Homework 1 solutions

---

1. Consider the instance of the Students relation shown below, sorted by age. Assume that these tuples are stored in a sorted file in the following order: the first 3 tuples are on page 1 and the last 3 tuples are on page 2 (each page can store up to 3 data records)

Sid	Name	Login	Age	gpa
53831	Madayan	madayan@music.com	11	1.8
53832	Guldu	guldu@music.com	12	2.0
53660	Mary	Mary@cs.com	13	3.0
53666	Jones	jones@cs.com	18	3.4
53688	Smith	smith@ee.com	19	3.2
53650	Smith	smith@math.com	19	3.8

Explain the data entries in each of the following indexes contain. If such an index cannot be constructed, say so and explain why.

1) An unclustered index on age using Alternative (1)

Not possible, since the records are already sorted by age.

2) An unclustered index on age using Alternative (2)

<11,(1,1),> <12,(1,2)> <13, (1,3)> <18, (2,1)> <19,(2,2)> <19,(2,3)>

3) An unclustered index on age using Alternative (3)

<11,(1,1),> <12,(1,2)> <13, (1,3)> <18, (2,1)> <19,(2,2),(2,3)>

4) An clustered index on age using Alternative (1)

<tuple 1> , <tuple 2)> , <tuple 3> , <tuple 4> , <tuple 5> , <tuple 6>

5) An clustered index on age using Alternative (2)

Same as 2)

6) An clustered index on age using Alternative (3)

Same as 3)

7) An unclustered index on gpa using Alternative (1)

Not possible

An unclustered index on gpa using Alternative (2)

<1.8, (1,1)>, <2.0, (1,2)>, <3.0, (1,3)> , <3.2, (2,2)>, <3.4, (2, 1)>, <3.8, (2,3)>

8) An unclustered index on gpa using Alternative (3)

Same as 8).

9) An clustered index on gpa using Alternative (1)

Not possible

10)An clustered index on gpa using Alternative (2)

Not possible

11)An clustered index on gpa using Alternative (3)

Not possible

---

2. Consider a disk with the following characteristics (these are not parameters of any particular disk unit): sector size B=512 bytes, each block contains 15 sections and the number of sections per track=450, number of tracks per surface=400. A disk pack consists of 20 double-sided disks.

(a) How many cylinders are there?

400

(b) What is the capacity of a track, a cylinder and the disk?

Capacity of a track: 450 * 512 = 230,400

Cylinder: 20 * 2 * 230,400 = 9,216,000

Disk: 9,216,000 * 400 = 3, 686, 400, 000

(c) Suppose the disk platters rotate at a speed of 2400 revolutions per minute), what is the average rotational delay?
If one track of data can be transferred per revolution, what is the transfer rate?

Maximum rotational delay: (1/2400) * 60 = 0.025 sec

The average rotational delay: 0.025/2 = 0.0125.

Transfer rate: 230,400/0.025 = 9,216,000

(d) Suppose the average seek time is 30 msec. How much time does it take (on the average) to locate and transfer a single block given its block address?

One track contains 450/15 = 30 blocks.

Transmission time of one track of data is 0.025 sec, so the transmission time of one block is 0.025/30 sec

Total time:

seek time + average rotational delay + transfer time

= 0.03 + 0.0125 + 0.025/30 = 0.0433

(e) Calculate the average time it would take to transfer 30 random blocks.

30 * 0.0433 = 1.299

(f) Calculat the time it would take to transfer 30 consecutive blocks.

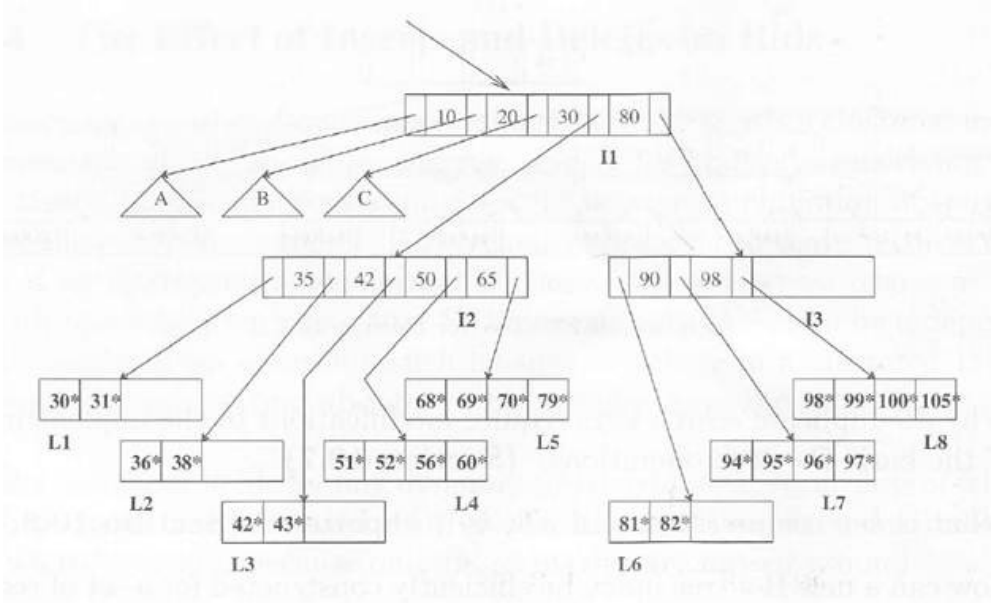
30 blocks are in the same track

seek time+average rotational delay + transfer time of 30 blocks (one track)

0.03 + 0.0125 + 0.025 = 0.0675

---

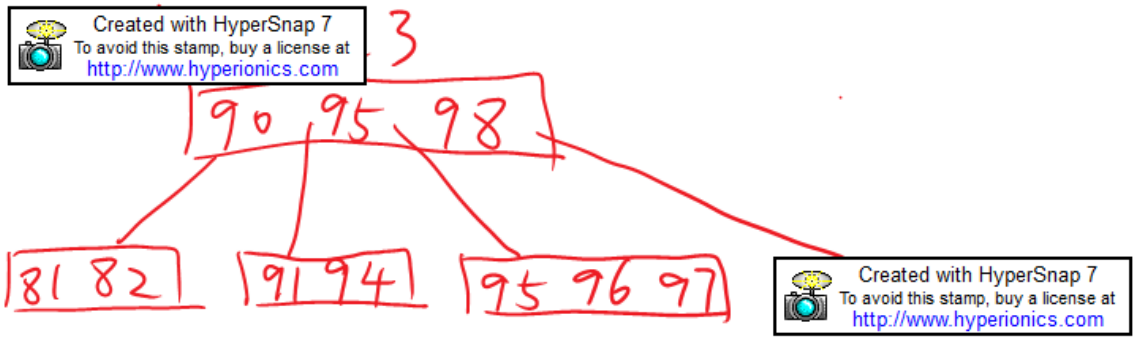
3. **Consider the B+ tree index shown in the following figure, which uses Alternative 1 for data entries. Each intermediate node can hold up to five pointers and four key values. Each leaf can hold up to four records, and leaf nodes are doubly linked as usual, although these links are not shown in the figure. Answer the following questions.**



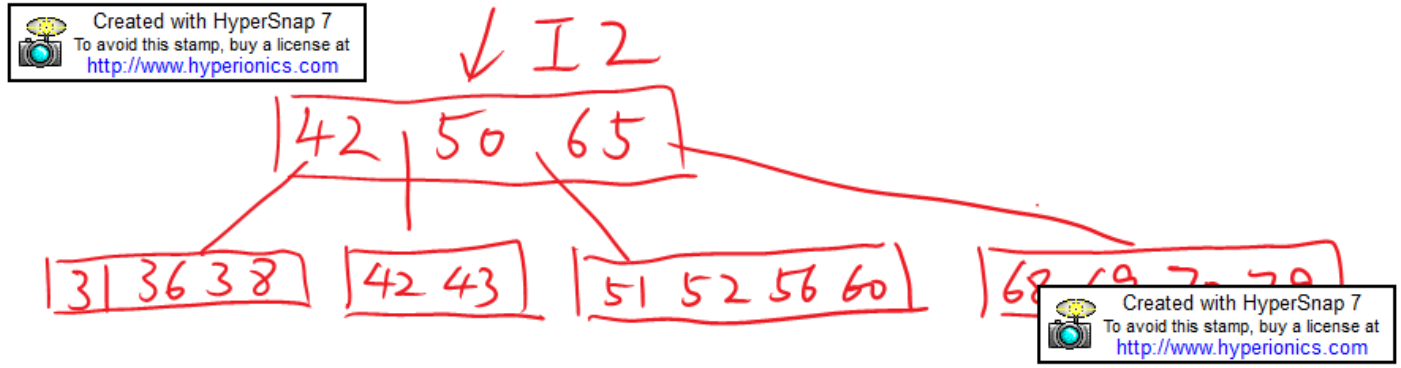
1) Name all the tree nodes that must be fetched to answer the following query: "Get all records with search key greater than 32 and less than 81"

I1, I2, L1-L5, I3, L6

2) Insert a record with search key 91 into the tree



3) Delete the record with search key 30 from the original tree



4) Name a search key value such that inserting it into the original tree would cause an increase in the height of the tree

keys X such that inserting X would increase the height of ey in the range [65..79] would suffice. A key in this range would go in L5 if there were room for it, but since L5 is full already and since it can't redistribute any data entries over to L4 (L4 is full also), it must split; this in turn causes I2 to split, which causes I1 to split, and assuming I1 is the root node, a new root is created and the tree becomes taller.

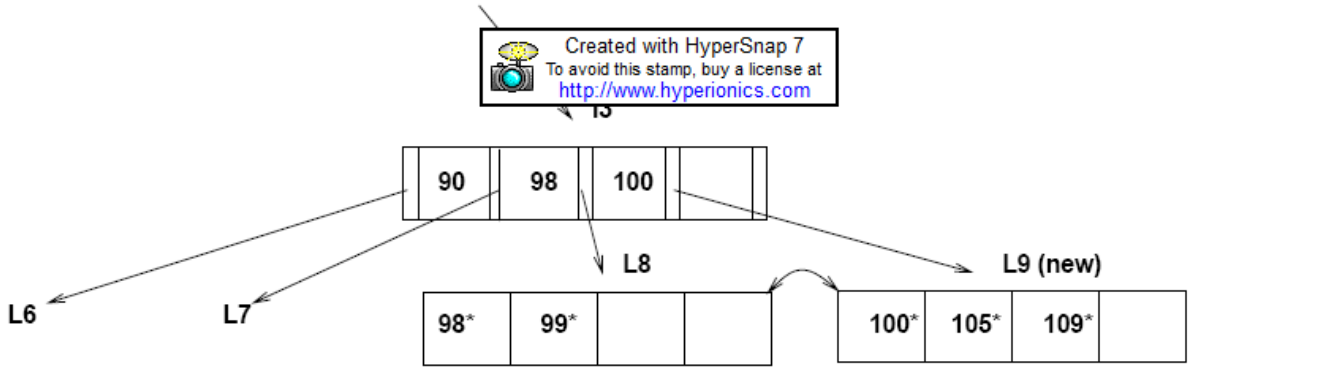
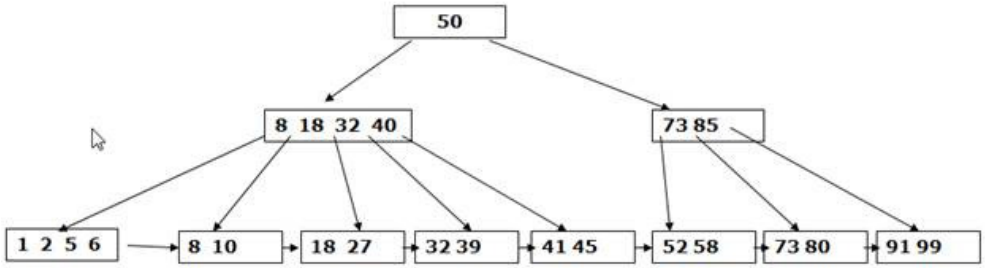


Figure 10.11

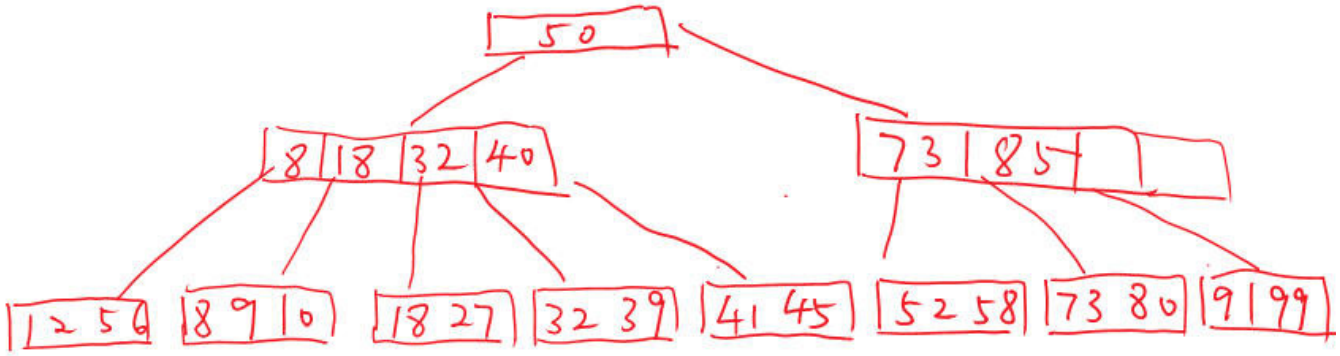
5) Suppose that this is an ISAM index. What is the minimum number of insertions needed to create a chain of two overflow pages? Give an example of such insertions.

7. If this is an ISAM tree, we would have to insert at least nine search keys in order to develop an overflow chain of length three. These keys could be any that would map to L4, L5, L7, or L8, all of which are full and thus would need overflow pages on the next insertion. The first insert to one of these pages would create the first overflow page, the fifth insert would create the second overflow page, and the ninth insert would create the third overflow page (for a total of one leaf and three overflow pages).

4. Consider the B+ tree shown in the following figure. Show the structure of the new tree after the following insertions and deletions.



1) Insert 9*



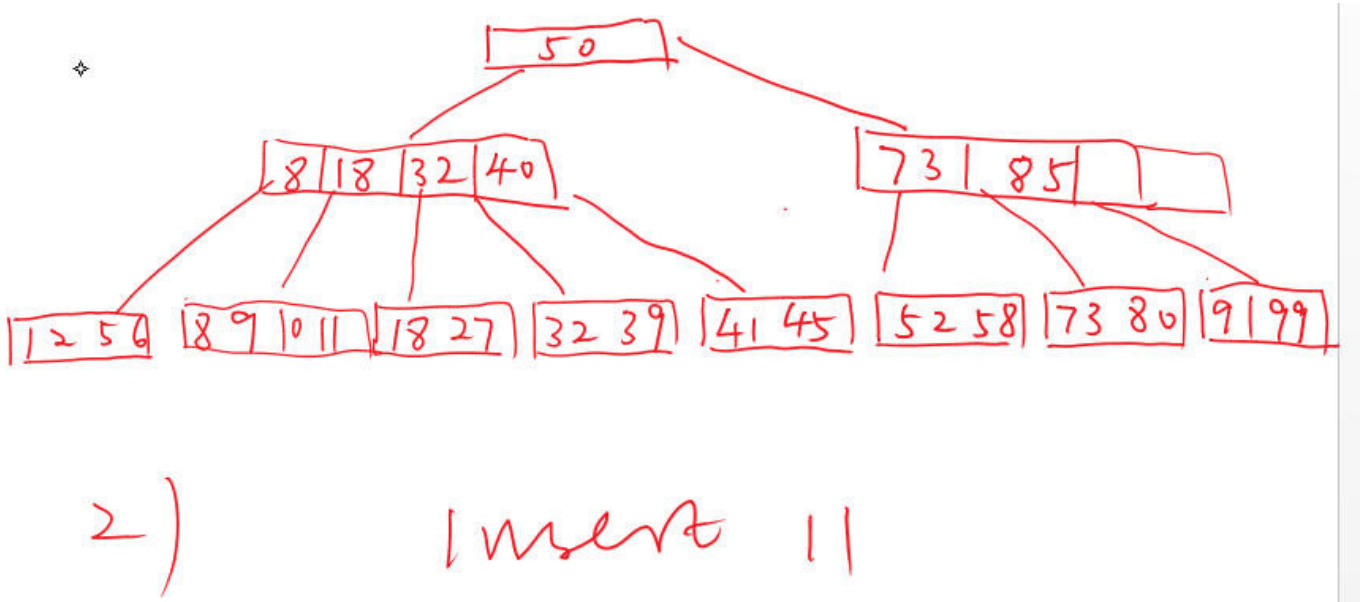
1) Insert 9

2)

3)

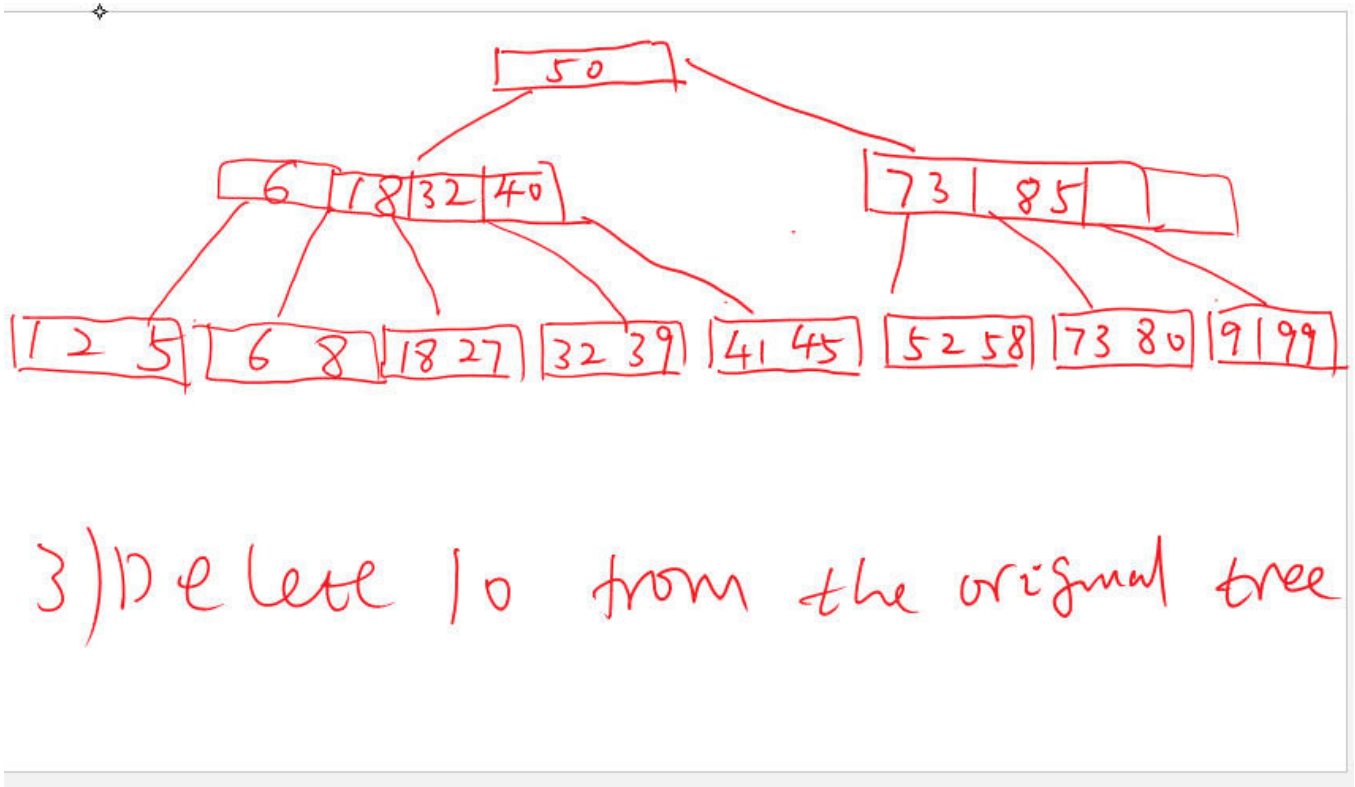
4) Insert 11* after 1)

5)

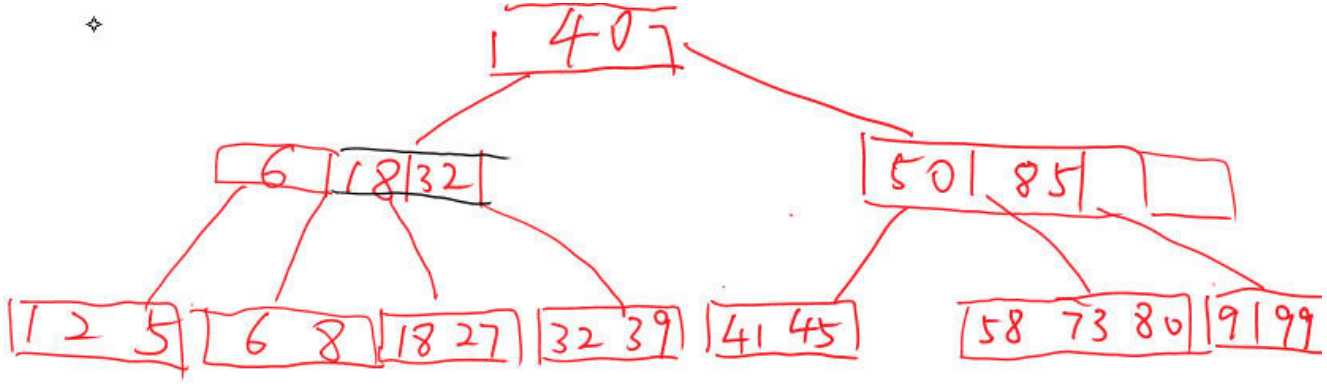


6)

7) 3 **Delete 10* from the original tree**



8)
9)
10)
11)4)Delete 52* after 3)
12)

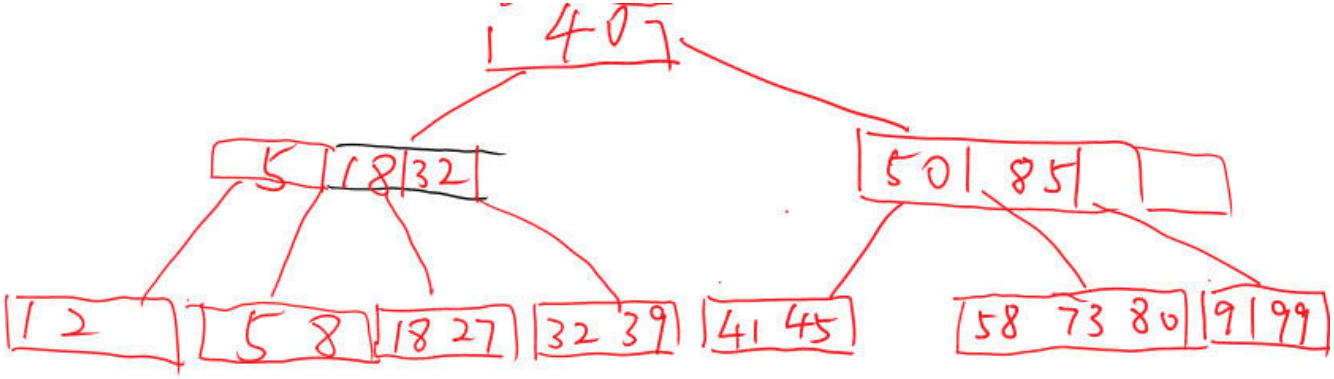


4) Delete 52 after 3)

13)

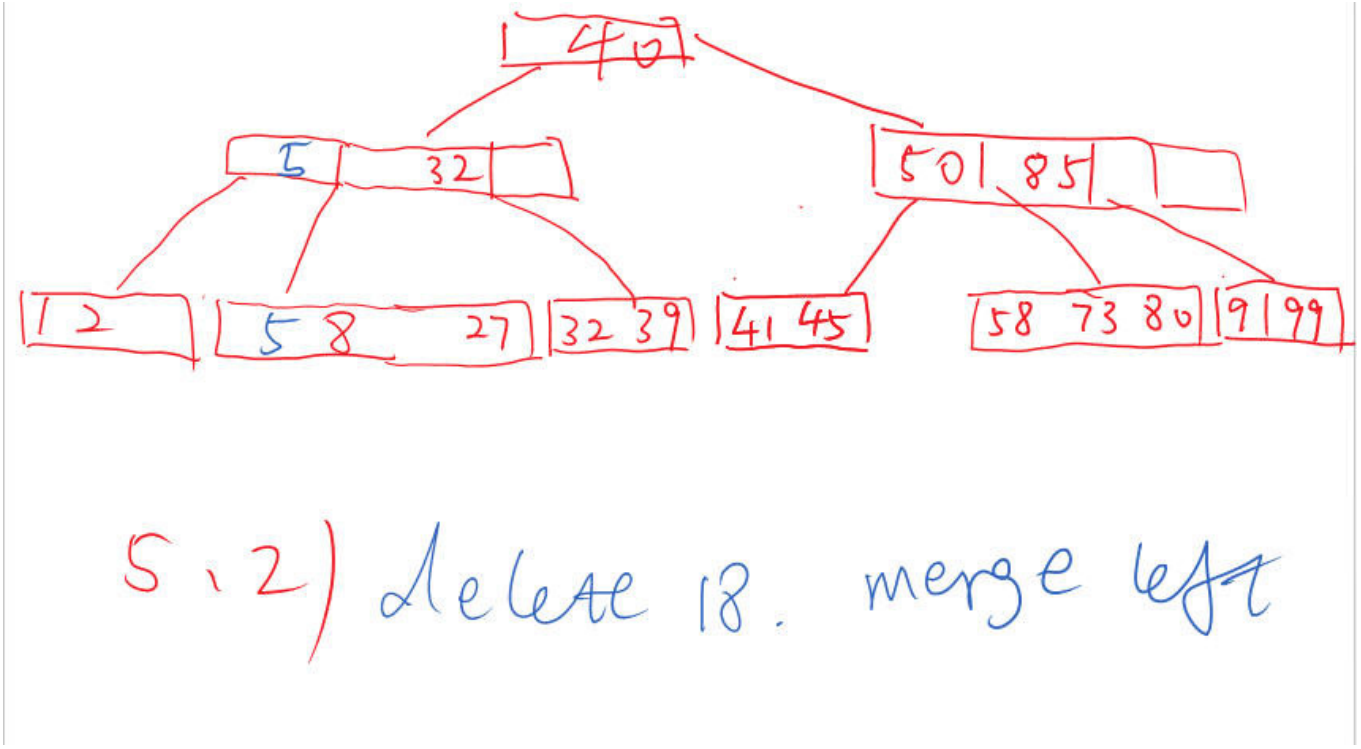
14)Successively delete 6, 18, 27, 41, 45, 39, 58 after 4)

15)

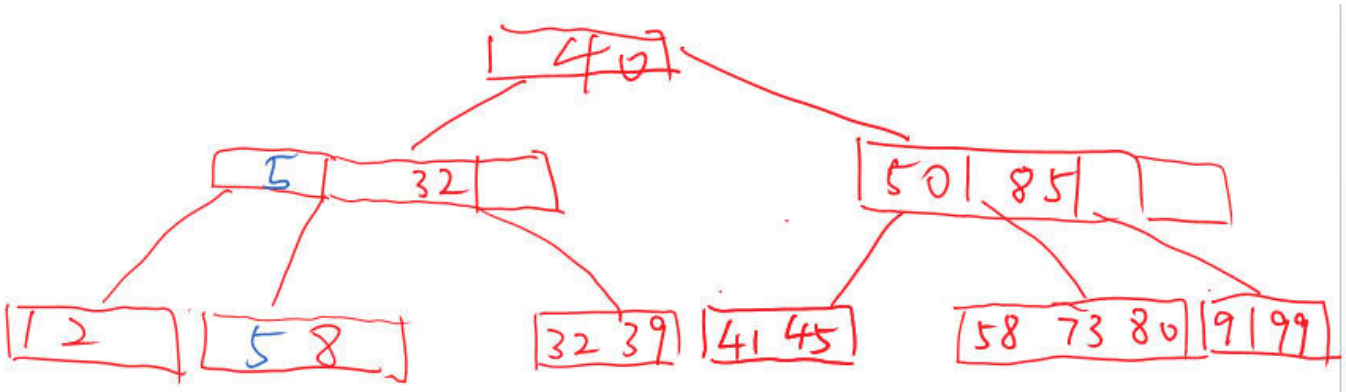


5.1) Delete 6

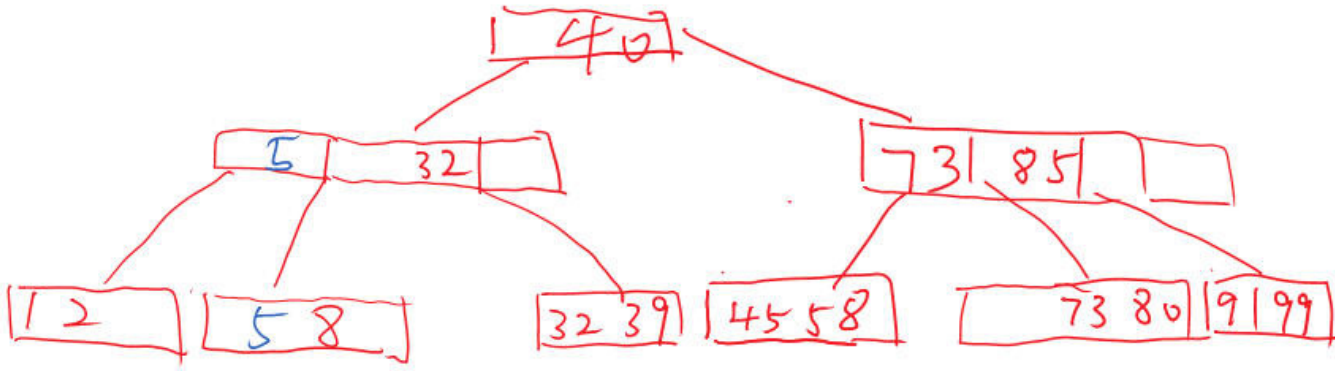
16)
17)



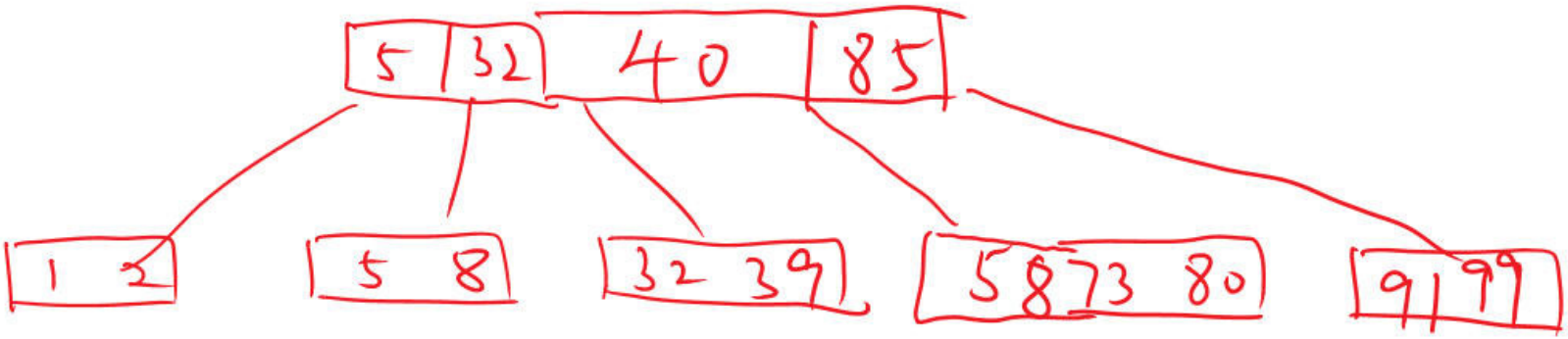
5.2) delete 18. merge left



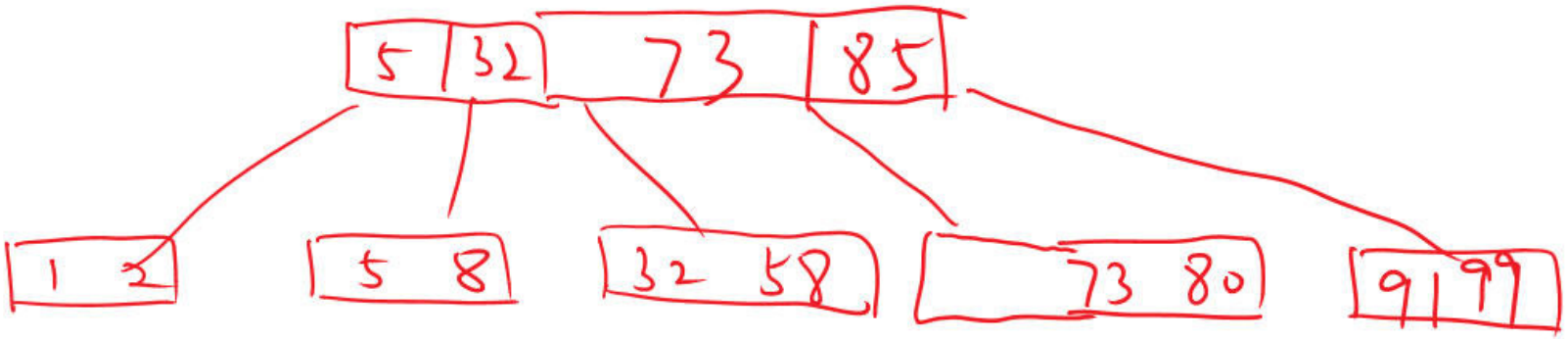
5.3) delete 27.



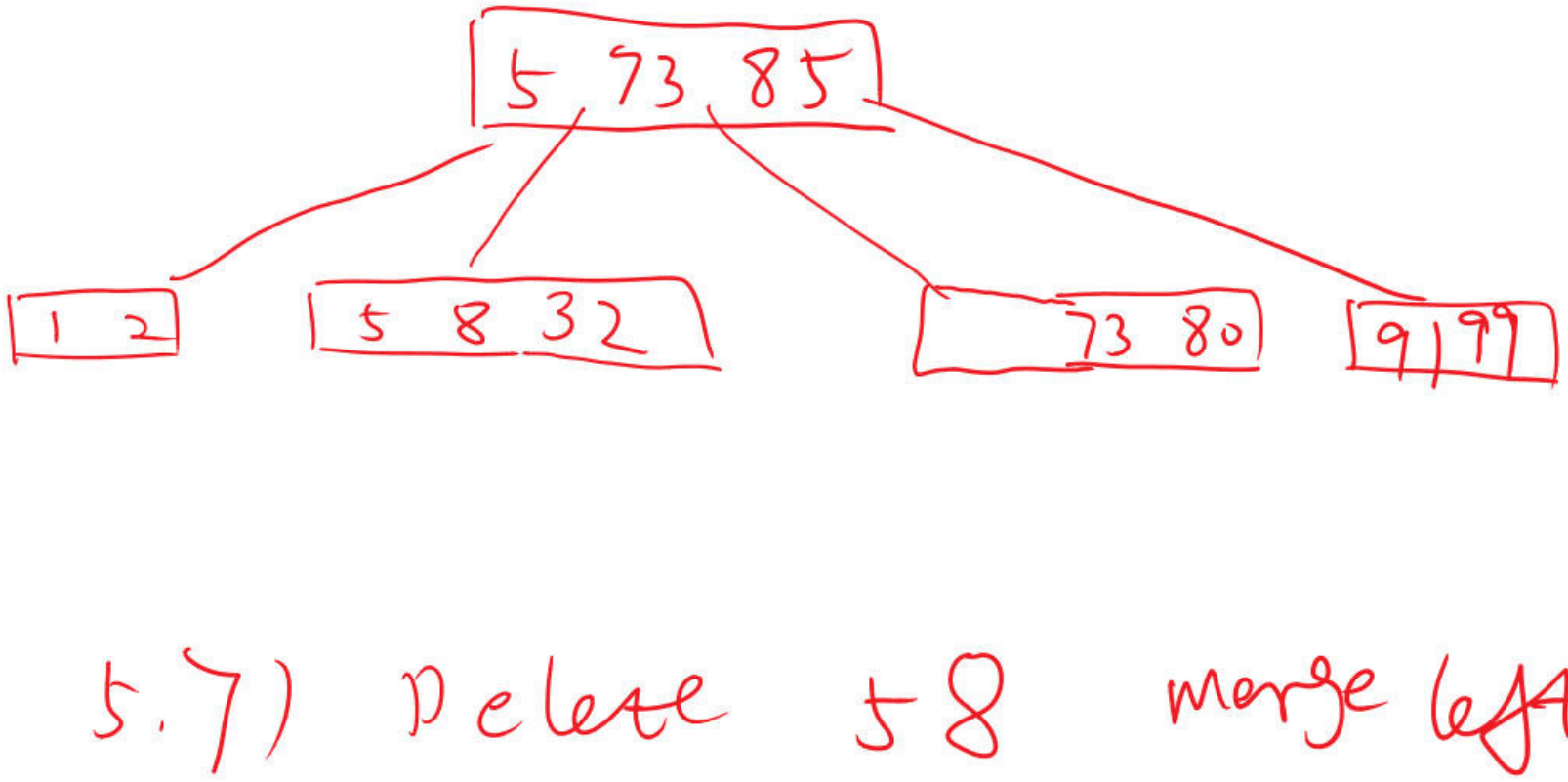
5.4) delete 41



5.5) Delete 45



5.6) Delete 39



5(1pt). Assume that you have just built a dense B+ tree index using Alternative 2 on a heap file containing 40,000 records. The key field for this B+ tree index is 20-bytestring, and it is a candidate key. Pointers (i.e., record ids and page ids) are (at most) 10-byte values. The size of one disk page is 2000 bytes. The index was built in a bottom-up fashion using the bulk loading algorithm, and the nodes at each level were filled up as much as possible.

1) How many levels does the resulting tree have?

Since the index is a primary dense index, there are as many data entries in the B+ tree as records in the heap file. An index page consists of at most 2d keys and 2d+1 pointers. So we have to maximize d under the condition that $2d \cdot 20+(2d+1) \cdot 10 \leq 2000$. The solution is d = 33, which means that we can have 6 keys and 67 pointers on an index page. A record on a leaf page consists of the key field and a pointer. Its size is 20+10=30 bytes. Therefore a leaf page has space for (2000/30)=66 data entries. The resulting tree has $\log_{67}(40{,}000/66) + 1 = 3$ levels

2) For each level of the tree, how many nodes are at that level?

Since the nodes at each level are filled as much as possible, there are 40000/66 = 607 leaf nodes (on level 3). (A full index node has 2d+1 = 67 children.) Therefore there are 607/67 = 10 index pages on level 2, and there is one index page on level 1 (the root of the tree).

3) How many levels would the resulting tree have if key compression is used and it reduces the average size of each key in an entry to 10 bytes?

$2d \cdot 10+(2d+1) \cdot 10 \leq 2000$. The solution is d = 49, which means that we can have 98 keys and 99 pointers on an index page. A record on a leaf page consists of the key field and a pointer. Its size is 10+10=20 bytes. Therefore a leaf page has space for (2000/20)= 100 data entries. The resulting tree has $\log_{99}(40{,}000/100) + 1 = 2$ levels

4) How many levels would be the resulting tree have without key compression but with all pages 70% full?

Since each page should be filled only 70 percent, this means that the usable size of a page is $2000 \cdot 0.70 = 1400$ bytes. Now the calculation is the same as in part 1 but using pages of size 1400 instead of size 2000. An index page consists of at most 2d keys and 2d + 1 pointers. So we have to maximize d under the condition that $2d \cdot 20+(2d+1) \cdot 10 \leq 1400$. The solution is d = 23, which means that we can have 46 keys and 47 pointers on an index page. A record on a leaf page consists of the key field and a pointer. Its size is 20+10=30 bytes. Therefore a leaf page has space for (1400/30)= 46 data entries. The resulting tree has $\log_{47}(40{,}000/46) + 1 = 3$ levels.
