The assignment is to be turned in before Midnight (by 11:59pm) on January 25th, 2018. You should turn in the solutions to this assignment as a pdf file through the TEACH website. The solutions should be produced using editing software programs, such as LaTeX or Word, otherwise they will not be graded. Trees can be drawn on paper and scanned.

1: File Structures (1 point)

1. Consider a file with a large number of *Customer(id, name, birth-date)* records. Assume that users frequently search this file based on a the field *id* to find the values of *name* or *birth-date* for customers whose information is stored in the file. Moreover, assume that users rarely update current records or insert new records to the file. Which file structure, heap versus sorted, provides the fastest total running time for users' queries over this file? Explain your answer (0.5 point).
   For this example the best file structure would be a sorted file structure sorted based on the ID of the Customer record. I would choose this file structure because a sorted file structure offers better run time efficiency for search type operations. This file structure does have some disadvantages when it comes to writing to the file structure however as indicated in the question we don't expect this operation to happen often.

2. Consider a file with a large number of *Transaction(id, customerID, productID, amount)* records, which keeps track of the purchases made by a customer on various products. Assume that users frequently insert new records into this file. Users also query this file to compute the total amount of money each customer has spent on her purchases, similar to a SQL query with *Group By customerID.* Which file structure, heap versus sorted, provides the fastest total running time for users' queries over this file? Explain your answer (0.5 point).

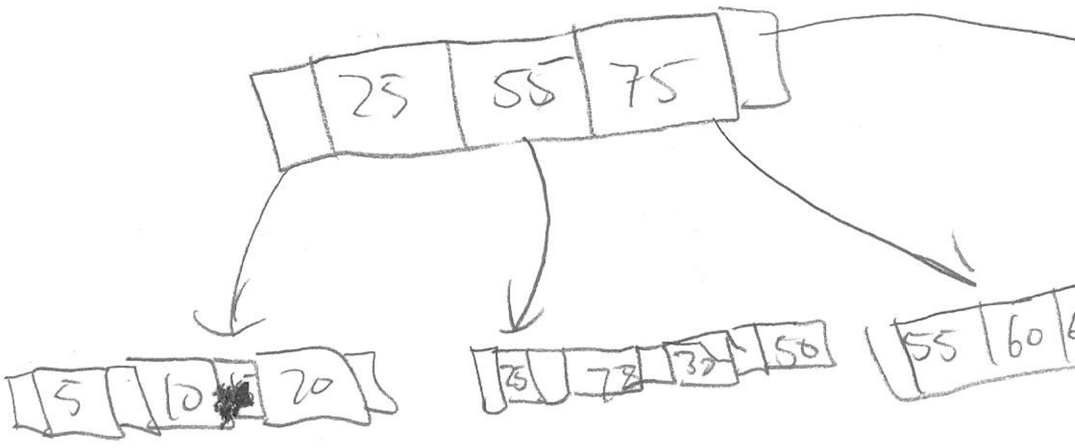
   In this instance I would probably use a heap type files structure. As indicated by the question there will be a large number of records added to this file. A heap type file structure will offer the best performance when it comes to writing to the file. However, this file structure will be slower at querying using a group by statement as mentioned in the question prompt. However, if you are using a group by statement you are probably reading a decent amount of the records in the file any ways and likely wont suffer enough to justify giving up the write performance of a heap file structure.

2: B+ Tree Indexing (2 points)

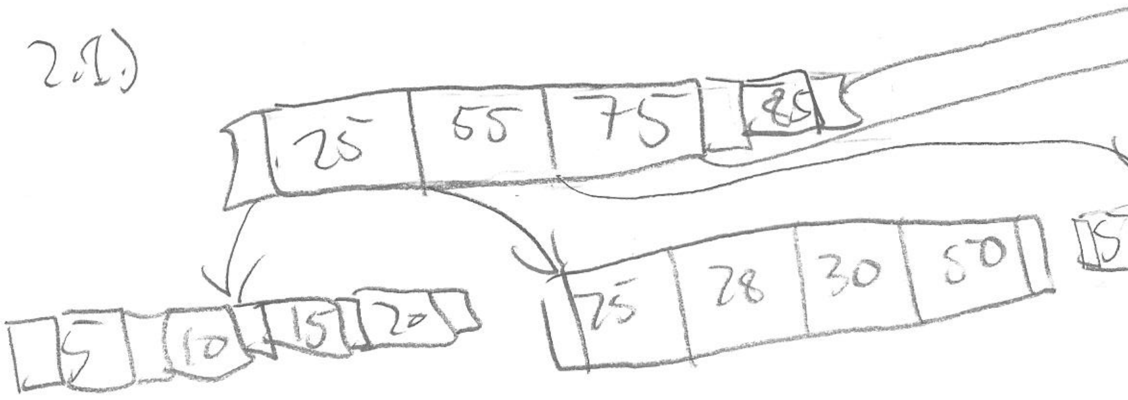
Consider the B+ tree index shown in Figure **??**. Each intermediate node can hold up to five pointers and four key values. Each leaf can hold up to four pointers to data, and leaf nodes are doubly linked as usual, although these links are not shown in the figure. Answer the following questions.

1. Show the B+ tree that would result from inserting a record with search key 95 into the tree.

   solution:
   See the graphic below for the solution to this problem.

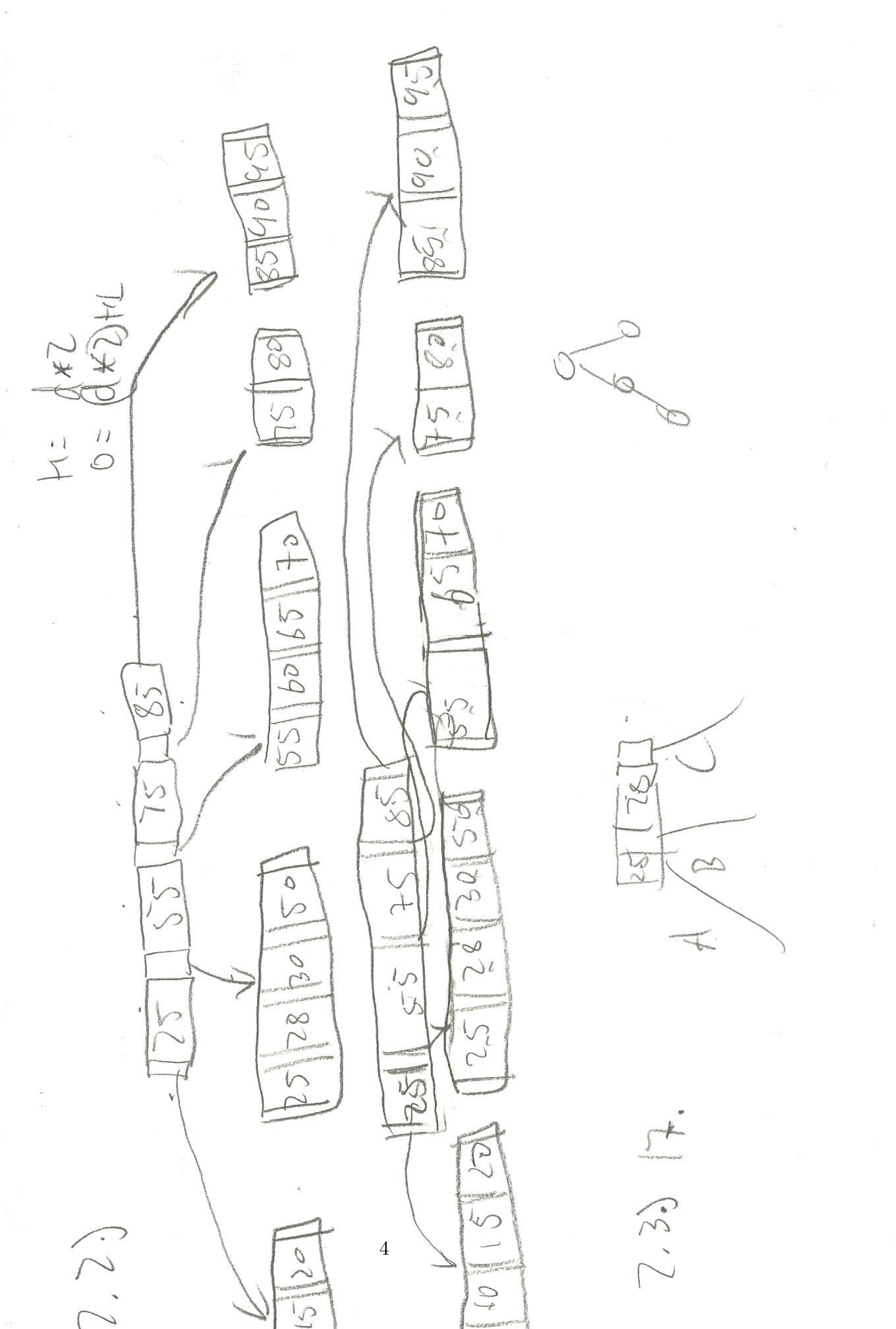


2.1.)



2. Use the result/solution tree from (1) and Show the B+ tree that would result from deleting the record with search key 60.

solution:
See graphic below for solution.


2.2.)
H = d*2
O = (d*2)+L
25 55 75 85
15 20
25 28 30 50
55 60 65 70
75 80
85 90 95
2.3.) 17.
A
B
C
25 78

3. Name a search key value such that inserting it into the result/solution tree from (1) would cause an increase in the height of the tree.

   solution:
   If you insert a value into either the left most or second to left most node (Between 5 and 30) would cuase hight to be added to the tree.

4. What can you infer about the contents and the shape of A, B and C subtrees?

   solution :
   We know that the A will contain values less than 10. We also know that b will contain key values between 10 and 19 and C will contain values between 20 and 29. We can also say that A, B and C must be leaf's because the tree must be balanced. Finally, we can say that each node must have between 2 and 4 keys and can at most have 5 pointers.

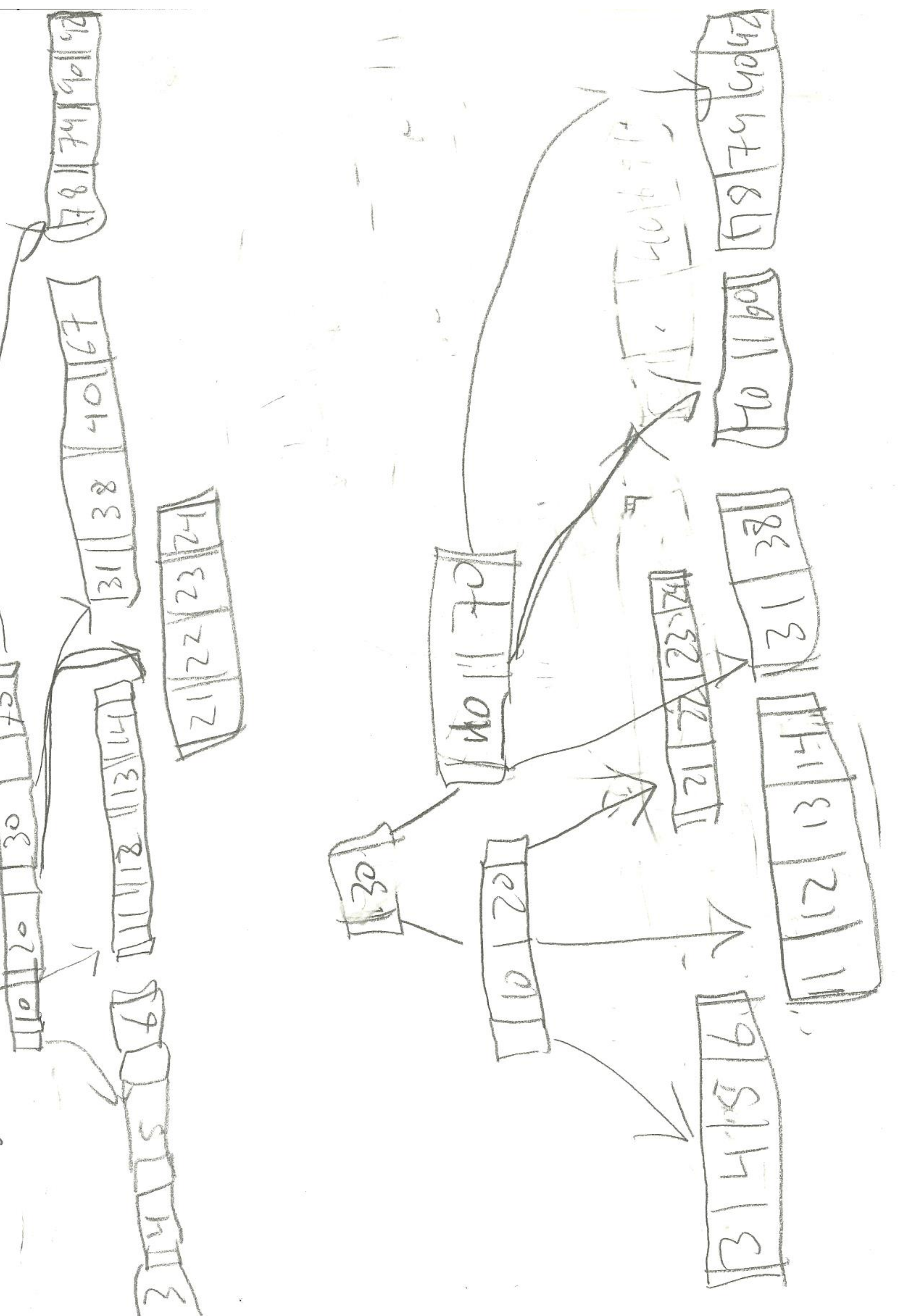
3: B+ Tree Indexing (1 point)

Suppose that a block can contain at most four data values and that all data values are integers. Using only B+ trees of degree 2, give examples of each of the following:

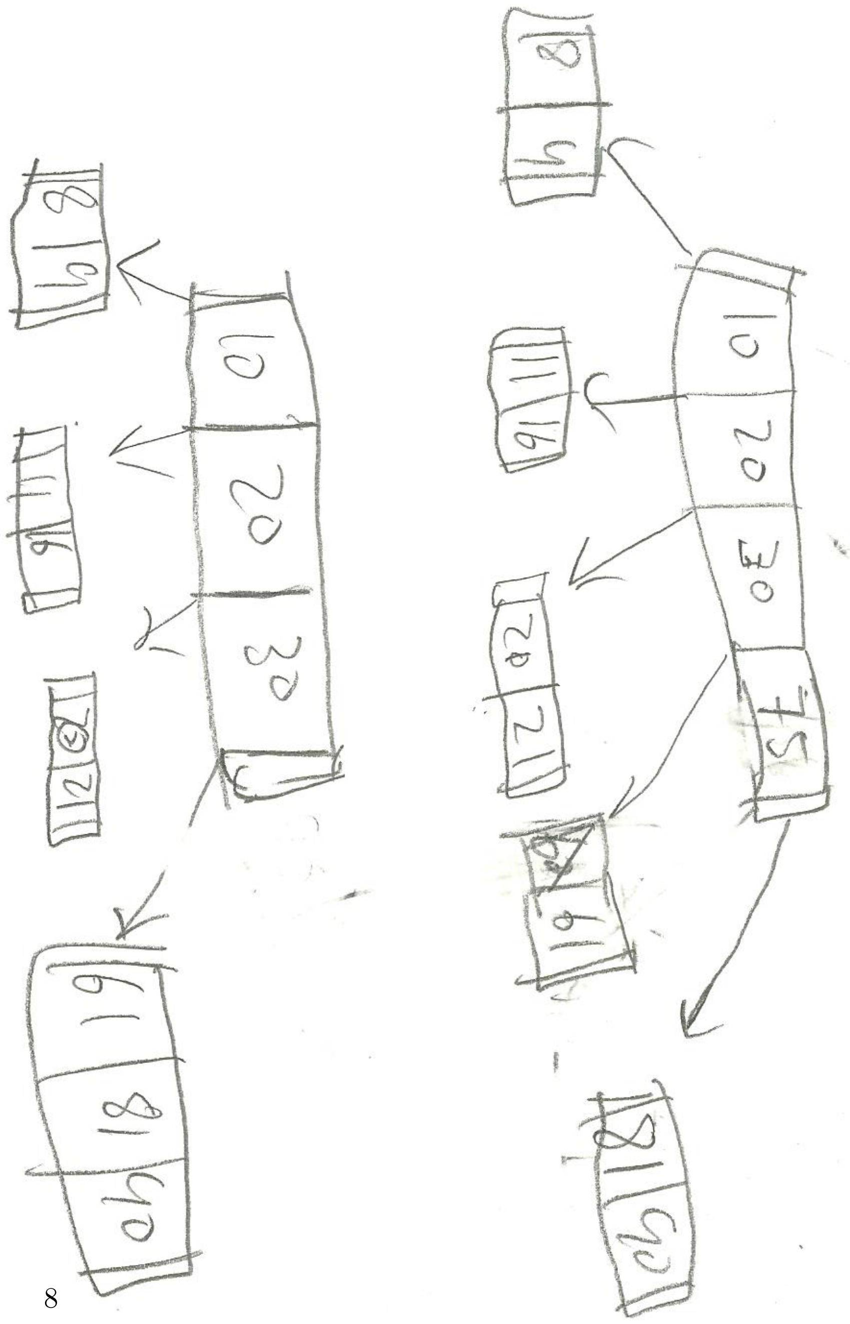
1. A B+ tree whose height changes from 2 to 3 when the value 60 is inserted. Show your structure before and after the insertion.

   solution is shown below in graphic.

3.1.)


10 | 20 | 30 | 72
3 | 4 | 5 | 6
11 | 12 | 13 | 14
21 | 22 | 23 | 24
31 | 38 | 40 | 67
78 | 84 | 90 | 92
40 | 70
30
10 | 20
3 | 4 | 5 | 6
11 | 12 | 13 | 14
21 | 22 | 23 | 24
31 | 38
40 | 60
78 | 84 | 90 | 92

2. A B+ tree in which the deletion of the value 60 leads to a redistribution. Show your structure before and after the deletion.


10
20
30
75

4: B+ Tree Indexing (1 point)

Consider the instance of the Students relation shown in Figure **??**.

1. To reduce the number of I/O access in index search, each B+ tree node should fit in a block. Let *sid* be an integer requiring 16 bits. Let a pointer require 32 bits. If the block size is 28 bytes (consisting of 8 bits), what is the maximum degree of the B+ tree index on *sid* so each B+ tree node fit in a block?

   Solution:
   Shown below is an equation for the block size given the Key size and the Record pointer size. We will use this to solve this problem.

$$k_v = KeyValuesize$$
$$r_p = Recordpointersize$$
$$b_s = blocksize$$
$$d = degree$$
$$2dk_v + (2d+1)r_p <= b_s \quad (1)$$

   Given this equation and the values given in the question prompt we can say the follow:

$$2d * 16 + (2d+1) * 32 <= 28 * 8$$
$$32d + 32 * (2d+1) <= 224$$
$$d + (2d+1) <= 7$$
$$3d <= 6$$
$$d <= 6 \quad (2)$$

   As shown by the equation above the degree of this tree can be no larger than 2.

2. Show a B+ tree index on *sid* of degree calculated in part 1 for all records in Figure **??**.
   solution shown below in figure.

H_r = Key/Value Size
R_p = Record Pointer Size
B_s = Block Size

$2d * 8 + (2d+1) * 16 = 4064$

$2d * H_v + (2d+1) * R_p <= B_s$

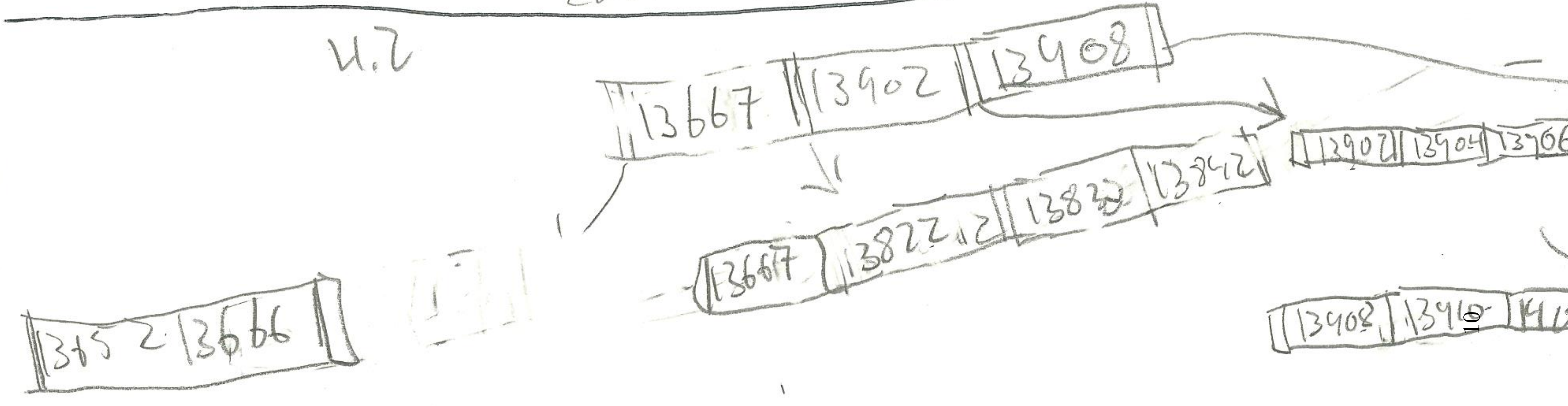
→ version

R_p = Rec
BS = Block

$2d * 8 + (2d+1)$

$d2 * ku + (2d+1)$

4.2



13667
13902
13908
13752
13666
13667
13822
13832
13842
13902
13904
13906
13908
13910