Homework 2

Spring 2020

Due: 11:59PM EST, March 23, 2020. Submit using Blackboard.

(There will be a 10% penalty for each late day. After 5 late days, the homework will not be accepted.)

Part A: Disk and Files (3+3+4 = 10 pts)

1. Why variable density disk is desirable over equal density disk? Explain.

   Because in equal density disk, the further outside we go, the more distance we travel. This way the retrieval time of a block is different depending on how further the disk head is. It creates a huge difficulty for the algorithm.

2. What is sequential flooding of the buffer pool?

   Sequential flooding is a downside of using LRU eviction policy when requested pages are more than pages given in a memory buffer e.g. repeated scan of big files. For example, if we perform repeated sequential scans of N+1 pages of data and there is only N pages in the buffer. Firstly, Pages 1 to N are added into the buffer. However, now the page is full and page N+1 will replace page 1 because page 1 is LRU. As followed, requesting page 1 evicts page 2 which will be requested next, requesting page 2 evicts page 3 which will be requested next, requesting page 3 evicts page 4 which will be requested next, requesting page 4 evicts page 5 which will be requested next, so on and so forth.

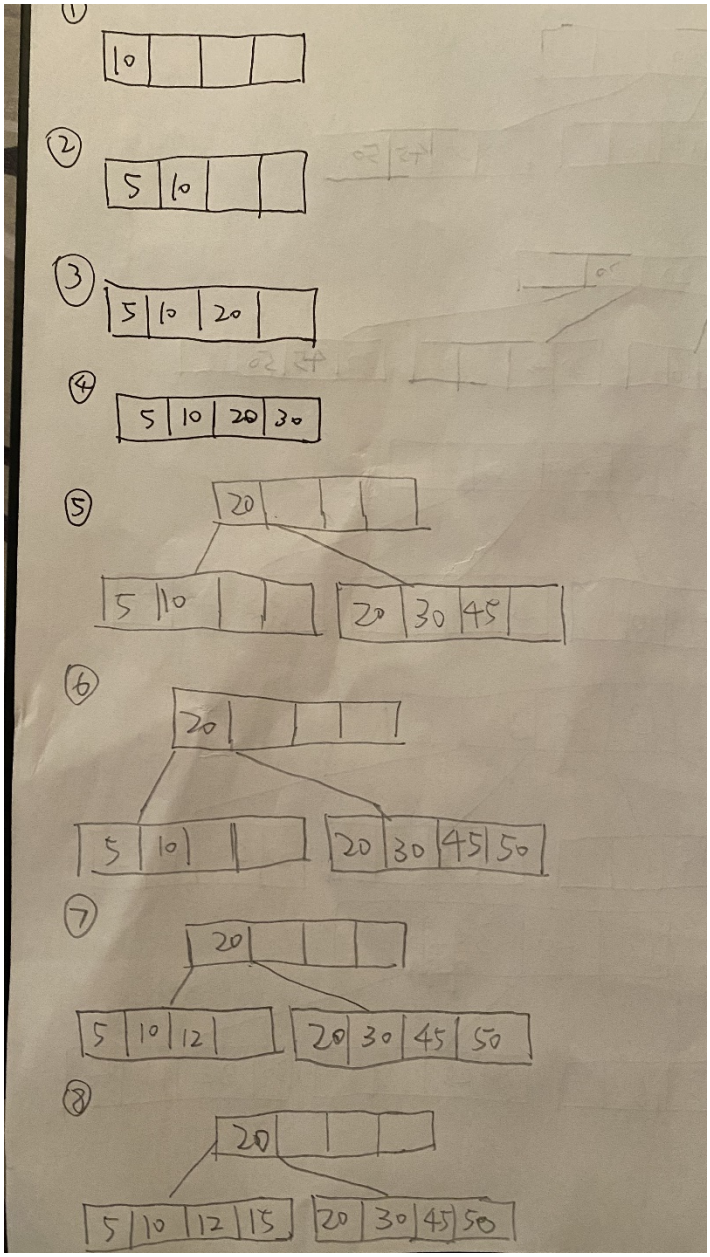
3. Which page replacement policy (MRU or LRU) is better for the Selection operator? For the nested loops join operator? Why?

   Least recent used is better for Selection Operation: For example, if we perform select x>10, we can predict that we will fetch a block of pages and look at one after another. After looking at one page, since we know access pattern is to look at the next page (we won't use what has been looked at before), we can replace LRU.

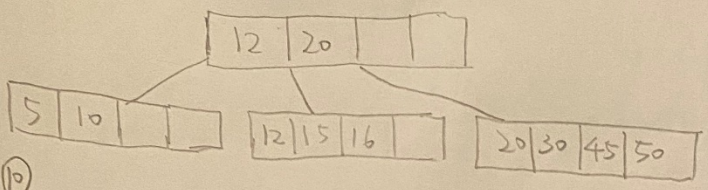
   Most recent used is better for Nested loops join Operation: For example, if table A is joining table B, after tuple x in table A joins with tuples in table B, tuple x+1 will join with tuples in table B again, starting with first tuple in table B (which is least recent used), then second, then third. Thus, we want to keep the least recent used in cache.

Part B: Tree-Based Indexing (50 pts)

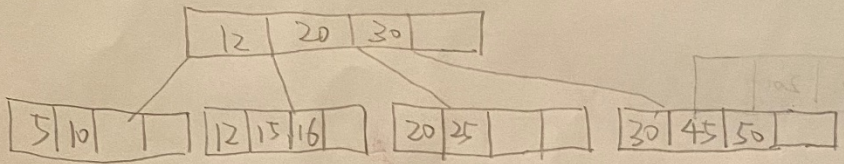
1. Show the result of inserting 10, 5, 20, 30, 45, 50, 12, 15, 16, 25, 32, 2 , 1, 23, 49 , 50 and 3 (in this order) into an initially empty B+-tree of order d = 2. Show every step (in drawing) after inserting each of the values. ((first 4x1pt) + (next 13x2pts) = 30 pts)


1
10
2
5 10
3
5 10 20
4
5 10 20 30
5
20
5 10
20 30 45
6
20
5 10
20 30 45 50
7
20
5 10 12
20 30 45 50
8
20
5 10 12 15
20 30 45 50

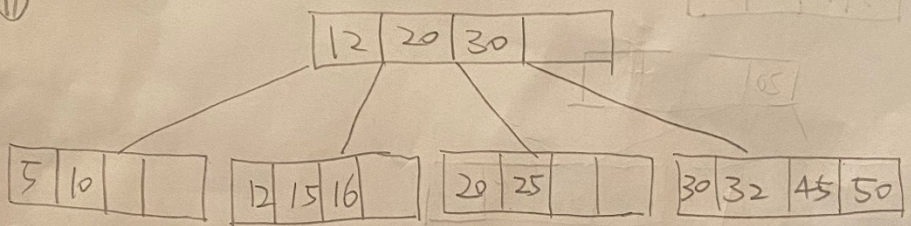
⑨


12 20
5 10
12 15 16
20 30 45 50

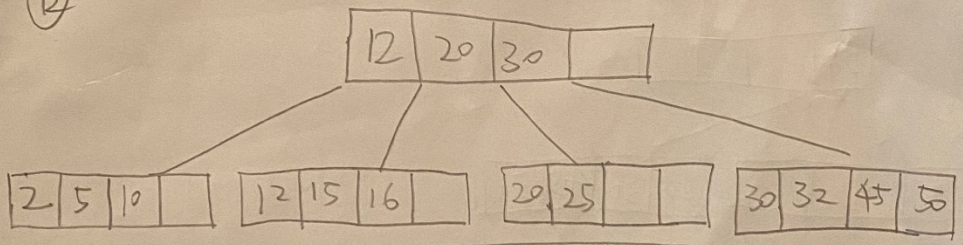
⑩


12 20 30
5 10
12 15 16
20 25
30 45 50

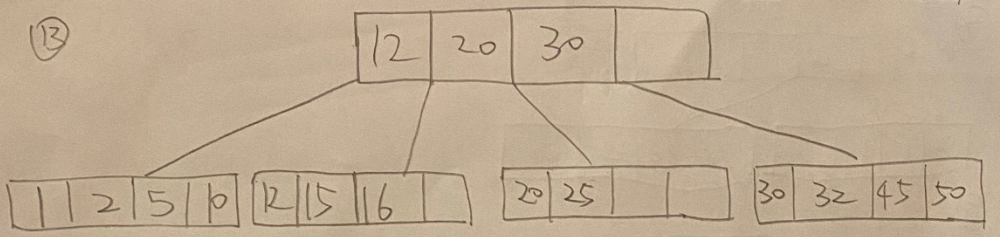
⑪


12 20 30
5 10
12 15 16
20 25
30 32 45 50

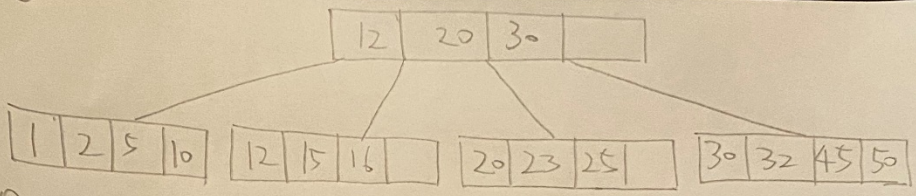
⑫


12 20 30
2 5 10
12 15 16
20 25
30 32 45 50

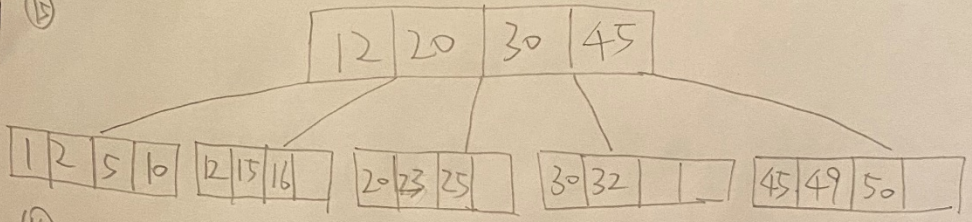
⑬


12 20 30
1 2 5 10
12 15 16
20 25
30 32 45 50

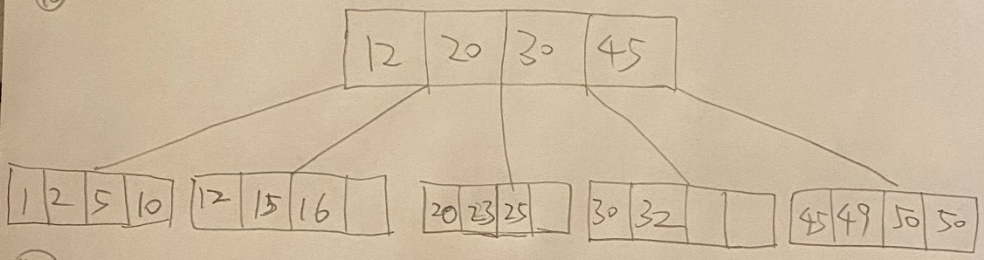
(14)


12 20 30
1 2 5 10
12 15 16
20 23 25
30 32 45 50

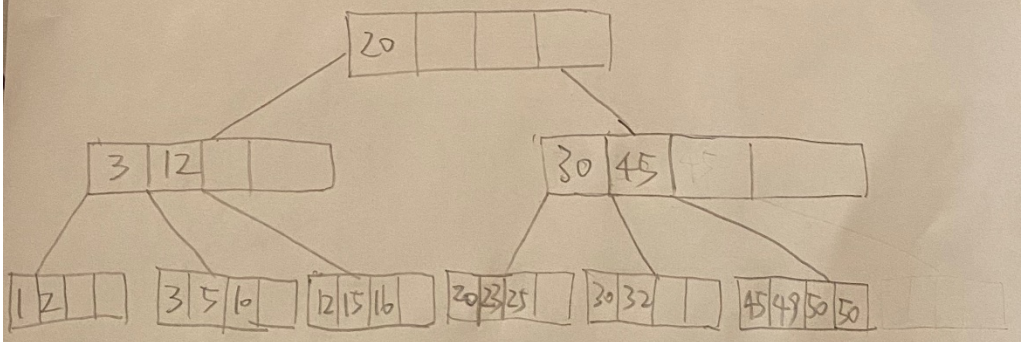
(15)


12 20 30 45
1 2 5 10
12 15 16
20 23 25
30 32
45 49 50

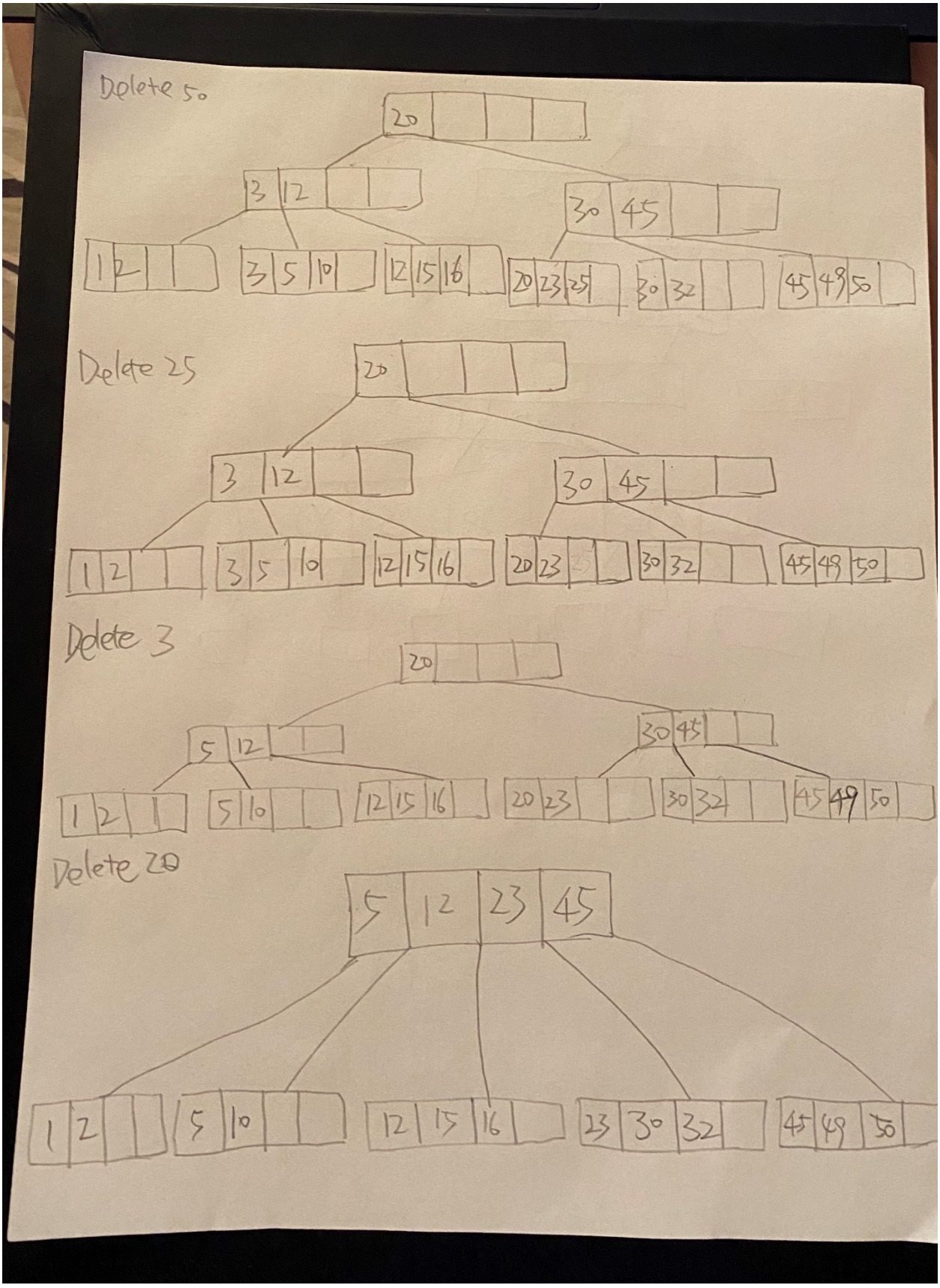
(16)


12 20 30 45
1 2 5 10
12 15 16
20 23 25
30 32
45 49 50 50

(17)


20
3 12
30 45
1 2
3 5 10
12 15 16
20 23 25
30 32
45 49 50 50

2. Show the result of deleting 50, 25, 3, 20 (in this order)) from the previous B+-tree. Show the B+-tree after each deletion. ((first 3x2pts) + (next 1*4pts =10 pts)



3. Assume that you have a database system where the workload is very dynamic (lots of inserts, updates and deletes). Now, you are asked to choose between the following index structures:

ISAM and B+Tree. What will be your preferred index structure and why? Explain in detail by justifying your choice. Now, repeat Question B.3. if the database is entirely static. (10 pts)

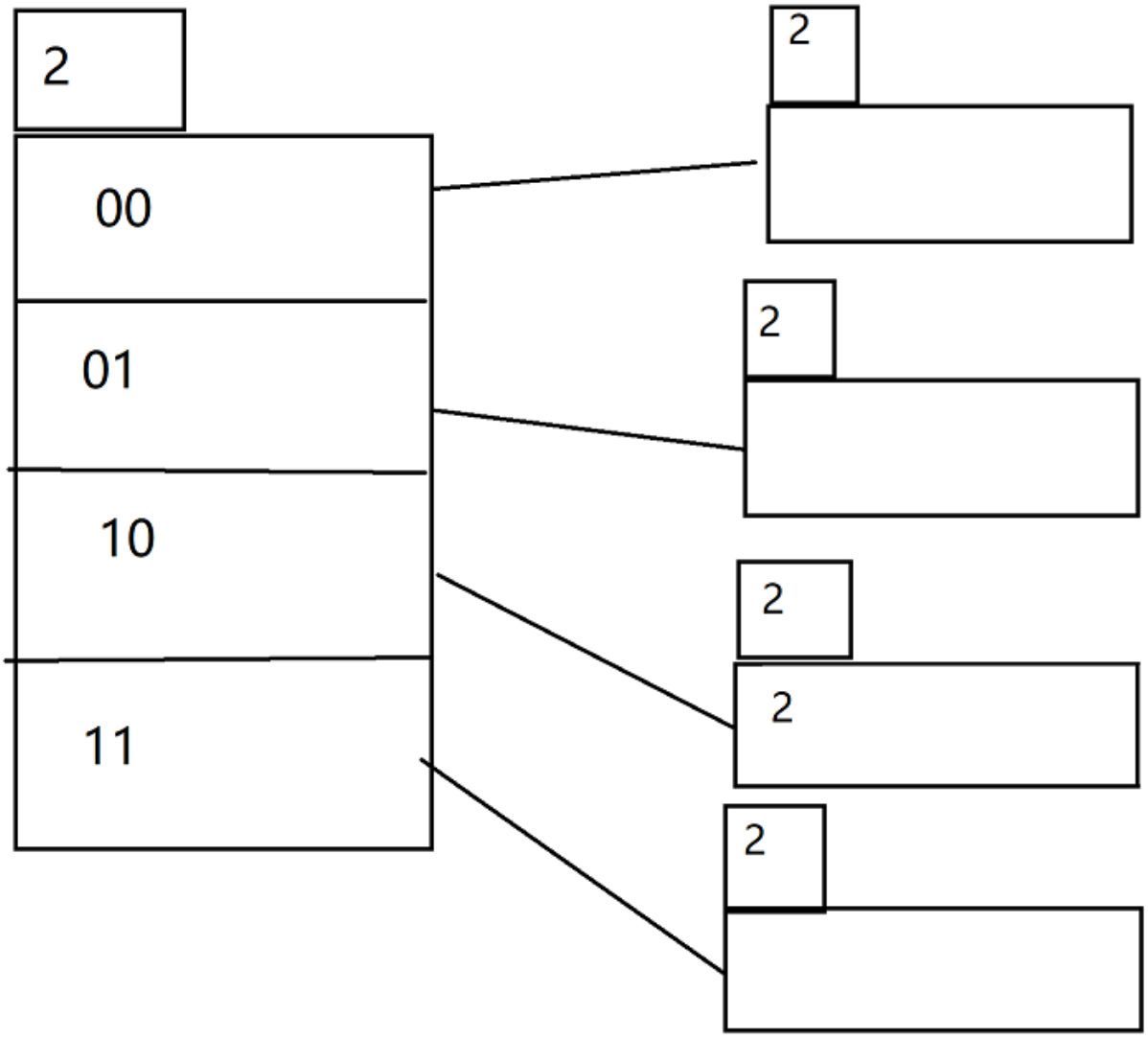
If my database system has a very dynamic workload, I would choose B+ tree as index structure. Because my database has lots of inserts, updates and deletes happening. With ISAM, my data structure would be overflowed (have overflow chains) quickly, and it would take more time to get to a data page which was in one of the overflow chains. B+ tree also has the advantage that all data pages are in the bottom leaf level. So, scanning data pages is much easier compared to ISAM who has data pages distributed in different level.

If my database is entirely static, I would choose ISAM as index structure. Because since my database is entirely static (no inserts, updates and deletes), I wouldn't have overflow chains which is a huge performance degradation problem for ISAM approach. Compared to B+ tree which has an additional leaf level, ISAM requires fewer disk operations to visit a data page. ISAM also takes less space than B+ tree, because B+ tree's approach only requires 50% to 100% occupancy. Also take concurrency as consideration, ISAM doesn't require locking in the index pages, while B+ tree needs index locking, which harms concurrent performance.

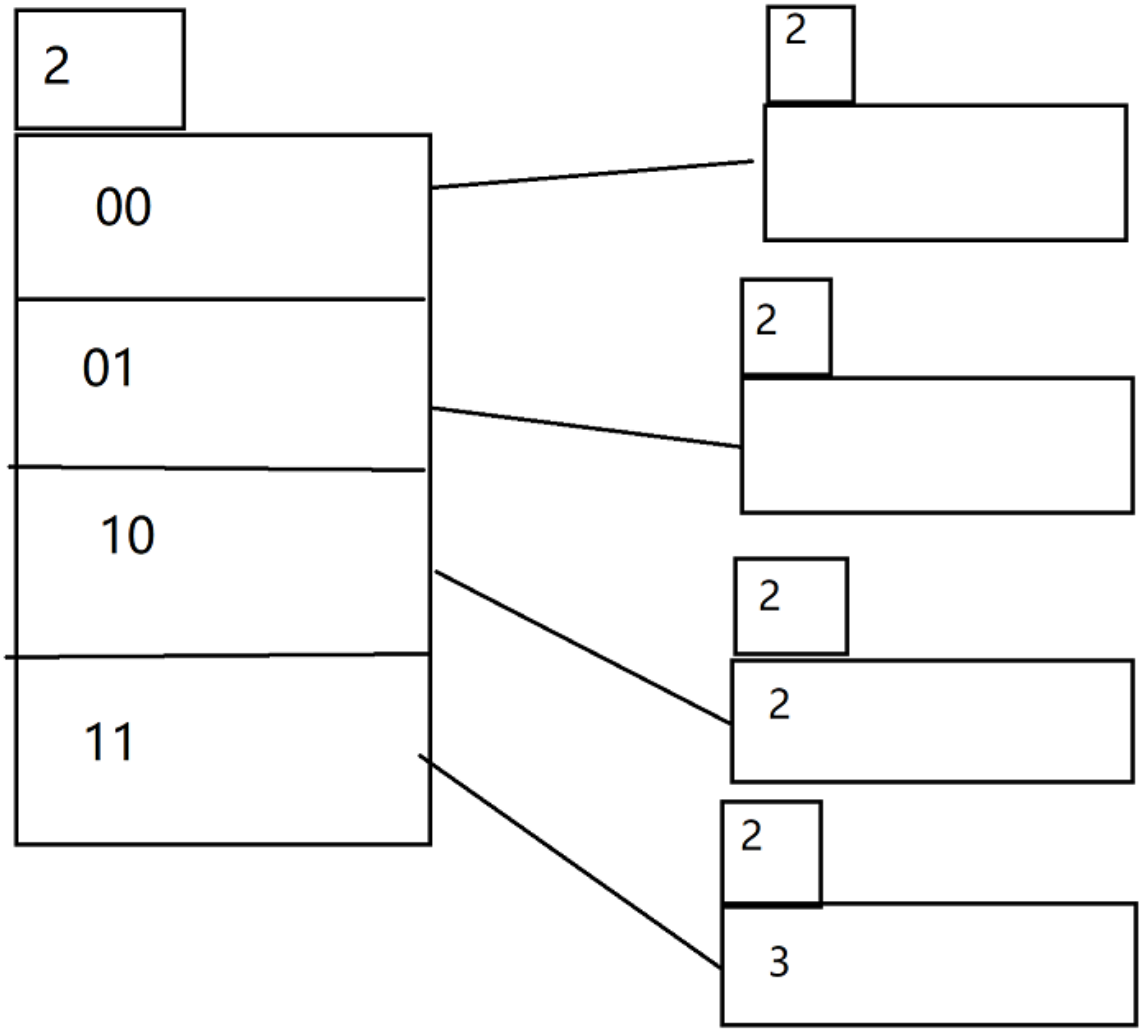
Part C: Hash-based Indexing (40 pts)

1. Consider an extendible hashing structure where:
    a. Initially each bucket can hold up to 3 records.
    b. The directory is an array of Size 4. Now, construct the extendible hashing structure by inserting the keys 2, 3, 7, 14, 15, 18, 20, 26, 27 (in this order). Start by using the lowest 2 bits for the hash function. Show each step. (10 pts)

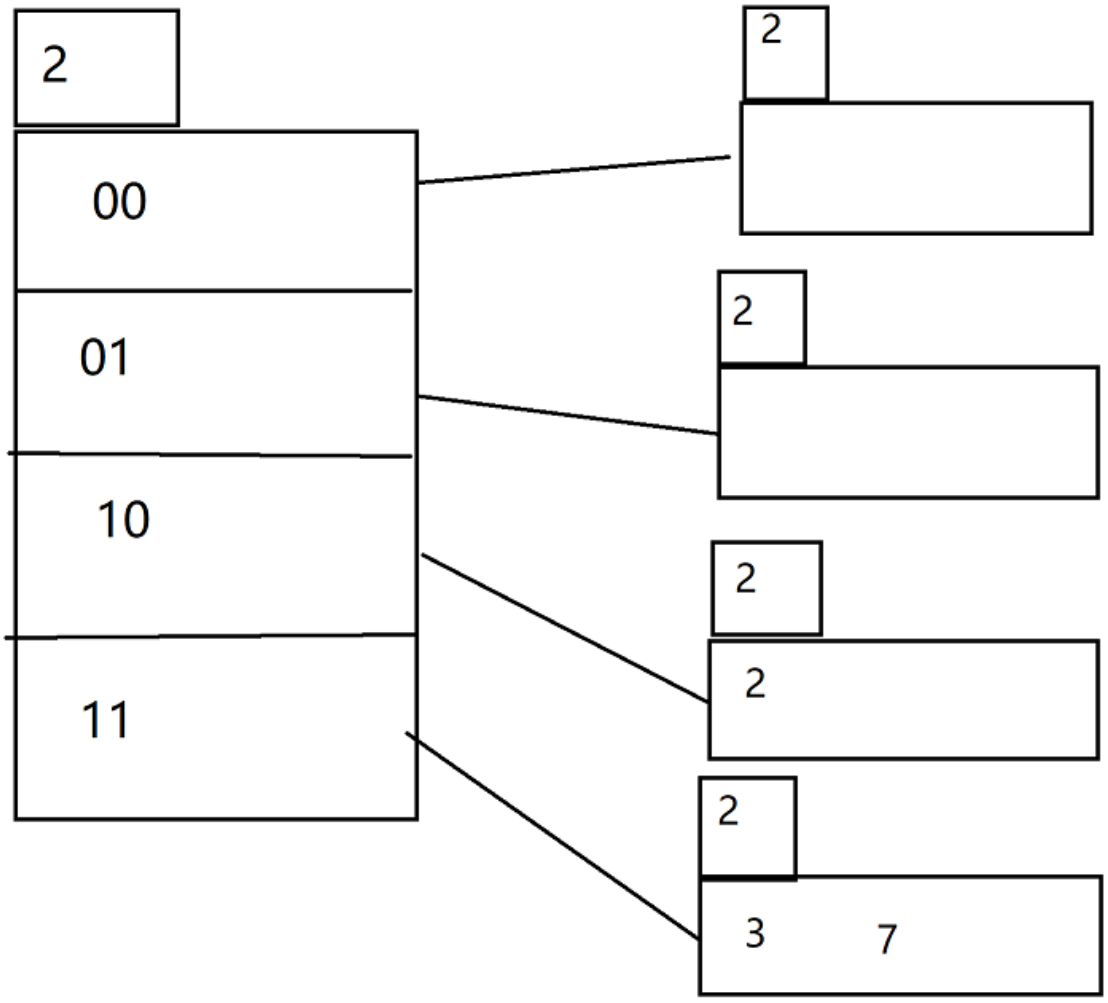
Insert 2:



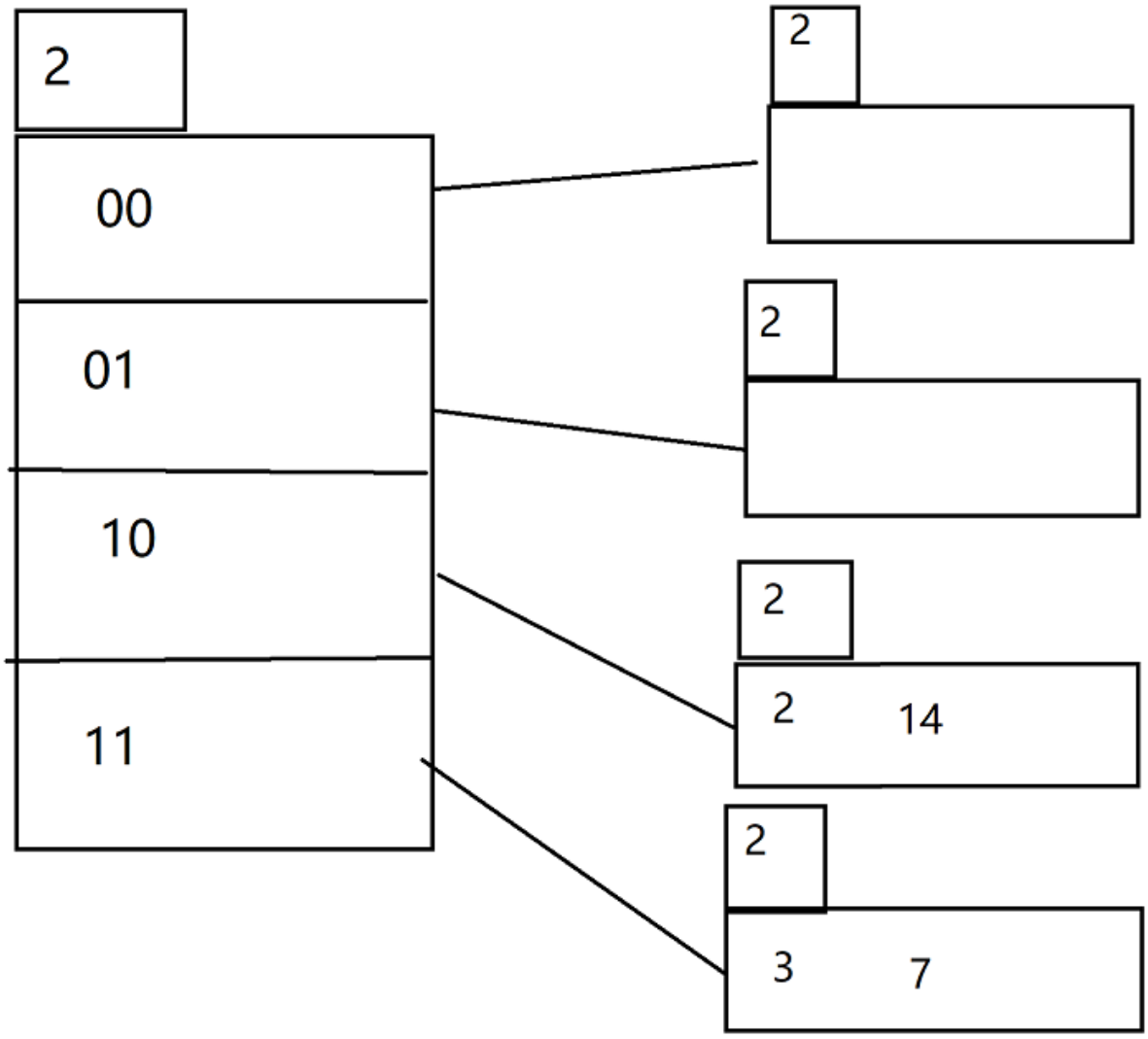
insert 3



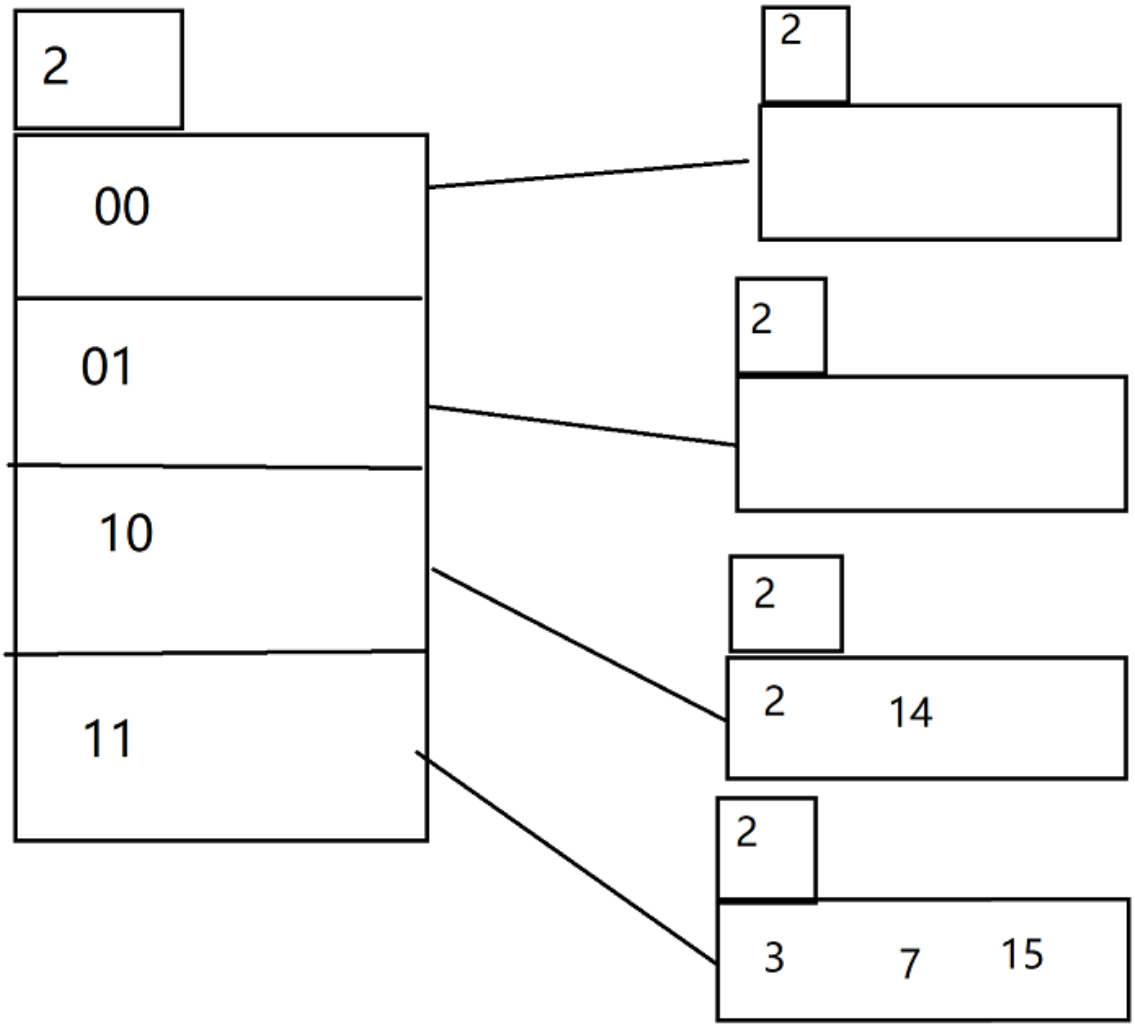
insert 7:



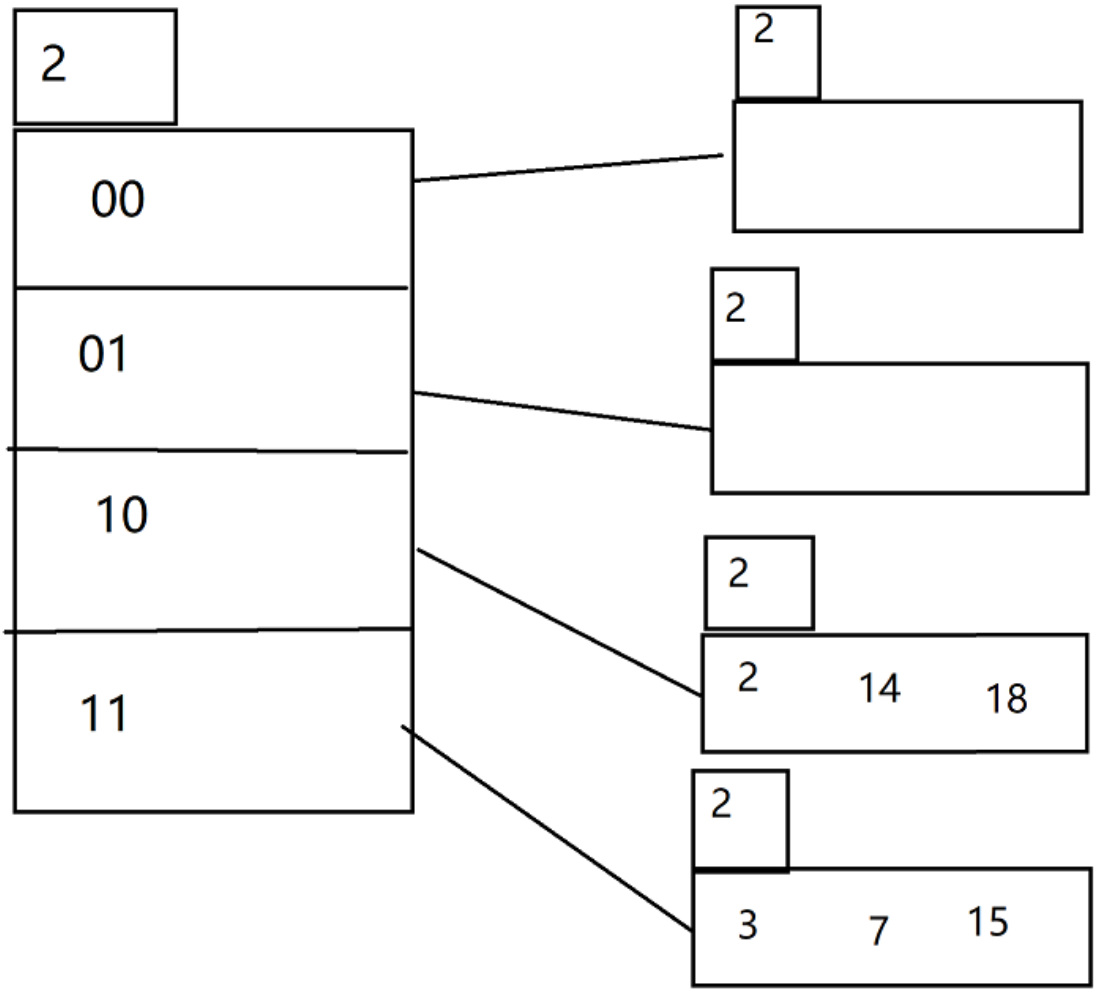
insert 14:



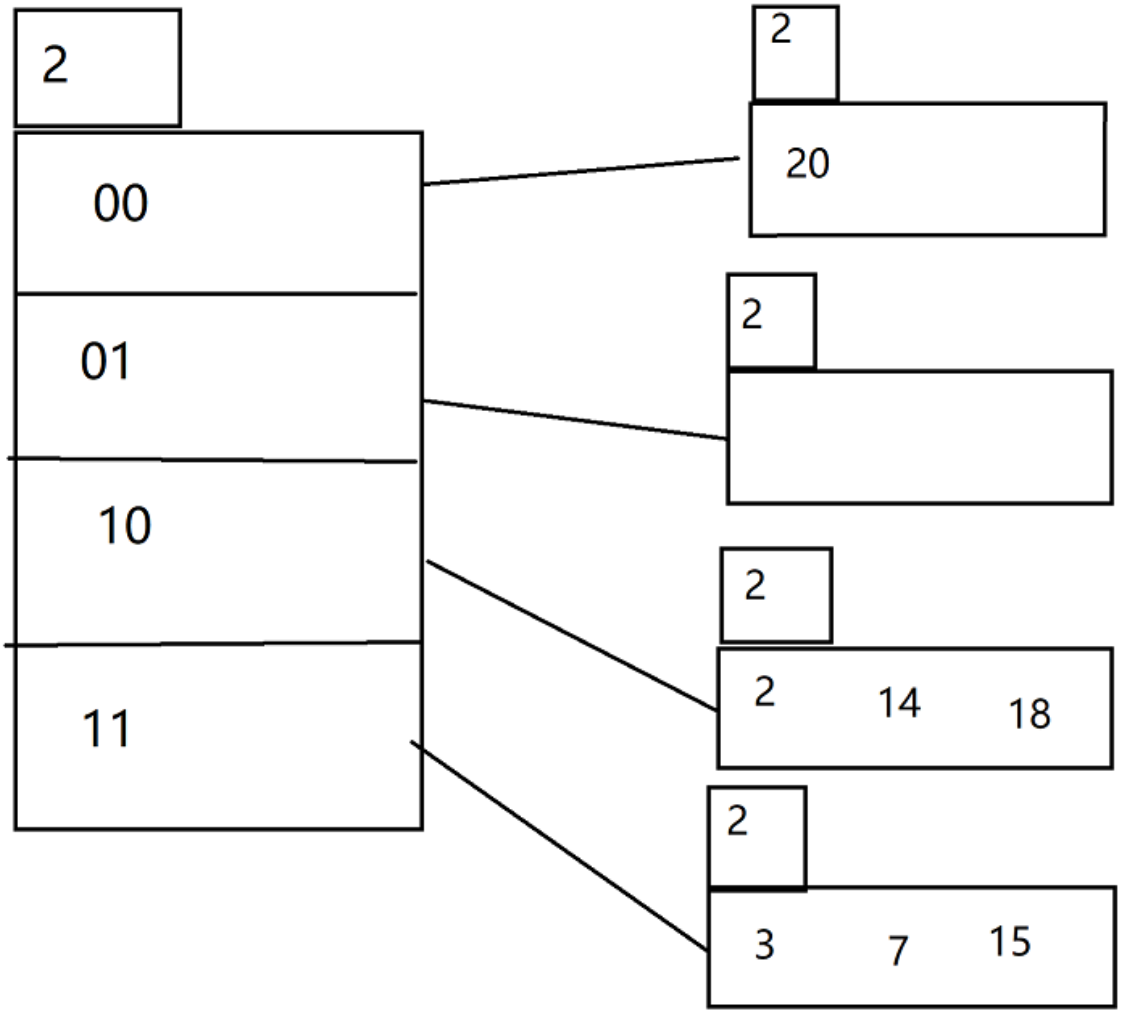
insert 15:


2
00
01
10
11
2
2
2
2
14
2
3
7
15

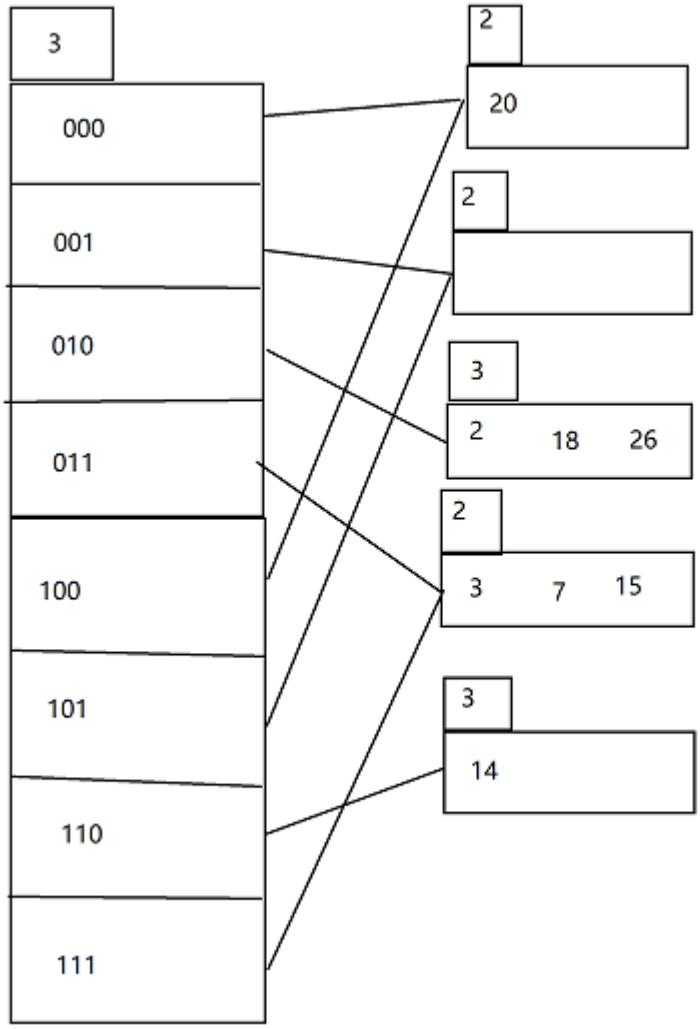
insert 18:



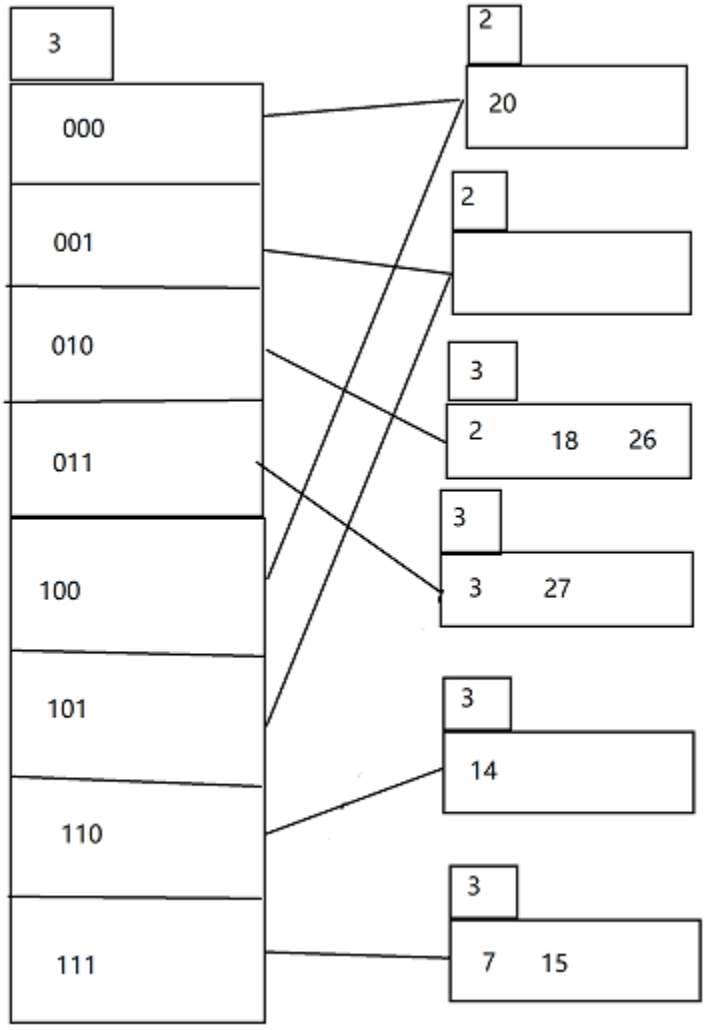
insert 20:



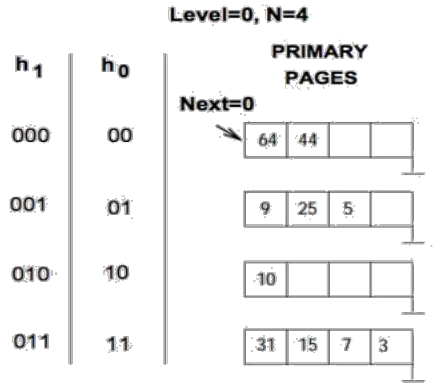
insert 26:



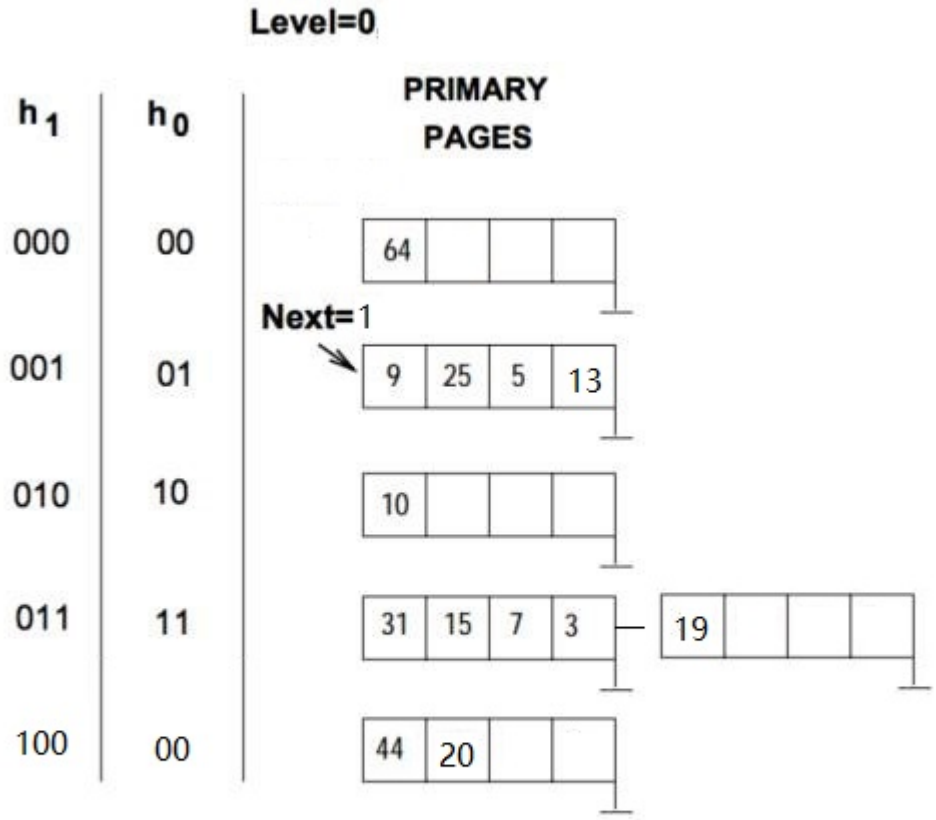
Insert 27:



2. Consider the following Linear Hash Table, and assume that a bucket split occurs whenever an overflow page is created. h0(x) takes the rightmost 2 bits of key x as the hash value, and h1(x) takes the rightmost 3 bits of key x as the hash value.
   Now insert 13, 19 and 20 and draw the final hash table. Write the new hash function (if any), and also update the "Next" pointer. (10 pts)



Final Result:



3. Let us consider a relation R (s, t, u) containing 1 billion (10^9) records. Each page of the relation holds 10 records. R is organized as a heap file with unclustered indexes and the records in R are

randomly ordered. Assume that atribute ***s*** is a candidate key for R, with value lying in range 0 to 999,999,999.
Now, for each of the following queries, name the approach that would most likely require the fewest I/Os for processing the query. Justify your answer by approximately calculating the cost (in terms of disk accesses) for each approach to answer each query. (20 pts)

The approaches to consider are as follows:

i. Scanning through the whole heap file for R.
ii. Using a B+ tree index on attribute R.s
iii. Using a hash index on attribute R.s

The queries are:

i. Find all R tuples.
ii. Find all R tuples where R.s < 100
iii. Find all R tuples where R.s = 100
iv. Find all R tuples where R.s >100 and R.s<150

Also assume that the height of the B+Tree is: **h=3**, the number of data entries per page is **M and M>10**. Additionally, (assume that) after accessing the data entry, it takes one more disk access to get the actual record and let the occupancy factor in hash index is: ***c***.

i. Find all R tuples.
Fewest I/Os: Scanning through the whole heap file for R.
The cost is: number of pages = 1 billion/10 = 100,000,000

Other approaches:
Using a B+ tree index on attribute R.s
The cost is: 100,000,000*M + I/Os spent on reading index
Using a hash index on attribute R.s
The cost is: 100,000,000*M + I/Os spent on reading index

ii. Find all R tuples where R.s < 100
Fewest I/Os: Using a B+ tree index on attribute R.s
The cost of is: logF(size of index) + I/Os spent on matching records

Other approaches:
Using a hash index on attribute R.s
The cost is: number of pages = 1 billion/10 = 100,000,000
Scanning through the whole heap file for R.
The cost is number of pages = 1 billion/10 = 100,000,000

iii. Find all R tuples where R.s = 100
Fewest I/Os: Using a hash index on attribute R.s
The cost is: 2 (one for directory and one for bucket)

Other approaches:
Using a B+ tree index on attribute R.s
The cost is 1 + logF(0.15*100,000,000), assuming the fanout is F and on average the tree has 67% occupancy.
Scanning through the whole heap file for R.
The cost is 0.5* Number of total pages (because on average we must scan half the file to find equality) = 50,000,000

iv. Find all R tuples where R.s >100 and R.s<150
Fewest I/Os: Using a B+ tree index on attribute R.s
The cost of is: logF(size of index) + I/Os spent on matching records

Other approaches:
Using a hash index on attribute R.s
The cost is: number of pages = 1 billion/10 = 100,000,000
Scanning through the whole heap file for R.
The cost is number of pages = 1 billion/10 = 100,000,000