Ryan Ballenger
CSCIE66, PS2 Part 1

1.

a) | 2980516 | 1519666 | BEST-ACTOR#---------- | 2015 |

b) 7 + 7 + 23 + 2 = 39 bytes

c) | 7 | 2980516 | 7 | 1519666 | 10 | BEST-ACTOR | 2 | 2015 |

d) 2 + 7 + 2 + 7 + 2 + 10 + 2 + 2 = 34 bytes

e)

Record: | 10 | 17 | 24 | 34 | 36 | 2980516 | 1519666 | BEST-ACTOR | 2015 |

Offsets: 0 2 4 6 8 10 17 24 34 36

f) 2 * 5 + 7 + 7 + 10 + 2 = 36 bytes

g)

An advantage of the fixed length record is that fields in the table can be updated and other fields do not have to move, even if the changing field is variable length. Another advantage is that in cases where all the fields are fixed width, it uses the least amount of storage or memory. This is because no offsets need to be stored and each field fits the space it is allocated. The disadvantage of fixed length records are that they waste storage space when VARCHARs are shorter than the maximum width. If a VARCHAR(25) field is storing a length 2 string, 23 extra bytes will be allocated but will be unused for this field.

An advantage of the variable length record, with each field preceded by its length, is that all other fields can be shifted when a field is updated to a new length. That does not affect the length of the other fields and their offsets are still correct after they get shifted. Another advantage is that the minimum amount of space is allocated for VARCHARs. Unlike the fixed width, this format takes VARCHAR lengths into account and allocates the minimum and precise amount of space needed for them. A disadvantage is that extracting a column value requires hopping through all the previous fields. Each offset must be read as the iterator travels through the tuple to reach the targeted field, which can be costly especially if there are many fields.

An advantage of the variable length record, that begins with a header of offsets, is that storage space is saved by allocating the minimum amount for VARCHARs. Another advantage is that when a tuple is accessed to retrieve one column value, that specific column can be quickly accessed by its offset in the header. A disadvantage is that updating is expensive. First of all, all the fields must be shifted if a VARCHAR changes length in an update. Secondly all the offsets for the updated field and fields afterwards must be updated to the new offset length.

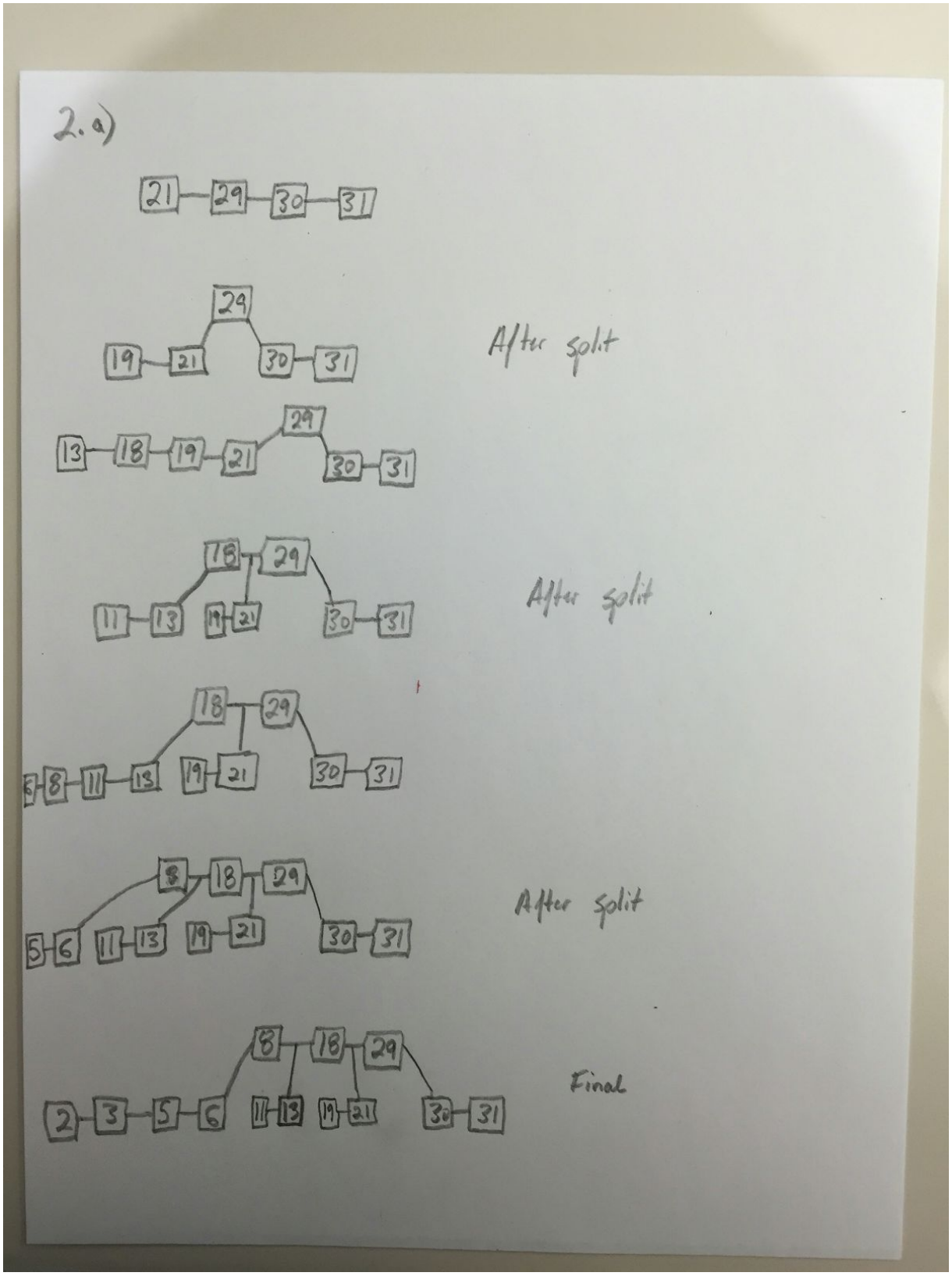
h)

For a read heavy workload with mostly select commands, the best choice is a variable length record with a header of offsets. This would save space by allocating the exact length needed for the VARCHAR compared to the over-allocation of the fixed width record. Also, there are not many updates which are expensive for the variable length record, that has to shift values and recalculate offsets. With mostly reads, the variable length record should be used to save storage space, and I chose the header of offsets because it enables fast access to specific columns instead of requiring the hop through previous fields.

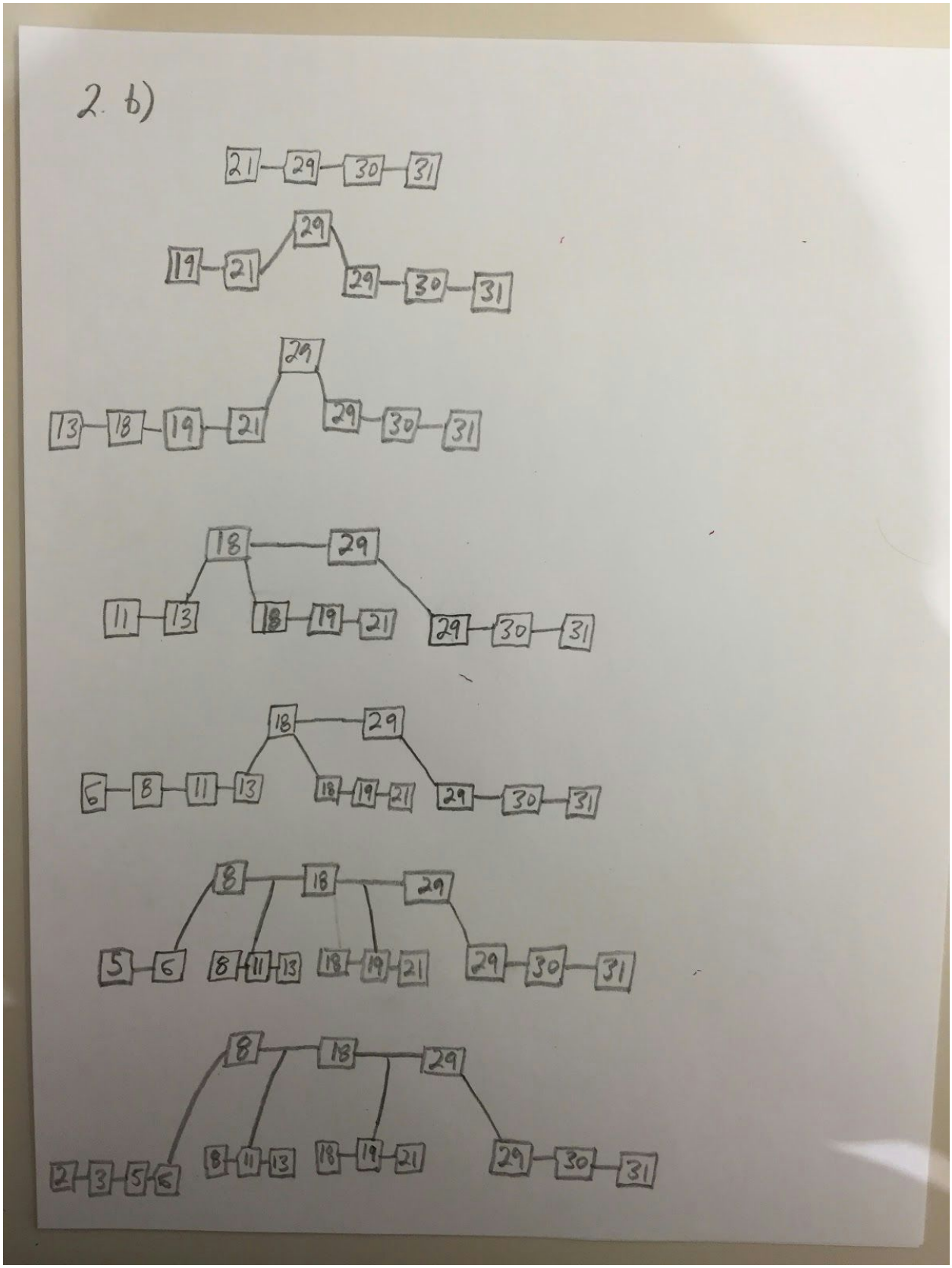
For a write heavy workload with mostly update commands, the fixed length record is the best choice. The update process is inexpensive for the fixed width record, as the old value can simply be overwritten by the new value. Since the maximum width is allocated for VARCHARs, other fields never need to be shifted and there are not offsets to adjust.

For a scenario in between, such as a 50/50 update and select workload, the fixed length record is probably the best choice. With the fixed length record, the updates are processed quickly since other fields do not need adjusted, as described earlier. Also the fixed width record only slightly over allocates storage space in this scenario. In fact, it saves 8 or 10 bytes that are required for offsets in the variable length options and only allocates unused storage to the VARCHAR by approximately 13 bytes or less. For longer named awards like BEST-DIRECTOR, it uses the same amount of storage as the variable-length options. For its quick updates and only slight over-allocation of storage space for this table, the fixed width record is the best choice for a split workload.

2. a)


2. a)
21 29 30 31
29
19 21 30 31
After split
29
13 18 19 21 30 31
18 29
11 13 19 21 30 31
After split
8 18 29
5 6 11 13 19 21 30 31
After split
8 18 29
2 3 5 6 11 13 19 21 30 31
Final

2. b)


2. b)
21 29 30 31
19 21 29 29 30 31
13 18 19 21 29 30 31
11 13 18 19 21 29 30 31
6 8 11 13 18 19 21 29 30 31
5 6 8 11 13 18 19 21 29 30 31
2 3 5 6 8 11 13 18 19 21 29 30 31

2. c)

Bucket	**Hashed Values**	**Extra notes**
0	30	
1	31, 29, 21	Before (4 count)
0		
1	31, 29, 21, 19	
10	30	After (5 count)
0		
1	31, 29, 21, 19	
10	30, 18	Before (6 count)
0		
1	29, 21, 13	
10	30, 18	
11	31, 19	After (7 count)
0		
1	29, 21, 13	
10	30, 18	
11	31, 19, 11	Before (8 count)
00	8	
01	29, 21, 13	
10	30, 18	
11	31, 19, 11	After (9 count)
100		
00	8	
01	29, 21, 13	
10	30, 18, 6	
11	31, 19, 11	
100		Before (10 count)
00	8	
01		
10	30, 18, 6	
11	31, 19, 11	
100		
101	29, 21, 13, 5	After (11 count)

00	8	
01		
10	30, 18, 6	
11	31, 19, 11, 3	
100		
101	29, 21, 13, 5	Before (12 count)

00	8	
01		
10	18, 2	
11	31, 19, 11, 3	
100		
101	29, 21, 13, 5	
110	30, 6	After (13 count)