EECS 484 Database Management Systems
Homework 3 (100 points)
Due Mar 16th.

Please read the following instructions before starting the homework. Failing to follow the submission rules will lead to -4 points (e.g., wrong file format, wrong group submission).

You are *allowed* to work on this homework in pairs.

Deliverables: You need to submit all your solutions in a single PDF **on CANVAS**. Your solutions can be either handwritten or created electronically, as long as they are clear. If you are working in pair, make sure to **JOIN THE SAME GROUP** on Canvas (you need to join the group every time for different homework and projects) and make only one group submission.

No late days for homework! *If you miss the due date, you get 0 points. No exceptions on this.*

By submitting this homework, you are agreeing to abide by the Honor Code:
I have neither given nor received unauthorized aid on this assignment, nor have I concealed any violations of the Honor Code.

UPDATE & CLARIFICATION

3.4

Question 2: Add "size of record = 1KB".
Question 3: Change the subquestion label.

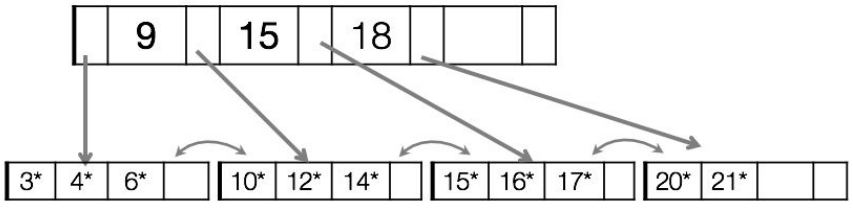
3.12

Here are some clarification about the order of tree in HW3:
Question 2: the order of tree is applied to both inner nodes and leaf nodes.
Question 6: the order of tree is applied to inner nodes only, and you need to calculate the number of data entries in leaf nodes by yourself.

Q1: B+ Tree Operations (21 points)

For the questions in this part, assume these rules for B+ tree operations:
- Each node must contain between d <= m <= 2d entries (where m is the number of entries) with the exception of the root node
- The left pointer points to values that are strictly less than the key value.
- During redistribution, favor the right node over the left when both are possible redistribution partners
- During the splitting of a node, the left node will have one more value than the right node
- For insertions, favor redistribution over splitting
- For deletions, favor merging over redistribution
- Only redistribute one entry at a time

Draw out the resulting B+ tree after EACH of the the following chains of operations is performed on the given B+ tree.



The operations are:

1. Insert 11*, 13*

2. Insert 9*
3. Delete 20*
4. Insert 5*
5. Insert 1*, 2*, 7*, 8*
6. Delete 14*, 15*

Draw what the tree should look like at the end of EACH chain of operations--you should have 6 B+ tree drawings. Note that you will be continually updating your tree after each set of operations, NOT restarting with the original tree each time.

Q2. Clustering and IO cost (20 points)

Consider a table Sailors, where there is a B+ tree index on attribute Age, and the following query:

SELECT * FROM Sailors ORDER BY Age;

Furthermore, assume that the first level (root) of the B+ tree is kept in memory at all times, and that the B+ tree has the following properties:

The properties of this B+ tree are:
- total size of records = 2MB
- size of record = 1KB
- size of page = 4KB
- the tree has an order of 4
- the fill factor is 67%

Assume 1MB = 1024KB. Assume only root node is in memory and nothing else is saved in memory. We try to fit all records in the smallest number of pages possible and each node is in it's own page. Assume alternate 2 (as described in lecture) is used

Answer the following questions:

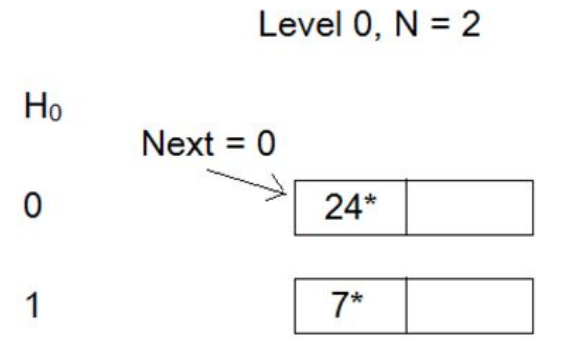
2.1) What is the number of record pages and leaf nodes? What is the height of the B+ tree?

2.2) How many IOs (worst case) will it take to run this query if our index was clustered?

2.3) How many IOs (worst case) will it take to run this query if our index was unclustered?

Q3. Linear Hashing (14 points)

Assume we initialize the Linear Hashing index as follows (the hash function used is given below the Linear Hashing index):



h_i (value) = h (value) mod (2^iN)

3.1) What is the minimum number of data entries that can be inserted, until we have 4 buckets and level is equal to 1?

3.2) Perform the following insertions on the linear hashing index: 33, 54, 11, 26, 42. Assume that we use the same algorithm discussed in the lecture. Draw the index structure after each insertion, and show the next pointer and level at each step.

Q4. Extendible Hashing (15 points)

In Q3, we looked at linear hashing, but what about extensible hashing? Suppose we have an extendible hash index that uses the hash function:

hashvalue = last *d* bits of the binary version of the given number

where *d* is the global depth. We will be using a bucket capacity of 2 for this problem. The initial global depth and local depth are 1. Assume that we use the same hashing function as above (except i is now the global depth)

Our index is initially empty. Then the following insertions occur: 39, 16, 26, 49, 18, 13, 41. Draw the index structure after each insertion. Be sure to clearly show the global and local depths and all bucket pointers at every step.

Q5. Blocked IOs (15 points)

Suppose we have a data set consisting of a 1TB (1T=1024G) file that we need to sort. We have 8GB of memory (RAM) to assist in sorting. The page size of the system is 32KB and the buffer block size is 32KB. Assume that replacement sort is used for the initial pass.

Answer the following questions accordingly:

5.1) How many passes do we need on average to sort the dataset, if we only use replacement sort?

5.2) How many I/Os are needed if we take 15 passes to sort the dataset?

5.3) How many passes do we need if we use blocked I/Os?

Q6. Bulk Loading (15 points)

In this question we will compare the IO cost between bulk loading and repeated insertions.

We make the following assumptions:
- 10,000 records
- A page size of 1024 bytes
- A tree of order 12
- A pointer/value/search key takes 8 bytes

- Leaf nodes hold as many data entries as possible using alternative 2 (as described in lecture).
- Each node takes up exactly one page of memory.
- Our buffer pool can hold exactly as many pages as the height+1 of the fully created B+tree.
- After bulk loading, all inner nodes are half full.

Answer the following questions:

6.1) Before bulk loading, we need to sort our record pointers. If we use general external merge sort, how many IOs will it take to do the sort?

6.2) Calculate the IO cost to build the B+ tree. Include the cost of "committing" the changes/saving to disk after completion. For simplicity, assume the buffer pool is flushed before this step. Also assume you can control which pages stay in memory.

6.3) Assume that the average number of IOs taken for each repeated insert is half the height of the final tree. Which method (bulk loading or repeated inserts) requires fewer IOs? By how much?