Name	CWID

Quiz 1

1

Due: October 13th 2021, 11:59pm

Please leave this empty! 1.1 1.2 1.3 Sum

Instructions

- **You have to hand in the assignment using your blackboard**
- **This is an individual and not a group assignment. Fraud will result in 0 points**
- For your convenience the number of points for each part and questions are shown in parenthesis.
- There are **2** parts in this quiz
  1. Disk Organization
  2. Index Structures

Part 1.1 Disk Organization (Total: 18 Points)

Question 1.1.1 Disk Access (18 Points)

Consider a disk with a sector size of 512 bytes, 2000 tracks per surface, 25 sectors per track, five double-sided platters, and average seek time of 5 msec. Suppose that a block size of 2048 bytes is chosen. Suppose that a file containing 100,000 records of 200 bytes each is to be stored on such a disk and that no record is allowed to span two blocks.

1. What is the capacity of a track in bytes? What is the capacity of each surface? What is the capacity of the disk?

2. How many records fit onto a block?

3. How many blocks are required to store the entire file?

4. If the file is arranged sequentially on the disk, how many surfaces are needed?

5. How many records of 100 bytes each can be stored using this disk?

6. If pages are stored sequentially on disk, with page 1 on block 1 of track 1, what page is stored on block 1 of track 1 on the next disk surface?

Part 1.2 Index Structures (Total: 40 Points)

Question 1.2.1 B^+-tree Construction (10 Points)

Assume that you have the following table:

Item

SSN	name	age
7	Pete	13
3	Bob	23
5	John	49
11	Joe	45
2	Alice	77
17	Lily	3
19	Manny	33
23	Gertrud	29
29	Heinz	14
39	Sammy	34

Create a `B`$^+$`-tree` for table **Item** on key *SSN*. Assume that the tree is initially empty and values are added in the order shown in the table above. Construct `B`$^+$`-tree` for the cases where the number of pointers that will fit in one node is as follows:

a. Six

b. Eight

Write down the resulting `B`$^+$`-tree` after each step and when splitting or merging nodes follow these conventions:

- **Leaf Split**: In case a leaf node needs to be split during insertion and n is even, the left node should get the extra key. E.g, if $n = 2$ and we insert a key 4 into a node [1,5], then the resulting nodes should be [1,4] and [5]. For odd values of n we can always evenly split the keys between the two nodes. In both cases the value inserted into the parent is the smallest value of the right node.
- **Non-Leaf Split**: In case a non-leaf node needs to be split and n is odd, we cannot split the node evenly (one of the new nodes will have one more key). In this case the "middle" value inserted into the parent should be taken from the right node. E.g., if $n = 3$ and we have to split a non-leaf node [1,3,4,5], the resulting nodes would be [1,3] and [5]. The value inserted into the parent would be 4.
- **Node Underflow**: In case of a node underflow you should first try to redistribute values from a sibling and only if this fails merge the node with one of its siblings. Both approaches should prefer the left sibling. E.g., if we can borrow values from both the left and right sibling, you should borrow from the left one.

Scratch Page - This page is intentionally left blank

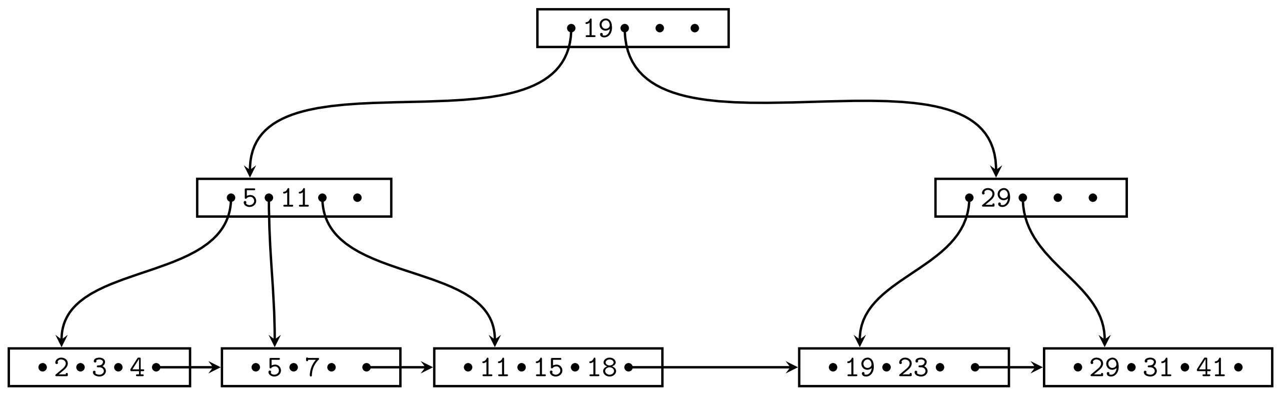
Scratch Page - This page is intentionally left blank

Question 1.2.2 Operations (30 Points)

Given is the `B`$^+$`-tree` shown below (n = *3*). Execute the following operations and write down the resulting `B`$^+$`-tree` after each operation:

insert(35), insert(1), delete(19), insert(16), delete(5), delete(29)

Use the conventions for splitting and merging introduced in the previous question.



Scratch Page - This page is intentionally left blank

Scratch Page - This page is intentionally left blank