2Q1. A) spanned: a bit complicated to implement in comparison, but doesn’t waste space, can support big tuples, good locality. Has slower random access

Unspanned: simple, easy to implement, good locality, however wastes space and doesn’t support big tuples. Has faster random access.

b)

1st: 9, 7

2nd: 8, 4

3rd: 16, 10

4th: 1, 3

Because we just consume input rows and put them into our hash table, but we only have 4 groups so assuming we have at least 4 buckets there’s no conflict.

Really unsure about the ordering because doesn’t it depend on the hash used? And what values of a we choose to start t entering the hash table?

c)

i) single\_tuple\_size = 8 bytes

Num\_of\_pages = single\_tuple\_size \* num\_of\_tuples / pagesize = 8 \* 4000000 / 64 = 500000 pages

Total\_IO = scan\_IO = num\_of\_pages = 500000 pages

ii)

buffer\_size\_in\_pages = 2^20 / 64 = 16384 pages

Selection\_input\_in\_pages = table\_size\_in\_pages = 4 (reference size) \* 4000000 / 64 = 250000 pages

Selection\_selectivity = 1 / 32\

Selection\_output\_in\_pages = Ceiling[selection\_input\_in\_pages \* selection\_selectivity] = 7813 pages

Selection\_IO = selection\_input\_in\_pages + selection\_output\_in\_pages = 257813 pages

Toal\_IO = selection\_IO = 257813 pages

ALTERNATIVE **(Correct me if I am wrong)**

buffer\_size\_in\_pages = 2^20 / 64 = 16384 pages

Selection\_input\_in\_pages = table\_size\_in\_pages = 4 (reference size) \* 4000000 / 64 = 250000 pages

Selection\_selectivity = 1 / 32

Selection\_output\_in\_pages = Ceiling[selection\_input\_in\_pages \* selection\_selectivity] = 7813 pages < bufferpoolSize

Selection\_IO = selection\_input\_in\_pages = 250000 pages

TuplePerPage = 64/4 =16

Project\_input\_in\_pages = ceiling (p(selectivity, TuplePerPage) \* table\_size\_in\_pages) = ceiling(1-(1-1/32)^16 \* 250000) = 99573

Project\_output\_in\_pages = ceiling(selectivity \* table\_size\_in\_pages) = ceiling(1/32 \* 250000) = 7813 Pages < BufferPool

Projection\_IO = Project\_input\_in\_pages = 99573

Total\_IO = Selection\_IO + Projection\_IO = 250000 + 99573 = 349573 Pages

iii)

Single\_bin\_size = num\_of\_tuples \* 1bit / 8 = 50000 bytes

Selection\_selectiveity = ¼

Selection\_input\_in\_pages = 4 (reference size) \* 4000000 / 8 (8 bins) / 64 = 31250 pages

Selection\_output\_in\_pages = 7813 pages

7813 < BufferPool (so don’t include this below?)

Total\_IO = single\_bin\_size + selection\_input\_in\_pages + selection\_output\_in\_pages = 89063 pages

Also needs to consider projection I/O

Projection IO same as before: 99573

d. B-Tree is row by row while decomposed is column by column?

I think “unnecessary” refers to the extra storage required to maintain the indices, for B-Trees this is 1 page per node or something, whereas decomposed you can store a lot more column values on a single page?

2.

a.

Pipeline-break: an operator which produces output only after all tuples from one participant have been consumed, examples: difference and product

Pipelinable: operator which has exactly one participant, examples: selection and projection

Implication for query: Pipeline breakers add opportunity for more page I/O if the buffers don’t fit into memory, so lessens the advantages of little I/O being associated with Volcano processing

b.

RLE with Exclusive Prefix Sum

(2, 0) (1, 1) (2, 2) (1 , 3) (2, 6) (1, 7) (2, 9) (1, 12) (2, 13) (1, 14)

The calculation method I guess is just showing the cumulative summing process.