a) I tried to understand this through visualization of the array. We have a 64B cache, and we are assuming a size of n = 32. This means our ColorPoint structures are 32B in size, and we can store two of these structs in our 64B cache. Let's assume a 4 x 4 2D array as:

```
points = [(c0, c1, c2, c3),
          (c4, c5, c6, c7),
          (c8, c9, c10, c11),
          (c12, c13, c14, c15) ]
```

Where c0 - c15 are the color point instances. We are calling elements as **points[j][i].** At the start we have an empty cache. To help me with this, I visualized how the code will interact with the array:

-Empty Cache-
i = 0
j = 0
points[0][0].a = c0.a

Right now our cache is empty, so we get a cold miss for c0.a. This causes the whole c0 structure to be loaded into the cache, since we can't just load individual parts of the structure into the cache. I believe c1 will also be loaded into the second cache slot as well at this point since it is the lower-level neighbor to c0. This means that

points[0][0].rgb = c0.rgb

result in all hits, because all the rgb elements are loaded when we cold miss on c0.a. So on this iteration, we have 1 miss and 3 hits.

Now, we increment our value for j. What we notice from this is that since we are calling elements as points[j][i] we end up jumping to a whole new row for j = 1.

-Cache with previous ColorPoint instance-
i = 0
j = 1
points[1][0].a = c4.a

Now what we should notice is that for this iteration of j we have jumped to a whole new row in our matrix. This creates a conflict miss, because c0 and c4 both exist in position 0 of their respective lists, and as such want to be in the same place in the cache. Also, our cache is already full as it stores both c0 and c1, so c4 cannot go into the second cache slot. This causes the whole cache to reload, now with c4 and its lower level neighbor c5 stored. So, once again, we have a miss and then 3 hits.

On our next iteration of j, the same thing will happen. We will be looking at c8, which again is in position 0 of its respective will want to go where c4 currently is. This causes another reload, which means another miss followed by 3 hits. This trend should continue until the end of the loop, with each new row jump encompassing a miss followed by 3 hits. This means for every call of points[j][i].a we get a miss, and then 3 hits. We have 16 stored ColorPoint objects in this example, which means we have 16 total misses and 48 total hits, so we have

Miss rate = misses/(hits + misses) = 16/(48 + 16) = 1/4 = 25% miss rate

b) Once again I am going to use visualization to understand this:

```
points = [(c0, c1, c2, c3),
          (c4, c5, c6, c7),
          (c8, c9, c10, c11),
          (c12, c13, c14, c15) ]
```

This time around we are looking up elements as **points[i][j]** rather than **points[j][i]**. Once again, we start with an empty cache.

-Empty Cache-
i = 0
j = 0
points[0][0].a = c0.a

Like before, we get a cold miss here since the cache is empty. This results in both c0 and its neighbor c1 being loaded into the cache to fill it up. This means that the rgb component of c0 is all hits. Now, j increments.

-Cache stores c0 and c1-
i = 0
j = 1
points[0][1].a = c1.a

Now we have something exciting. In the previous stage, both c0 and c1 are loaded into the cache, so now when we go to access c1 elements they are all in the cache, so we get 4 hits for c1.argb. So far, then, we have 1 miss and 7 hits.

Now j increments again, so we are looking at

-Cache stores c0 and c1-
i = 0
j = 2
Points[0][2].a = c2.a

Now, since our cache is full, we get a conflict miss trying to access c2 elements from the cache. This results in another reload for the cache, which is updated to store c2 and it's neighbor c3. This means we get a hit and 3 misses for c2, and then, since c3 is already in the cache, 4 hits for c3. This process will repeat again when we move to the next row, 1 miss and 3 hits followed by 4 hits followed by 1 miss and 3 hits followed by 4 hits. That means that for each row we have 2 misses and 14 hits. With 4 rows, this amounts to 8 misses and 56 hits.

Miss rate = misses/(hits + misses) = 8/64 = 1/8 = 12.5% chance of a miss.