CS 351 DATA ORGANIZATION AND MANAGEMENT
Homework 4

1) In an extendible file environment, the number of entries is 2^k where k is the first number of digits that are used to find the correct entry in the table. In other words, k can be referred as the level of the index as well.
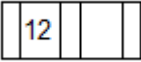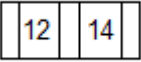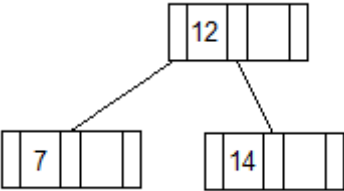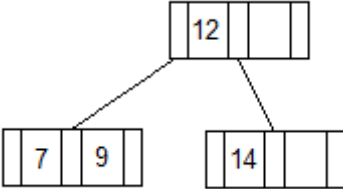
In the question,
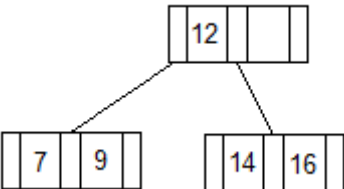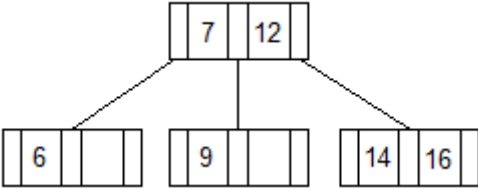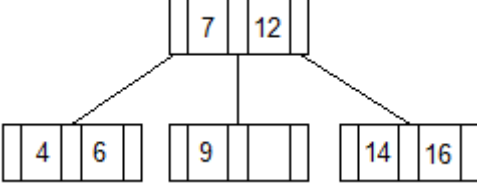
- pseudo keys are obtained by mod(key, 5693)
- R (record size) = 400 bytes
- Disk Block Address representation (pointers) requires 4 bytes.
- Block Size = 2400 bytes (the parameter/value of IBM 3380)

a) **The maximum memory requirement of the file directory** is obtained as follows: The value 5693 is between 2^{12} and 2^{13}. Since the size of extendible-hashing tables is always a power of 2, we assume the table size to be 2^{13}. Thus the index takes 2^{13} x 4 bytes of storage for pointers and the table in total takes: 2^{13} x 13 bits + 2^{13} x 4 x 8 bits = 2^{13} x 45 bits = 2^{10} x 45 bytes = **45 KB**.

b) Normally, we do not have any overflow pages in extendible hashing. However, it is possible to have overflow blocks in an extendible file environment as follows: If the entries all have the same hash value and although we double the directory size, the entry still wants to go to the bucket where there is no space. In that manner, the file directory continues to double and when it reaches its maximum size, it starts to have overflow buckets in order for the entries to be placed. The example for this case will be as follows: Assume IBM 3380 environment and each bucket includes 2400/400 = 6 records. There are 7 entries that have the same hash value; i.e. the same data-bucket address which is 0 for instance. Then, i^{th} level of the data bucket equals the kth level of the index. In this case, although the directory size is doubled, all entries need to go to the same index, and the directory size reaches the maximum value. At this point, overflow buckets are formed and the values are distributed through those overflow bucket in the index where overflow occurred.

c) **The maximum file size in terms of number of records can be obtained as follows:** In the maximum file size calculation, 5693 entries will each have one bucket (page) and each bucket will have 2400/400 = 6 records (According to IBM 3380 parameters). Therefore, in total it makes 5693 x 6 = **34,158** records.
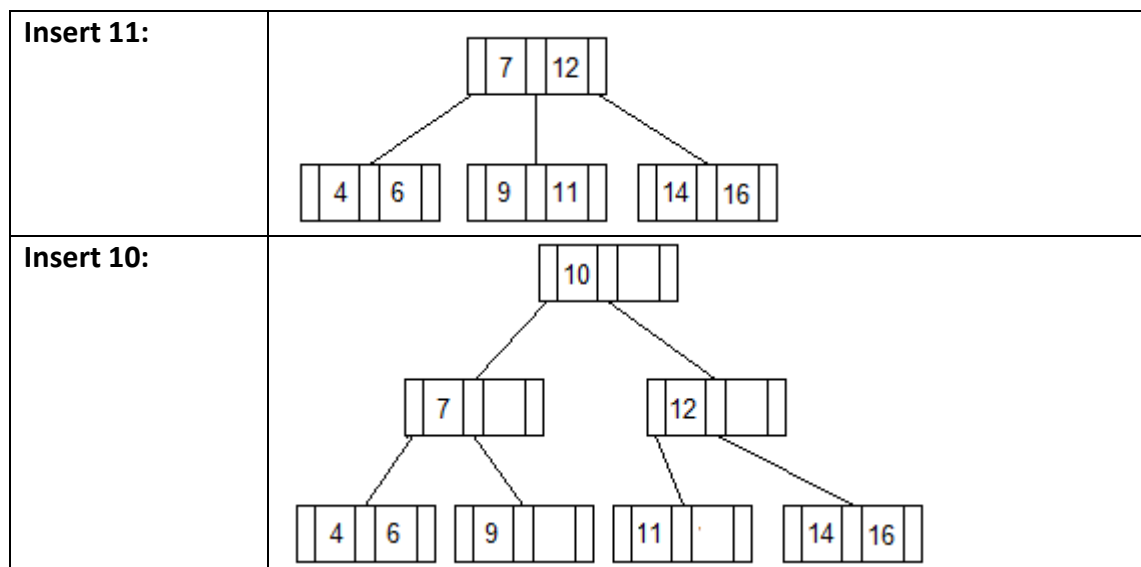
2) In an extendible hashing file with d (directory level) equal to 4, the minimum and maximum number of pages with p (page hashing level) equal to 1, 2, 3, 4 and 5.

d = 4 (directory level)	Minimum number of pages	Maximum number of pages
p = 1 (page hashing level)	In minimum case, there is no page with p=1, for example all pages may have page hashing level as 4. Then since d = 4, all pages are with the value p = 4, there will be no pages with p = 1. Therefore the minimum number of pages with p=1 is **0**.	2^{d-p} is the number of links pointing connected to a page. When p is 1, the leftmost bit (prefix 1) will be used for distinguishing the entries. There are 2^4 entries in the table and the first 8 of them links to page 1 and the last 8 of them links to other pages with p>1. Therefore the maximum number of pages with p=1 is **1**.
p = 2 (page hashing level) = 2	In minimum case, there is no page with p=2, for example all pages may have page hashing level as 4. Then since d = 4, all pages are with the value p = 4, there will be no pages with p = 2. Therefore the minimum number of pages with p=2 is **0**.	2^{d-p} is the number of links pointing connected to a page. When p is 2, the leftmost two bits (prefix 2) will be used for distinguishing the entries. There are 2^4 entries in the table and the first 12 entries will point to 3 pages with p=2. Each page with p=2 will get 4 links pointing to it. Therefore, the maximum number of pages with p=2 is **3**.
p = 3 (page hashing level)	In minimum case, there is no page with p=3, for example all pages may have page hashing level as 4. Then since d = 4, all pages are with the value p = 4, there will be no pages with p = 3. Therefore the minimum number of pages with p=3 is **0**.	2^{d-p} is the number of links pointing connected to a page. When p is 3, the leftmost three bits (prefix 3) will be used for distinguishing the entries. There are 2^4 entries in the table and the first 14 entries will point to 7 pages with p=3. Each page with p=3 will get 2 links pointing to it. Therefore, the maximum number of pages with p=3 is **7**.
p = 4 (page hashing level)	In minimum case, there are at least two pages with p=4. For example, the first 8 indexes point to page with p=1, the next 4 indexes point to page with p=2, the next two indexes point to page with p=3. Then the last two indexes each point to pages with p=1. Since the doubling occurs when there is an overflow in the page, at least two pages with p=d=4 will appear. Therefore the minimum number of pages with p=4 is **2**.	2^{d-p} is the number of links pointing connected to a page. When p is 4, all of the 4 bits (prefix 4) will be used for distinguishing the entries. There are 2^4 entries in the table and the first 16 entries will point to 16 pages with p=4. Each page with p=4 will get 1 link pointing to it. Therefore, the maximum number of pages with p=4 is **16**.
p = 5 (page hashing	It is not possible to have a page with p>d, because d indicates the	It is not possible to have a page with p>d, because d indicates the directory level and p

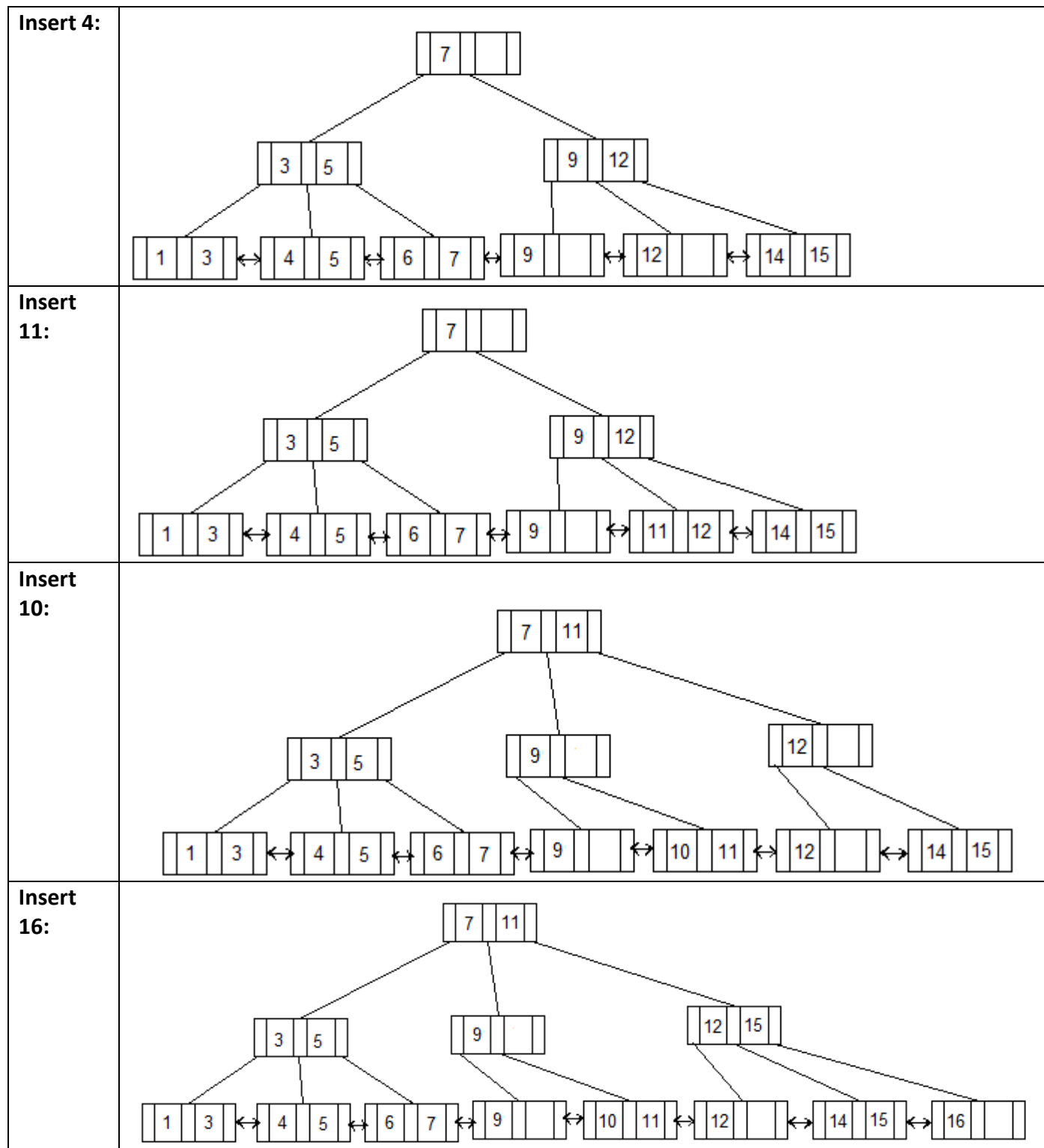| level)	directory level and p indicates the page hashing level (the number of prefixes that will be used to identify the elements in the corresponding page). We cannot identify 4 bit index with 5 prefixes. Minimum value: **0**	indicates the page hashing level (the number of prefixes that will be used to identify the elements in the corresponding page). We cannot identify 4 bit index with 5 prefixes. Maximum value: **0**

3)

Insertion	Contents of B Tree of degree (order) 1 after insertions
Insert 12:	
Insert 14:	
Insert 7:	
Insert 9:	
Insert 16:	
Insert 6:	
Insert 4:	

Insert 11:	
Insert 10:	

4)

Insertion	Contents of B+ Tree of degree (order) 1 after insertions
Insert 14:	
Insert 7:	
Insert 9:	
Insert 5:	
Insert 15:	

Insert 3:	
Insert 1:	
Insert 6:	
Insert 12:	

Insert 4:	
Insert 11:	
Insert 10:	
Insert 16:	

5) **a) Time needed to find the average account balance with B+ trees in which the data nodes are not connected to each other can be calculated as follows:**
key size = 8 bytes
pointer size= 4 bytes
fo= (2400/12)x0.7= 140 (we can assume this value as fan out using Salzberg's values)
n: no. of records
bk: no. of data nodes
bk= n/(2 x d x 0.7) , ln2= 0.7 (approx.)

No. of disk accessed in order to find the average can be approximated as 2 x bk; however this is not a good estimate since when we access an index node in the disk it will be kept in main memory by the operating system as long as we need it. Recall that each index node has fan out number of data node pointers and therefore by each index node we will be able to access (point to) fo no. of data nodes. By this observation a much more accurate estimation for the number of disk accesses is bk/fo + bk.

The calculation of the average includes exhaustive reading of the file, i.e. T_X.
In IBM 3380 environment, bucket size is 2400 bytes => **Bkfr = 2400/400 = 6**
T_X = (bk/fo) x (s+r+btt) + bk x (s+r+btt)
# of data nodes(bk) = 200,000/(ln2 x Bkfr) = 200,000/(0.7 x 6) = **47,619** (According to Yao, on the average each data node is 70% full, that is why we have divided the number of records by 0.7 as well)

Therefore, the time for finding the average account balance becomes:
T_X = 47,619 / 140 x (16 + 8.3 + 0.8) + 47,619 x (16 + 8.3 + 0.8) = **1,203,774 msec = 1,203 sec = 20 minutes**
T_F **=** s+r+btt+s+r+btt = 2s + 2r + 2btt = **50.2 msec**
T_N **= 1 / (Bkfrxln2) x [(s+r+btt) + 1/fo x (s+r+btt)] = 25.1/4.2 x (1+1/140) = 6 msec**
(with 1/(Bkfrxln2) probability, the next record will not be in the same data bucket with the one we have already read and with 1/fo probability we need to access another index node while accessing different data node than we have read **)**

b) Time needed to find the average account balance with sequential files can be calculated as follows:
The calculation of the average includes exhaustive reading of the file, i.e. T_X.
In IBM 3380 environment, bucket size is 2400 bytes => **Bkfr = 2400/400 = 6**
# of data blocks = 200,000 / Bkfr = 200,000/6 =33,334
T_F **=** b/2 x ebt = 33,334/2 x 0.84 **= 14,000 msec = 14 sec**
T_N **= T_F = 14 sec**
T_X **=** s + r + b x ebt = 16 + 8.3 + 33,334 x 0.84 = **28,025 msec = 28 sec**

6) 10 MB of memory are available and Salzberg calculates the number of data buckets that can be accessed with a sparse (primary) B+ tree with the leaves and the parent-of-leaf level on disk, and all levels above the parent-of-leaf level are kept in memory. Also, Salzberg makes the assumption that the index (internal) nodes of the B+ tree are one block (page 150).

- We have Leaf node size = 2400 bytes
- Available memory = 10 MB
- R (record size) = 400 bytes
- Bkfr = 2400/400 = 6
- Degree of index nodes = 80
- average fan-out = 80 x 2 x 0.7 = 112

Using bottom-up approximation, for the number of leaf nodes (data buckets) bk, bk/fo number of parent-of-leaf level of the index is needed. Salzberg assumes that there are at most three levels above the parent-of-leaf level which can be seen in Figure 1 and that the number of blocks above the parent-of-leaf level is: **$bk/(fo)^2$ + $bk/(fo)^3$ + 1**
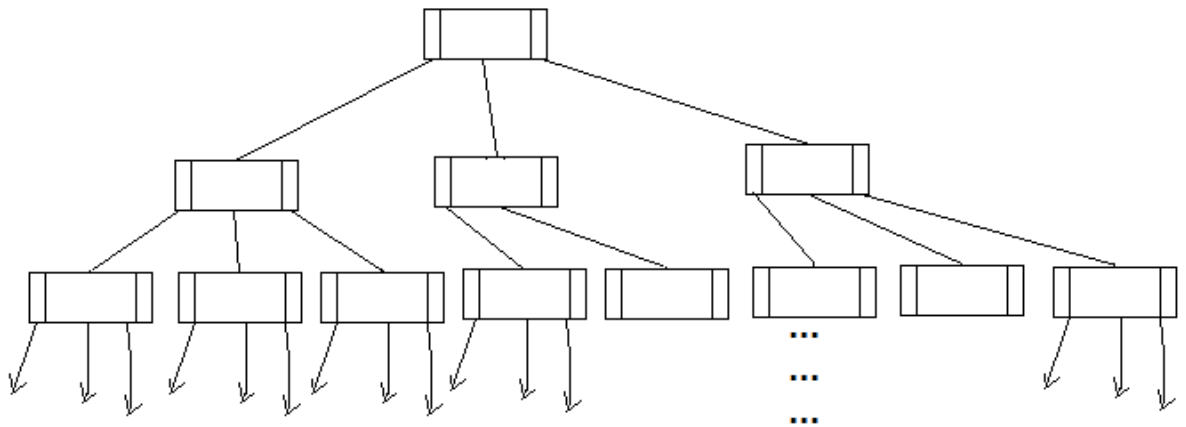

Figure 1: A B+ tree with three levels in memory

With 10 MB available memory, (10×10^6) / 2400 = 4167 index blocks can be kept, therefore if we solve the equation we get: **4167 = $bk/(fo)^2$** (we can neglect the space needed above the parent-of-parent-of-leaf level)

Then **bk = 4167 x $(112)^2$ = 52,266,667** and now we can find the number of records:
n = bk x ln2 x Bkfr = 52,266,667 x 0.7 x 6 = 219,520,000 records approximately.