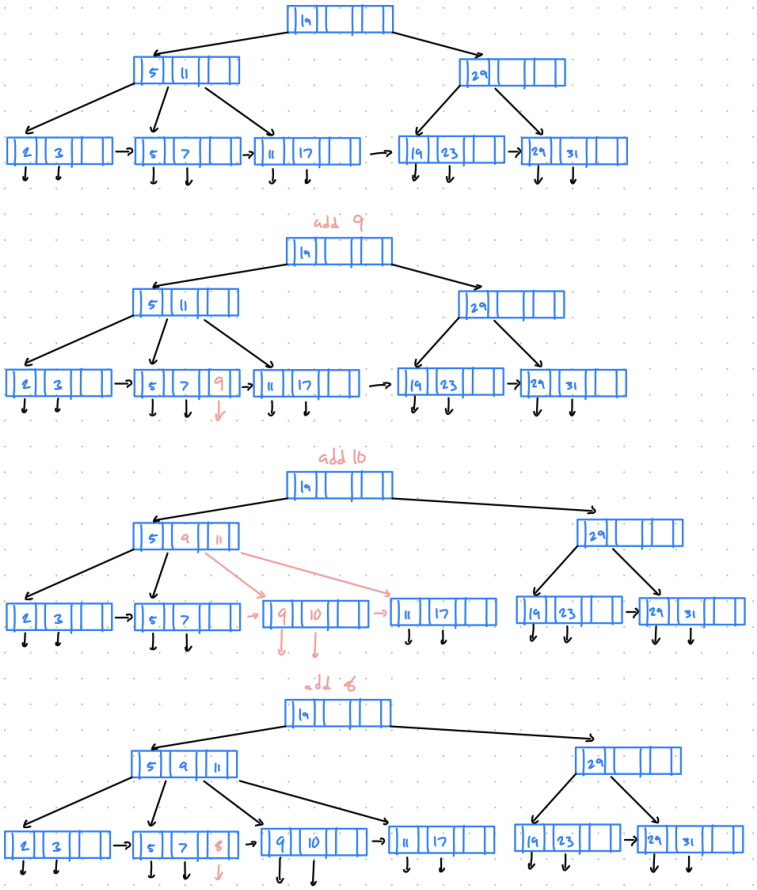
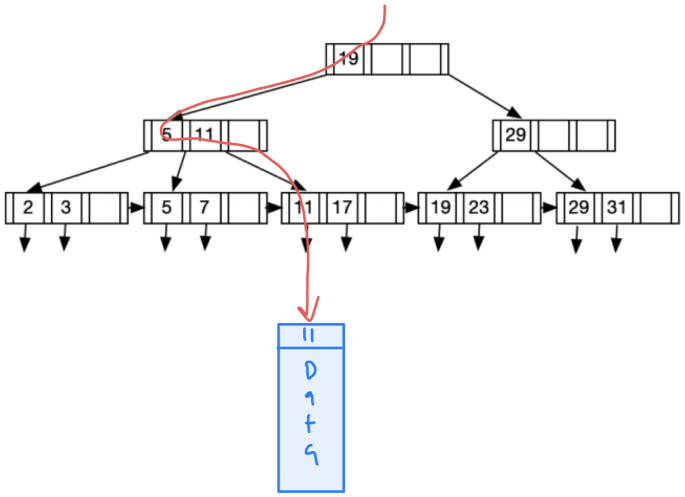
Homework 4

Exercise 1

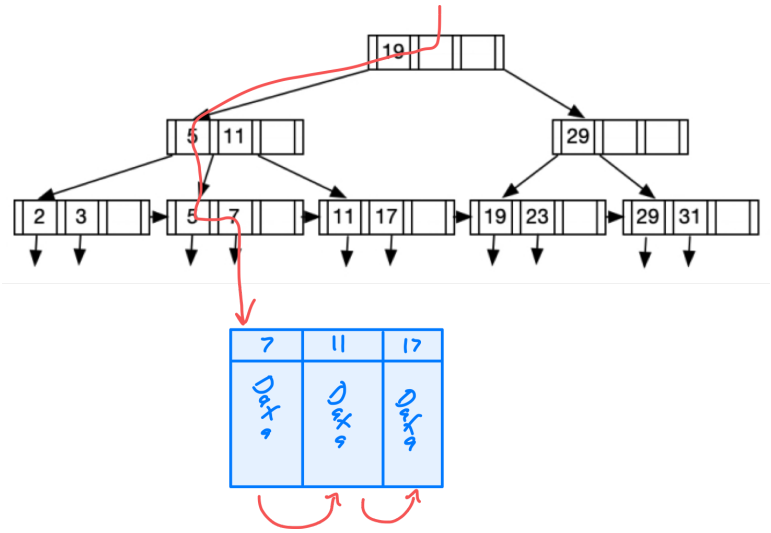
1a



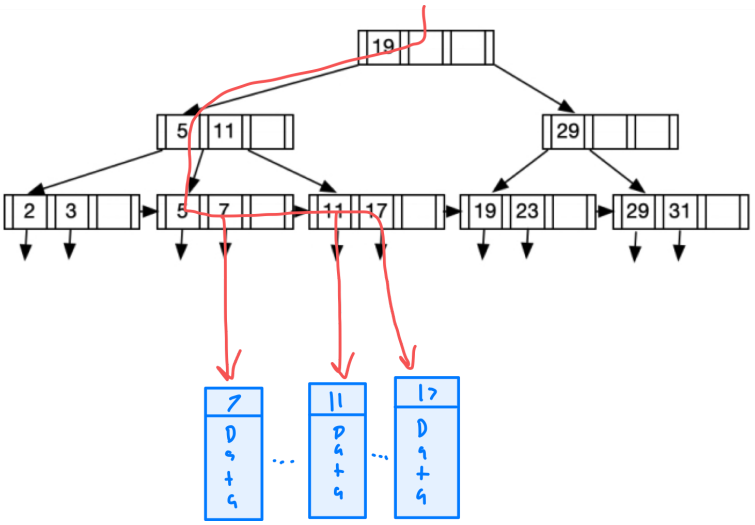
1b


19
5
11
29
2
3
5
7
11
17
19
23
29
31
11
Data

1c


19
5
11
29
2
3
5
7
11
17
19
23
29
31
7
11
17
Data
Data
Data

1d



Exercise 2

It is better to use a clustering B+ Tree to perform a range query because records on a B+ tree are ordered on disk.

Because the records are ordered in a B+ tree, when we pull a block it might contain more than just one of our records, so we can reduce the number of block transfers.

Range query

- `K` = # of keys
- `n` = # of key-values per block
- `M` = # of pointers per block
- `N` = length of the range
- `N/n` = # blocks needed to get all contiguous records (because it is a clustered B+ Tree) considering a worst case where we read all blocks fully.

We would need

$$\log_{\lceil \frac{n}{2} \rceil} K + \frac{M}{\lceil \frac{n}{2} \rceil} + 1 + \frac{N}{n}$$

block transfers with a **B+ tree**.

But with a **Hash index** we would have to take in N blocks where N is the length of the range.

We can see from comparing these two values that the number of blocks needed to be transferred for a B+ tree with a range query is much less than if we used a Hash index.

Exercise 3

Given

- `n` = number of records processed
- `b` = number of blocks read

Let

- `k` = number of keys per node
- $L = \frac{n}{\lceil \frac{k}{2} \rceil} - 1$ = number of leaf node blocks read in after traversing h (h includes the first leaf node)

The time estimate for the total worst execution case would be

$(t_T + t_s) * (h + L + n) = 92\mu s * (h + L + n)$

Exercise 4

4a

- b_r = # of blocks in relation R = `20k/50 = 400` blocks
- b_s = # of blocks in relation S = `50k/20 = 2500` blocks
- n_r = # of tuples in R = `20K tuples`
- n_s = # of tuples in S = `50K tuples`

Block Transfers

$b_r * b_s + b_r = 400 * 2500 + 400 = 1,000,400$ block transfers

Seeks

$2 * b_r = 2 * 400 = 800$ seeks

4b

Block Transfers

$b_s * b_r + b_s = 2500 * 400 + 2500 = 1,002,500$ block transfers

Seeks

$2 * b_s = 2 * 2500 = 5000$ seeks

Exercise 5

I think that the **Hash Join** will be better because it is fast and efficient for equi-joins, but this would not be as good for memory usage.

If it would preserve memory usage it would choose a **sort-merge join** which would be more memory efficient but it would not be as fast as a hash join.