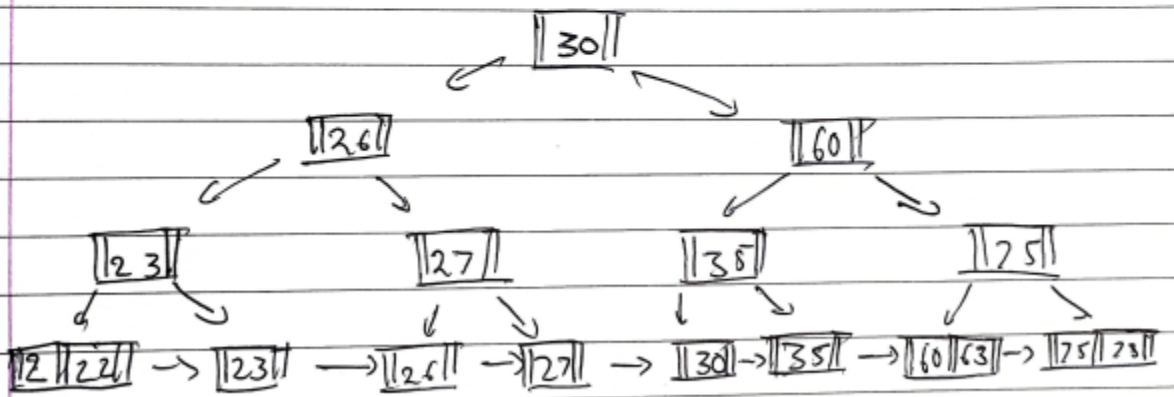


AFTER INSERTING 28

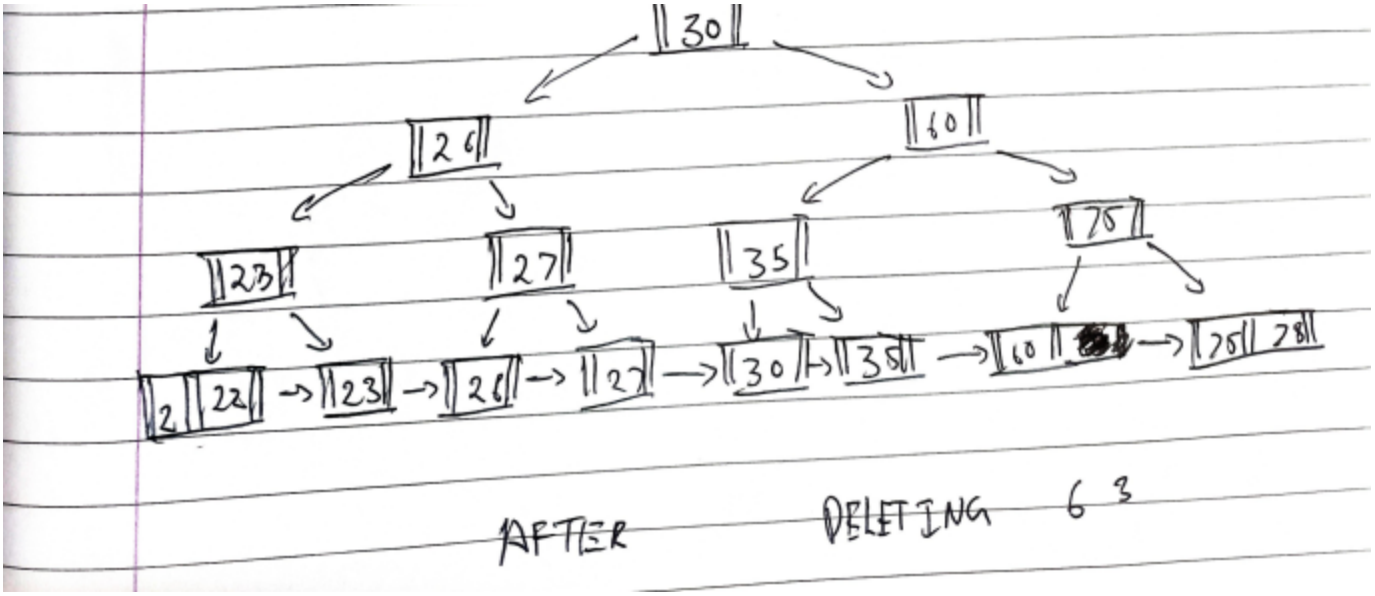


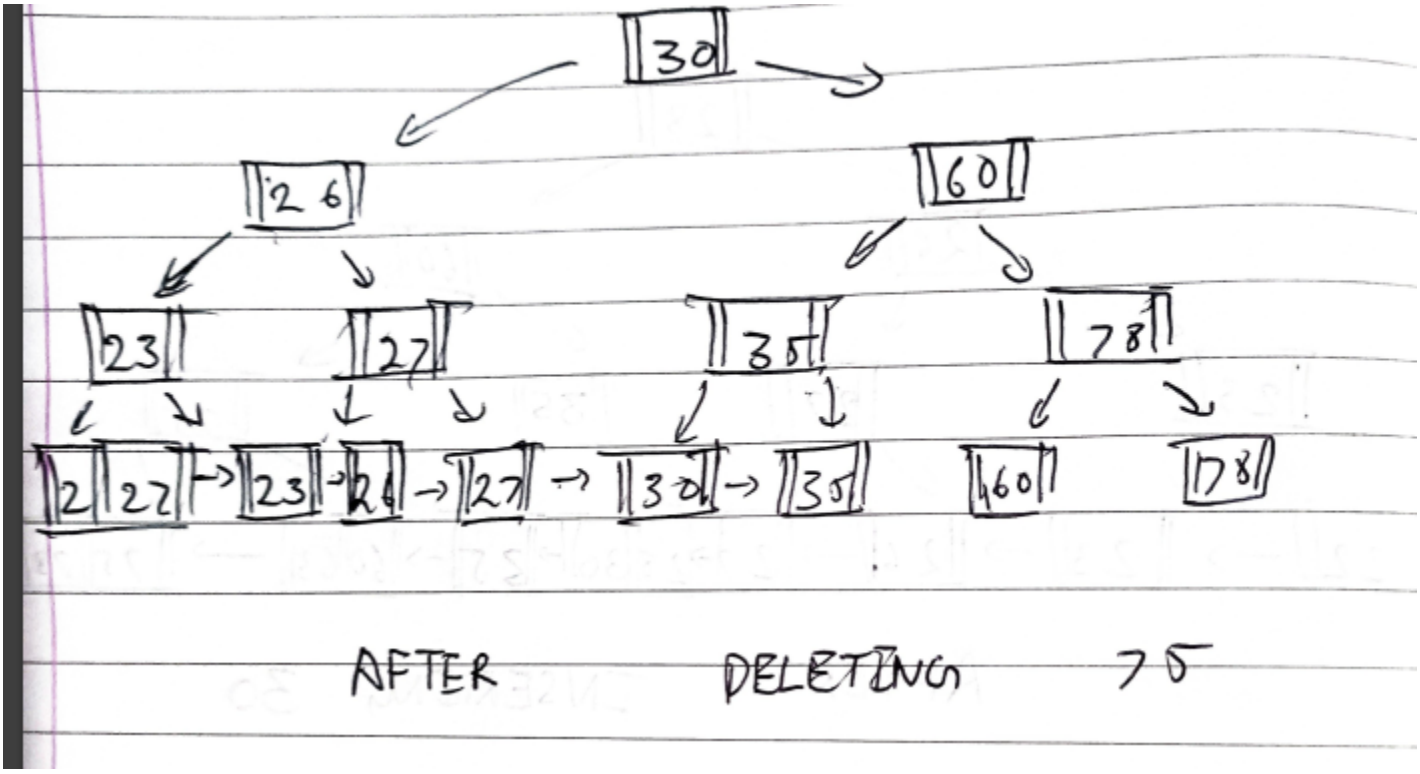
AFTER INSERTING 30

c)



AFTER DELETING 28





Assignment 3: Query Optimization

(3) Consider the following Relations –
employee (employee_id , name , department_id)
department (department_id , department_name)
product (product_id, name, price, category)
sales(employee_id, product_id, quantity)

(i)Which type of join (hash or merge) would be more efficient for optimizing the following query and why ? Explain the procedure for the chosen join operation with illustration if any.

```
SELECT employee.employee name, department.department name
FROM employee JOIN department
ON employee.department id = department.department id ;
```

For the given query, a hash join would be more efficient as it does not require the input relations to be sorted based on the joining attribute. In a hash join, the input relations are partitioned into buckets based on their joining attribute values and then a hash table is built for one of the relations. The steps for the hash join are as follows:

1. Partition the employee and department relations based on their department_id attribute into respective buckets.
2. Build a hash table for the smaller department relation using its department_id attribute as the hash key.
3. For each tuple in the larger employee relation, probe the hash table using its department_id attribute to find the matching tuples from the department relation.
4. Output the name attributes of the employee and department tuples for the matching tuples.

(ii) In the following query, which type of join (hash or merge) would be more efficient for optimizing the above query and why ? Explain the procedure for the chosen join operation with illustration if any.

```
SELECT employee.employee name, department.department name
FROM employee JOIN department
ON employee.department id = department.department id
ORDER BY dept id ;
```

For the given query, a merge join would be more efficient as it requires the output to be sorted based on the department_id attribute. The steps for the merge join are as follows:

1. Sort the employee and department relations based on their department_id attribute.
2. Initialize two pointers to the first tuples of each sorted relation.
3. Compare the department_id attribute values of the tuples pointed by the two pointers.
4. If the department_id values match, output the name attributes of the employee and department tuples and advance both pointers.
5. If the department_id values do not match, advance the pointer pointing to the tuple with the smaller department_id value.
6.

(iii)Which index is better to improve the performance of the following query and why ?

```
SELECT name
FROM product
WHERE price <= 10000 AND price >= 5000
```

An index on the price attribute would be better to improve the performance of the given query as it allows for efficient range-based retrieval of tuples based on the price condition. The index allows the database system to retrieve only the tuples that satisfy the price condition, rather than scanning the entire product relation.

(iv)What is the equivalent relational algebra for the following query and what would be the optimized query plan ?

```
SELECT sales.quantity, product.name
FROM product NATURAL JOIN sales NATURAL JOIN employee
WHERE employee.name = Jerry and product.category = music and sales.quantity > 10
```

The equivalent relational algebra for the given query is:

π quantity, name (σ employee.name = "Jerry" ∧ product.category = "music" ∧ sales.quantity > 10 (product ⋈ sales ⋈ employee))

The equivalent relational algebra for the given query is:

π quantity, name (σ employee.name = "Jerry" ∧ product.category = "music" ∧ sales.quantity > 10 (product ⋈ sales ⋈ employee))
For optimised query plan ,Select first then join .

Assignment 4: Query Cost Estimation
(4) Consider a relation S (A, B) with the following characteristics :
• a total of 7,000 tuples appear in S and each block/page can hold maximum 70 tuples
• a hash index is created on attribute A
• The values of attribute A are integers and are uniformly distributed within [1, 200].
(i)Assuming that the aforesaid index on A is a non-cluster index, estimate the number of disk accesses needed to compute the query σ A=18 (S).
(ii)What would be the cost estimate if the index were clustered ? Explain your reasoning.