Name: Baihan Lin
UWID: sunnylin, 1360521
Section: BC

1. Describe your approach to implementing the hash table. (separate arrays for keys and values vs one array with objects that represent pairs.)

To implement my hash table, I decided to use one array with objects that represent pairs. I created a private inner class called **`HashPair`** inside ColorHash, and each HashPair stores a pair of ColorKey type key and a long type value.

Each HashPair holds two private fields ColorKey **`key`** as the key of the dictionary pair and long **`value`** as the value of the dictionary pair. Inside my class HashPair, there are several methods to facilitate my ColorHash class:

i. **`void setKey(ColorKey keyNew)`** is a public method to set the key of the hash pair to input key.

ii. **`void setVal(long valueNew)`** is a public method to set the value of the hash pair to input value.

iii. **`ColorKey getKey()`** is a public method to retrieve the key of the hash pair.

iv. **`long getVal()`** is a public method to retrieve the value of the hash pair.

v. **`boolean isKey(ColorKey testKey)`** is a public method to test whether the value matches the value of the ColorKey.

Thus, my ColorHash is implemented as an array of HashPair objects called hashTable.

2. If you implemented any methods other than the specs, describe them briefly here.

For ColorHash.class:

i. **`int twiceAndPrime(int tableSize)`** is a private helper method to support `resize`, taking `@param tableSize`: old table size that needs to be resized and `@return newTableSize`: the new table size which is at least twice as big the old table size and has to be a prime number.

ii. **`int[] doProbing(ColorKey key)`** is a private helper method to support `increment`, `colorHashPut`, `colorHashGet` and `getCount`, with purpose of retrieving the insert location of ColorKey while summing number of collisions based on linear or quadratic probing based on collision resolution method from constructor input. It takes `@param key`: the ColorKey to insert in the hash table and `@return` an integer array of size 2 storing the index of insertion location and number of collisions during probing.

iii. **`boolean ifRehash()`** is a private helper method to support `colorHashPut` and `increment`, with a purpose of determining whether rehash load factor is exceeded and rehash is necessary. It `@return` a boolean to indicate the necessity of rehashing. If necessary to rehash, it does rehashing.

iv. **`HashPair[] getHashTable()`** is a private helper method to support `resize`. It `@return` the raw HashPair[] format of hash table of the given ColorHash.

v. The methods specified inside HashPair class is described above in question 1.

For FeatureVector.class:

i. **`double dot(long[] A, long[] B)`** is a private method to help conduct `cosineSimilarity` and to calculate the dot product of two vectors. It takes `@param A` and `B` as vectors in format of long[], `@return prod`: the dot product of two vectors, and `@throws` Exception if length of two vectors are different.

ii. **`double vec(long[] A, long[] B)`** is a private method to help conduct `cosineSimilarity` and to calculate the vector product of two vectors. It takes `@param A` and `B` as vectors in format of long[], `@return prod`: the vector product of two vectors and `@throws` Exception if length of two vectors are different.

For ComparePaintings.class:

i. **`void extraCredit10ImagesTest()`** is a method to support extra credit. It chooses 10 different images of paintings from the web, creates a table with 10 rows and 10 columns that gives the results of running the cosine similarity analysis on each pair and present the table neatly in your report. It `@throws` Exception if ComparePaintings not met.

ii. **`long countBlack(String filename)`** is a method to count the number of black pixels in Mona Lisa. It takes `@param` filename: Mona Lisa in this case, `@return black`: the number of black pixels in this image, Mona Lisa in this case, and `@throws` Exception if condition not met.

3. When you use 6 bits per pixel, how many black pixels are there in the Mona Lisa image? (These are the pixels whose ColorKey bits value equals 0.)

In my main method in `ComparePaintings.java`, I called my `countBlack` method:

```
// countBlack
long black = cp.countBlack(IMG_01);
System.out.println("In Mona Lisa there is "+black+" black pixels ");
```

It outputs:

```
In Mona Lisa there is 62007 black pixels.
```

4. Provide a copy of the table of counts produced by your collisionTests method.

Bits Per Pixel	C(Mona,linear)	C(Mona,quadratic)	C(Starry,linear)	C(Starry,quadratic)	C(Christina,linear)	C(Christina,quadratic)
24	69148	51960	900255	283999	39010	30650
21	225331	70320	3578431	705893	67857	26791
18	177112	59422	73013	24469	12189	6814
15	193223	70659	204184	49508	450185	163897
12	19076	17312	71476	24391	1072	936
9	99	78	149	122	1012	701
6	0	0	0	0	2811	2812
3	0	0	0	0	0	0

5. Provide a copy of the table of similarity values by your fullSimilarityTests method.

Bits Per Pixel	S(Mona,Starry)	S(Mona,Christina)	S(Starry,Christina)
24	0.032761	0.017471	0.013991
21	0.039927	0.020397	0.016081
18	0.052170	0.025882	0.019386
15	0.080360	0.041097	0.024926
12	0.184209	0.114446	0.038655
9	0.416916	0.362219	0.080113
6	0.652556	0.397594	0.269563
3	0.835274	0.963935	0.859103

6. Examine the hashCode method of class ColorKey. What types of images might tend to cause lots of collisions relative to other images?

Algorithmically speaking, the images with more similar colors or more similar grey levels tend to cause lots of collisions, because these similar colors may be encoded as the same index which requires further probing to find the next available spots.

Empirically speaking, shown in the collisions table in question 4 in most bitsPerPixels scales, “Starry Night” causes more collisions than “Mona Lisa” and “Christina’s World”. Visually screening of the three images would easily tell that “Starry Night” have more similar colors than the other two. The repeating usage of blue-ish and yellow-ish colors in “Starry Night” may cause more collisions due to their tendency to go into same index and conduct further probing.

7. Extra Credit: Cosine similarity of 10 different images

Table of cosine similarity based on output from Java Eclipse console:

	MonaLisa	StarryNight	ChristinasWorld	WaterLilies	ParisAutumn	ButterflyShip	Kiss	StJeromeReading	CityRises	Untitled
MonaLisa	1.000000	0.652556	0.397594	0.327435	0.527633	0.455484	0.303041	0.897027	0.348119	0.230280
StarryNight	0.652556	1.000000	0.269563	0.659752	0.448848	0.496905	0.103536	0.637050	0.219680	0.261673
ChristinasWorld	0.397594	0.269563	1.000000	0.501001	0.445505	0.259131	0.178561	0.044816	0.247219	0.697557
WaterLilies	0.327435	0.659752	0.501001	1.000000	0.453714	0.339071	0.219641	0.145600	0.281676	0.535252
ParisAutumn	0.527633	0.448848	0.445505	0.453714	1.000000	0.639519	0.376285	0.425957	0.570880	0.318682
ButterflyShip	0.455484	0.496905	0.259131	0.339071	0.639519	1.000000	0.182543	0.429432	0.244072	0.151475
Kiss	0.303041	0.103536	0.178561	0.219641	0.376285	0.182543	1.000000	0.180826	0.526363	0.042205
StJeromeReading	0.897027	0.637050	0.044816	0.145600	0.425957	0.429432	0.180826	1.000000	0.254416	0.001983
CityRises	0.348119	0.219680	0.247219	0.281676	0.570880	0.244072	0.526363	0.254416	1.000000	0.127019
Untitled	0.230280	0.261673	0.697557	0.535252	0.318682	0.151475	0.042205	0.001983	0.127019	1.000000

As we can see from the table, C(MonaLisa, StJeromeReading) = 0.897027 is the most similar image pair with the maximum cosine similarity. Visually speaking, “Mona Lisa” and “St. Jerome Reading” indeed look similar.

Image References attached in Appendix behind:

Appendix: Image Summary

1. “Mona Lisa” by Leonardo Da Vinci
   https://courses.cs.washington.edu/courses/cse373/16au/A/A3/
2. “Starry Night” by Vincent Van Gogh
   https://courses.cs.washington.edu/courses/cse373/16au/A/A3/
3. “Christina’s World” by Andrew Wyeth
   https://courses.cs.washington.edu/courses/cse373/16au/A/A3/
4. “Water Lilies” by Claude Monet
   http://www.jenniferlynking.com/wp-content/uploads/2013/09/waterlilies_0410_-367.jpg
5. “Paris Autumn” by Dmitry Spiros
   https://img1.etsystatic.com/074/0/7777713/il_570xN.823315899_kot1.jpg
6. “Butterfly Ship” by Salvador Dali
   http://www.wallpaperawesome.com/wallpapers-awesome/wallpapers-famous-painting-artist-painter-brush-oil-on-canvas-awesome/wallpaper-salvador-dali-1.jpg
7. “Kiss” by Gustav Klimt
   https://uploads8.wikiart.org/images/gustav-klimt/the-kiss-1908(1).jpg
8. “St. Jerome Reading” by Georges de La Tour
   http://www.abcgallery.com/L/latour/latour66.JPG
9. “City Rises” by Boccioni
   http://exhibitions.guggenheim.org/futurism/content/small/futurism_heroic_boccioni_the_city_rises.jpg
10. “Untitled” by Cy Twombly
    https://news.artnet.com/app/news-upload/2015/02/2015-feb-11-cy-twombly-untitled-new-york-city-1970-christies-evening-auction.jpg