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).

   (solution)
   The file structure that provides the fastest total running time in this situation would be sorted. Sorting a list is a time consuming task but since the current records are rarely updated nor recieve new records very often we dont have to worry about sorting too much. Overall we save time once it sorted because searches are then much quicker.

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).

   (solution)
   Sorting is faster for searching but since they are expecting a high number of insertions, a heap would be the best option for this situation. Since this database will mainly be used to keep a record of all new purchases, insertion would be a priority over search.

2: B+ Tree Indexing (2 points)

Consider the B+ tree index shown. 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.

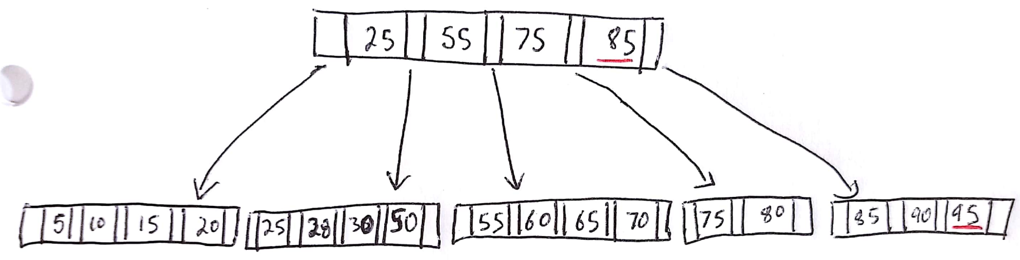

Figure 1: Tree for question 2.1

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

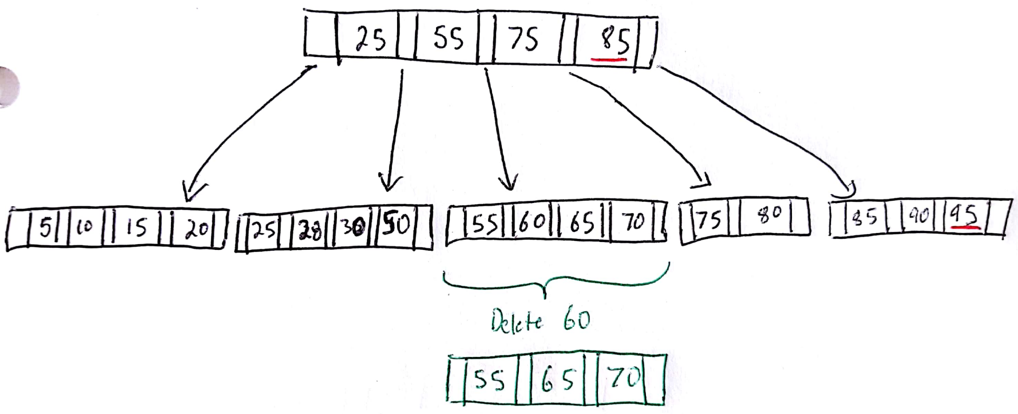

Figure 2: Tree for question 2.2

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)
   Any value below 75 would cause the resulting tree to increase in height because all of those nodes in the second layer and the root node are at maximum capacity. Adding another key

to the root node would force it to split and hence gain a new root node.

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

   (solution)
   We can infer that subtree A contains keys less than 10, subtree B contains keys more than or equal to 10 and less than 20, and subtree C contains keys more than or equal to 20 and less than 30. We can also infer that, since there is one node with 2 keys and one with 4, that d=2 which means A, B, and C each have 2-4 keys and 3-5 pointers in them.

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.

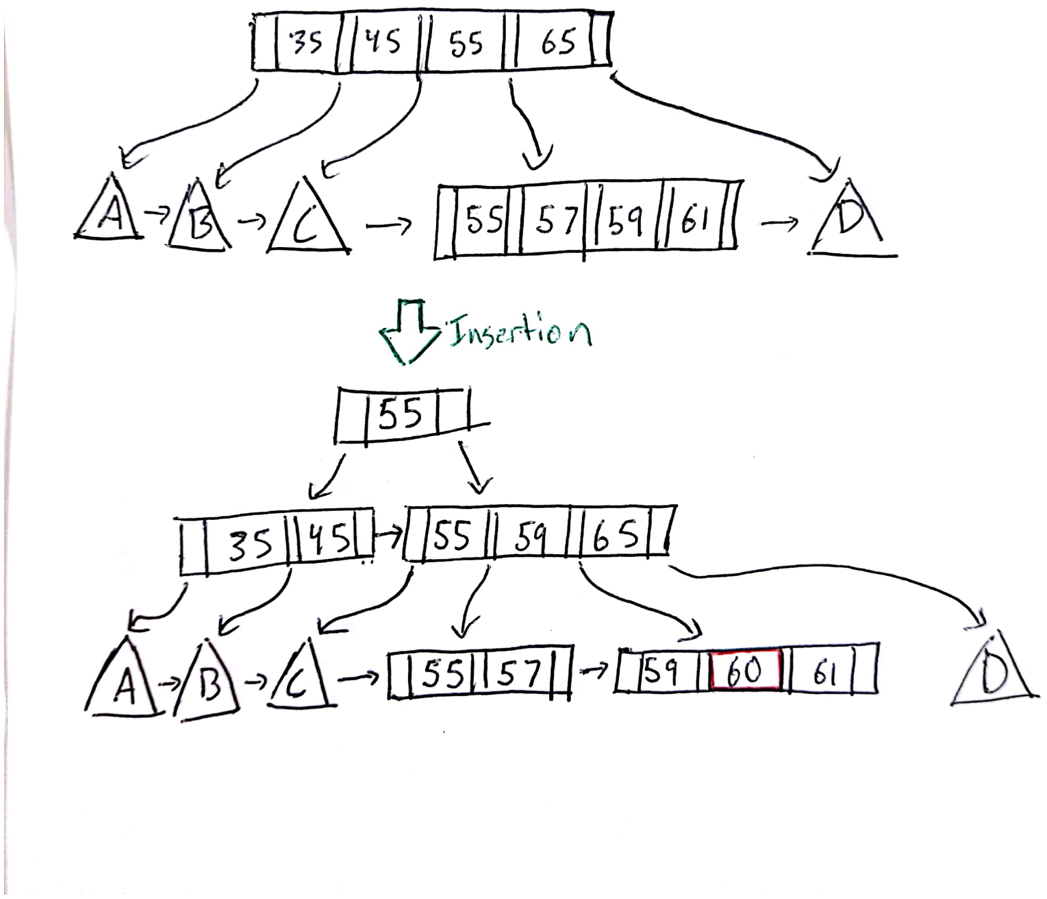

Figure 3: Tree for question 3.1

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

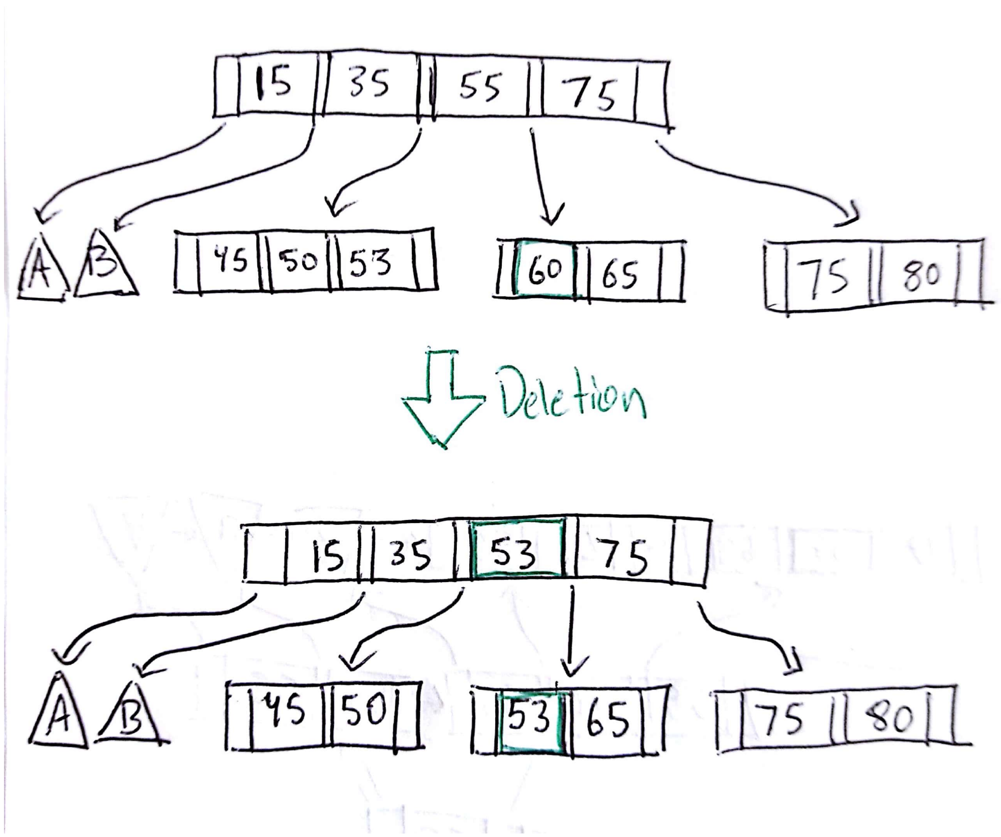

Figure 4: Tree for question 3.2

4: B+ Tree Indexing (1 point)

Consider the instance of the Students relation shown.

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)
   (2d)(2 bytes) + (2d+1)(4 bytes) <= 28 bytes
   4d + 8d + 4 <= 28
   12d <= 24
   d <= 2

2. Show a B+ tree index on *sid* of degree calculated in part 1 for all records.

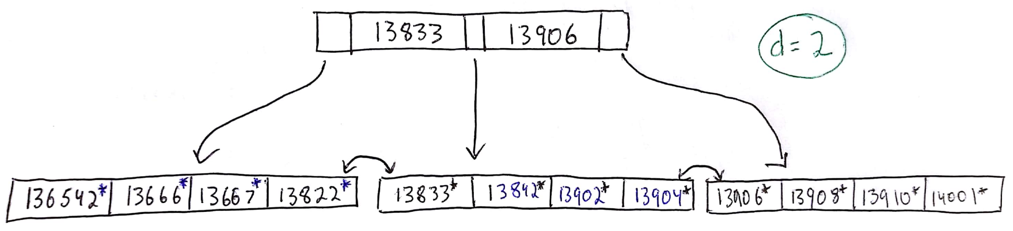

Figure 5: Tree for question 4.2