2.1

1. Smallest tuples size

According to the provided schemea, the size can be taken into 5 parts:

- Record header: 4B
- name VARCHAR: 0B
- student BOOLEAN: 1B
- birthday DATE: 8B
- state VARCHAR: 0B

The smallest tuple size is 4B + 0B + 1B + 8B + 0B = **13B**

2. Maximum number of records

The reason to conduct word-alignment is to make sure data can be accessed in more efficient way. If the data stored in a word-aligned way, CPU can read it faster.

To conduct word-alignment with 8B long, the smallest size of a tuple and its slot info is:

- Header + name + student: 8B
  - Header: 4B
  - name: 0B
  - student: 1B
- birthday: 8B
- state: 0B
- record pointer: 8B
- record length: 8B

Total smallest size: 8B * 4 = 32B

Page header:

- slot count: 8B
- free space pointer: 8B

Toal page header size: 8B + 8B = 16B

maximum number = floor((2KB - 16B) / 32B) = floor(2032B / 32B) = **63 records**

3. Maximum number of records with reordering

To conduct word-alignment with 8B long, the smallest size of a tuple and its slot info is:

- Header + name + student: 8B
  - Header: 4B
  - name: 0B
  - student: 1B
  - state: 0B
- birthday: 8B
- record pointer: 8B
- record length: 8B

Total smallest size: 8B * 4 = 32B

Page header:

- slot count: 8B
- free space pointer: 8B

Toal page header size: 8B + 8B = 16B

maximum number = floor((2KB - 16B) / 32B) = floor(2032B / 32B) = **63 records**

2.2

1(a). Pages read using DSM

Since the Class has "10000 tuples that fit into 500 pages", and it has 5 columns with same fix-width, the size of a single column will take 500 / 5 = 100 pages.

First, to find the values with `size` > 50, we need to scan all values, in the worst case, this will takes 100 pages.

Second, to find the `MAX(credits)`, we need to scan all values that with `size` > 50, in the worst case, this will takes another 100 pages.

In total, it will take 100 + 100 = **200** pages.

1(b). Pages read using NSM

Since we are using NSM, which means the data is row store, to find the values with `size` > 50 and the `MAX(credits)`, we need to scan all values, in the worst case, this will takes **500** pages.

2(a). Min & Max pages read using DSM

For the minimum pages, we can assume the two id are at the beginning of the table. Because we need to execute queries based on `id` and get the 3 columns, we need to read **4 pages**.

For the maximum pages, we can assume the two id are at the end of the table. Because we need to execute queries based on `id` and get the 3 columns, for the `id` we need to read 100 pages, for other 3 columns, they might appear in different pages, so we need to read 6 pages. In total, we need to read **106 pages**.

2(b). Min & Max pages read using NSM

For the minimum pages, we can assume the two id are at the beginning of the table. Because all required columns are in the same row, we need to read **1 page**.

For the maximum pages, we can assume the two id are at the end of the table. Because all required columns are in the same row, we need to read **500** pages.

2.3

1. Instering order

h1(16) = (0b10000 >> 2) & 0b11 = 0b00100 & 0b11 = 0b00

h2(16) = 0b10000 & 0b11 = 0b00100 & 0b11 = 0b00

h1(3) = (0b11 >> 2) & 0b11 = 0b0 & 0b11 = 0b00

h2(3) = 0b11 & 0b11 = 0b11

If 16 is inserted first, it will be inserted into bucket 0, location 00. Then 3 will be inserted into bucket 0, location 00. Since the location already have 16, 16 will be rehashed to bucker 1, location 00, which is not the same as the figure shows.

If 3 is inserted first, it will be inserted into bucket 0, location 00. Then 16 will be inserted into bucket 0, location 00. Since the location already have 3, 3 will be rehashed to bucker 1, location 11, which is the same as the figure shows.

So, **3 is inserted first**.

2. Insert and delete

h1(8) = (0b1000 >> 2) & 0b11 = 0b10 & 0b11 = 0b10

After inserting 8 and deleting 16, the hash table will be:

	Table1	Table2
00		
01		
10	8	
11		3

3. Insert

h1(19) = (0b10011 >> 2) & 0b11 = 0b100 & 0b11 = 0b00

h1(4) = (0b100 >> 2) & 0b11 = 0b01 & 0b11 = 0b01

After inserting 19 and 4, the hash table will be:

	Table1	Table2
00	19	
01	4	
10	8	
11		3

4. Causing infinite loop

h1(19) = (0b10011 >> 2) & 0b11 = 0b100 & 0b11 = 0b00

h2(19) = 0b10011 & 0b11 = 0b11

h1(3) = (0b11 >> 2) & 0b11 = 0b0 & 0b11 = 0b00

h2(3) = 0b11 & 0b11 = 0b11

We can see that the hash value for 19 and 3 are the same, to cause infinite loop, we only need to find a number that having the same hash value with 19 and 3.

Here, 0b100011, i.e., 35 meet the need:

h1(35) = (0b100011 >> 2) & 0b11 = 0b100 & 0b11 = 0b00

h2(35) = 0b100011 & 0b11 = 0b11

When inserting 35, it will evict 19, and 19 will be rehased to table 2 location 11, where is the location for 3, and 3 will be evicted to table 1 location 00, where 19 is. This will cause an infinite loop.