Jerry Perez

Problem 1

a.

- file has 200 blocks
- 5 buffers available

$M = 5$

$B(R) = 200$

$$\#\text{phases} = \underbrace{1}_{\text{preparation}} + \text{ceil}\left[\underbrace{\log_{M-1}}_{\text{merge phases}} \left(\overbrace{N}^{\text{blocks}} / \overbrace{M}^{\text{buffers}}\right)\right]$$

$$= 1 + \text{ceil}[\log_4 (200/5)]$$
$$= 1 + \text{ceil}[\log_4 40]$$
$$= 1 + \text{ceil}[2.661]$$
$$= 1 + 3$$
$$= 4$$

$$\#\text{of I/O's} = \underbrace{4 * 2}_{\text{read \& write}} * B(R)$$
$$= 8 * 200$$
$$= \boxed{1{,}600}$$

Sorted runs

- buffer with 5 pages
- file has 200 pages

preparation → Pass 0: [200/5] = 40 sorted runs with 5 pages each

Pass 1: [40/4] = 10 sorted runs of 20 pages each

Pass 2: [10/3] = 4 sorted runs of 60 pages each

(last run is only 20 pages)

b.

- file has 200 blocks
- 8 buffers available

$M = 8$

$B(R) = 200$

$$\#\text{phases} = \underbrace{1}_{\text{preparation}} + \text{ceil}\left[\underbrace{\log_{M-1}}_{\text{merge phases}} \left(\overbrace{N}^{\text{blocks}} / \overbrace{M}^{\text{buffers}}\right)\right]$$

$$= 1 + \text{ceil}[\log_7 (200/8)]$$
$$= 1 + \text{ceil}[\log_7 25]$$
$$= 1 + \text{ceil}[1.654]$$
$$= 1 + 2$$
$$= 3$$

$$\#\text{of I/O's} = \underbrace{3}_{\text{read \& write}} * 2 * B(R)$$
$$= 6 * 200$$
$$= \boxed{1,200}$$

Sorted runs

- buffer with 8 pages
- file has 200 pages

preparation → Pass 0: $[200/8]$ = 25 sorted runs with 8 pages each

Pass 1: $[25/7]$ = 4 sorted runs of 56 pages each
(last run is only 32 pages)

Pass 2: Sorted file of 200 pages

Problem 2

1. yes, each index entry contains 20 bytes and a pointer to the record being 8 bytes, for a total of 28 bytes. Since there are 1,000,000 blocks with 20 records each, we obtain 20,000,000 records. A dense index has exactly 1 entry for each record, meaning there will also be 20 million records in the index file. Each block can contain 292 index records (8192/28). The entire index would be 68,494 blocks (20,000,000/292).

2. yes, with a sparse index, all entries contain a pointer to the start of a block in the data file. Since we have 1,000,000 blocks and 20 records in each block, we would need 50,000 index entries (1,000,000/20) which would occupy 172 blocks (50,000/292 index records)

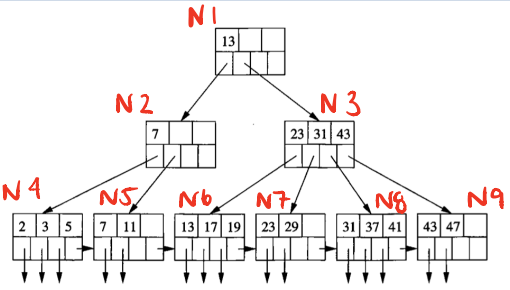
3. yes, each entry would contain k2 (20 bytes) as well as a pointer (8 bytes). The data file itself contains 20 million records total which would mean the index file would have 28 million bytes since we are dealing with a dense index. Each block could contain 292 index records (8192/28) which would mean the index file would be 95,891 blocks (28,000,000/292)

4. No, a sparse index cannot be created on an unsorted file.

5. Yes, it is possible to have a second-level sparse index on the index from part 1. Since it is a sparse index, for each index entry, there will be a pointer to the first key (k1) in the block within the first-level dense index. Since the first-level index has 68,494 blocks, the second-level would need to have the same number of entries, each of which being 28 bytes long. The second-level would occupy 235 blocks (68,494/292 index records) + first-level index of 68,494 blocks to total 68,729 blocks.

6. Yes, it is possible to have a second-level sparse index on the index in part 2. Since it is a sparse index, for each index entry, there will be a pointer to the first key (k1) in the block within the first-level sparse index. Since the first-level index has 172 blocks, the second-level would need the same number of entries, which can fit in one block. 1 block from the second-level sparse index + 172 blocks from the first-level sparse index generates 173 blocks total.

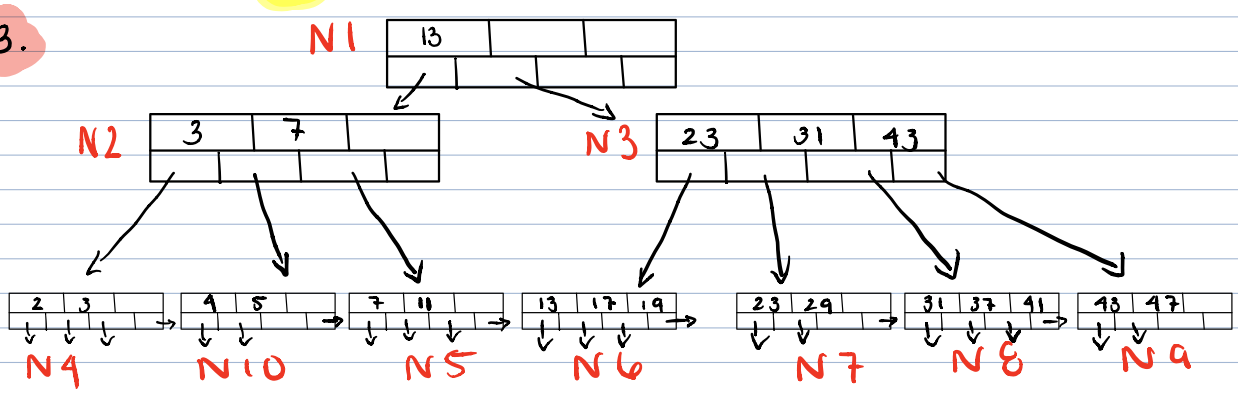
Problem 3



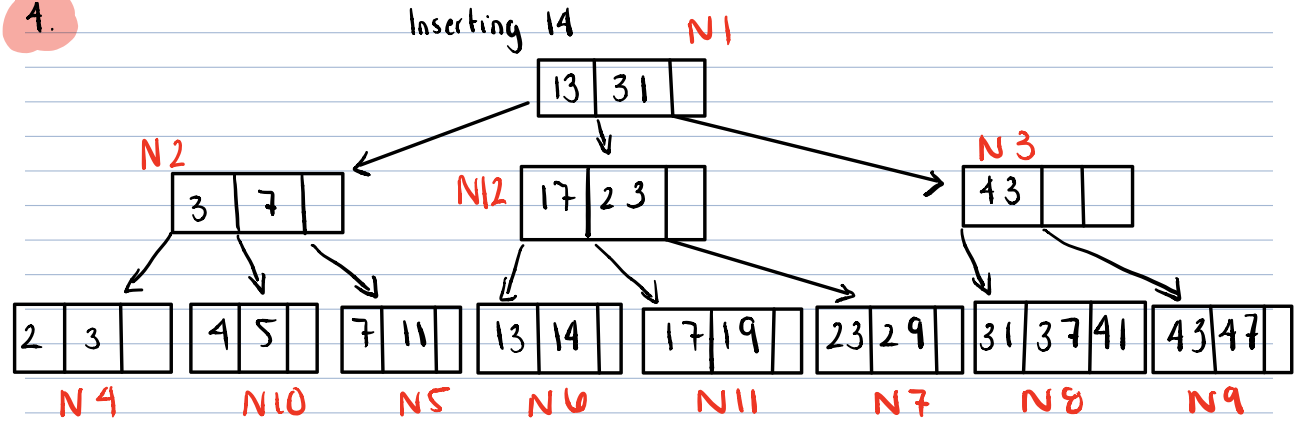
1. The root node (N1) gets accessed first, then it would follow the pointer to the right of the key to N3 since 35 > 13. Then, it would follow the pointer in between 31 and 43 to get to N8, and that is where it would notice that 35 is not in the data file.

2. The root node (N1) gets accessed first, then it would follow the pointer to the left of the key to N2 since 9 < 13. Then, it would follow the pointer to the right of 7 to get to N5. At N5, it retrieves key 11, then follows the leaf pointer to N6 where it retrieves keys 13, 17, and 19. It follows the pointer to N7 and stops there since 23 > 21.

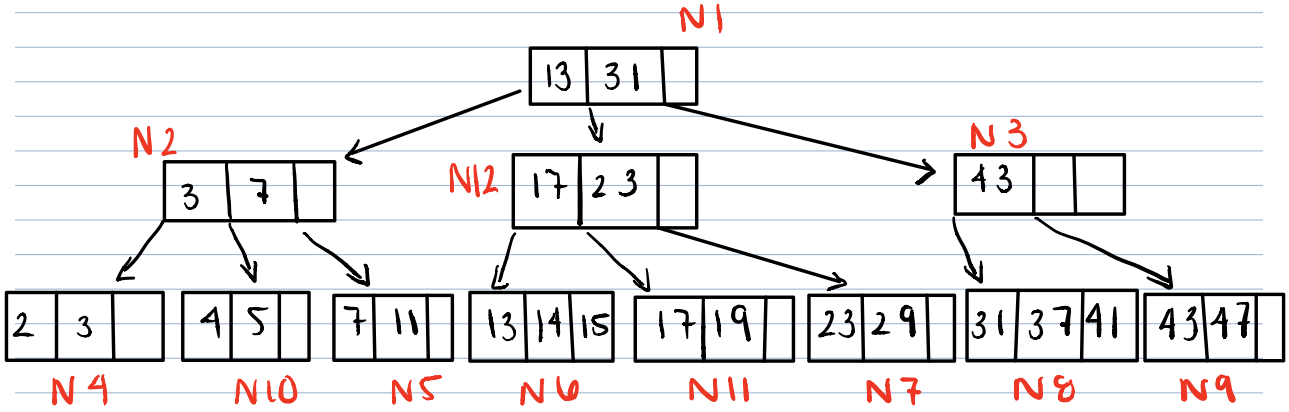
3.



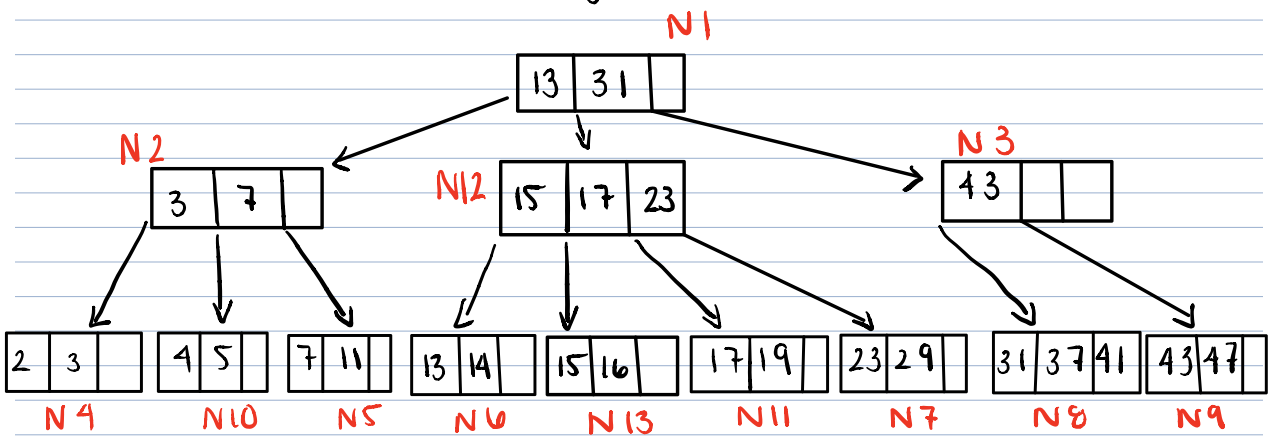
4.



Inserting 15



Inserting 16



5. The root node (N1) gets accessed first, then it would follow the pointer to the left of 13 to get to N2 since $6 < 13$. Then it would follow the pointer in between 3 and 7 to get to N10 where it wouldn't find any keys within the range. It moves right to N5 where it finds and retrieves keys 7 and 11. It moves right again to N6 where it finds and retrieves 13. At this point, it exits.

6. Deleting 23