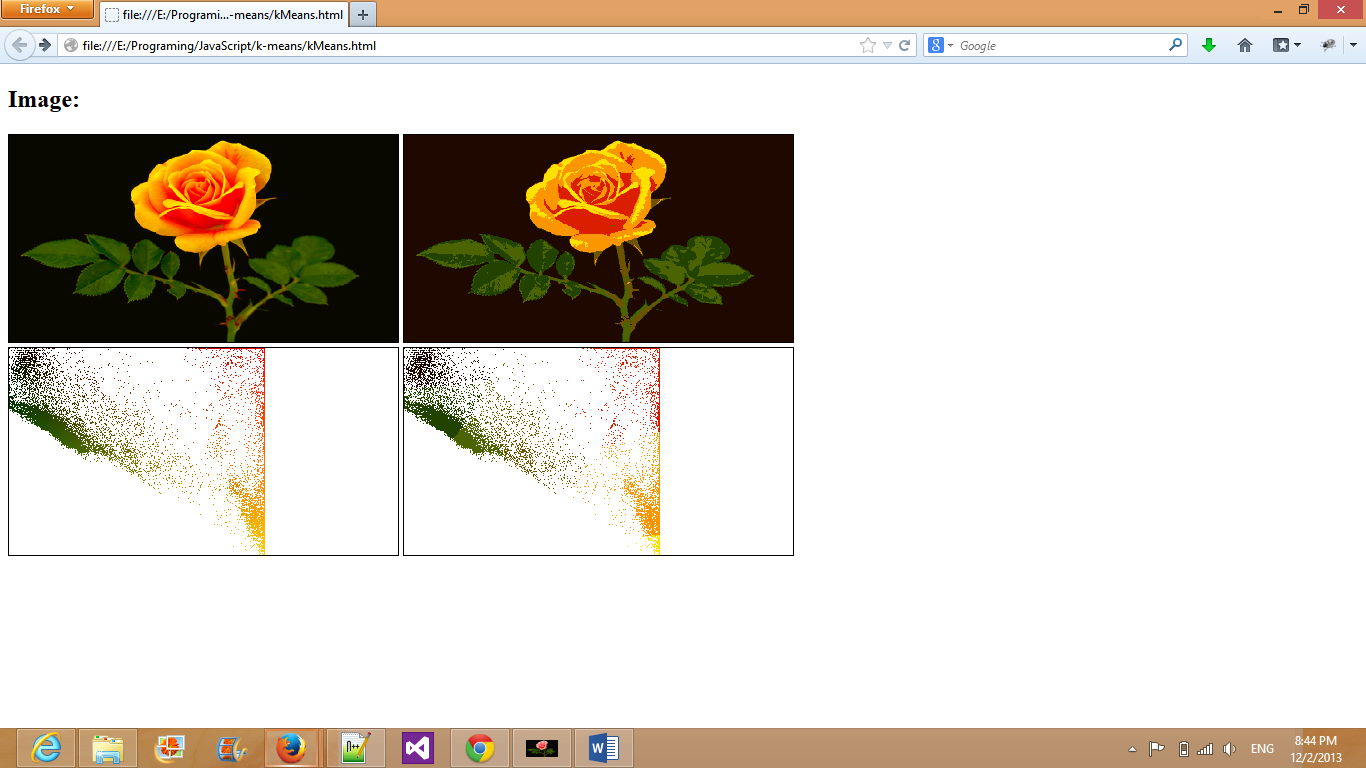
***K-Means***

**K-means Algorithm:**

The k-means algorithm is implemented in JavaScript and uses the HTML5 Canvas tag, I chose this due to its simplicity and easiness to share since it’s ran on a web-browser. The file is named “kMeans.html”, with some small HTML for Canvas and the rest JavaScript. The image is brought in and organized into an array of pixels (An object I created to hold RGB values). The implementation only uses the Red and Green values and makes the Blue value 0. The centroids are picked randomly at the start from a modified array of pixels (in which repeated values are removed); so the same RGB values are not picked. Then for every pixel on the screen the distance from that pixel to each k-point is computed and the minimum point is places into the cluster(2D array), the cluster is chosen from which index of the distance-points is the smallest. Then the clusters are re-calculated for new centroid based on the average (R,G) points (*for loop* for each cluster). Which is considered in a *while loop* until the centroids are the same as the previous *while loop*. In order to prevent a poor worst-case O this is restricted to 15 (or specified number) iterations (the standard while loop is done it probabilistic means). Then this data is show both as a main image and the clusters (the clusters color is specified by the color of the centroid), and a graph plot of (R,G) values and the grouped clusters. To show the clusters as a group in visual data. This was also purposely implemented to take the number of the same value pixels into account to show variability in the images computed, while probabilistic k-means did not to provide more accuracy.

*Complexity* : O(2(P\*k) + C\*avg\_c) [P = # of pixels, C = # of clusters, avg\_c = average cluster size]

Here is an example output from a normal k-means algorithm, where k = 7. The diagram below the images shows the (Red,Green) values of the image as a 2D plot to show visualization of the data. The graph in the lower right-hand corner shows the clusters group according to their centroids color.



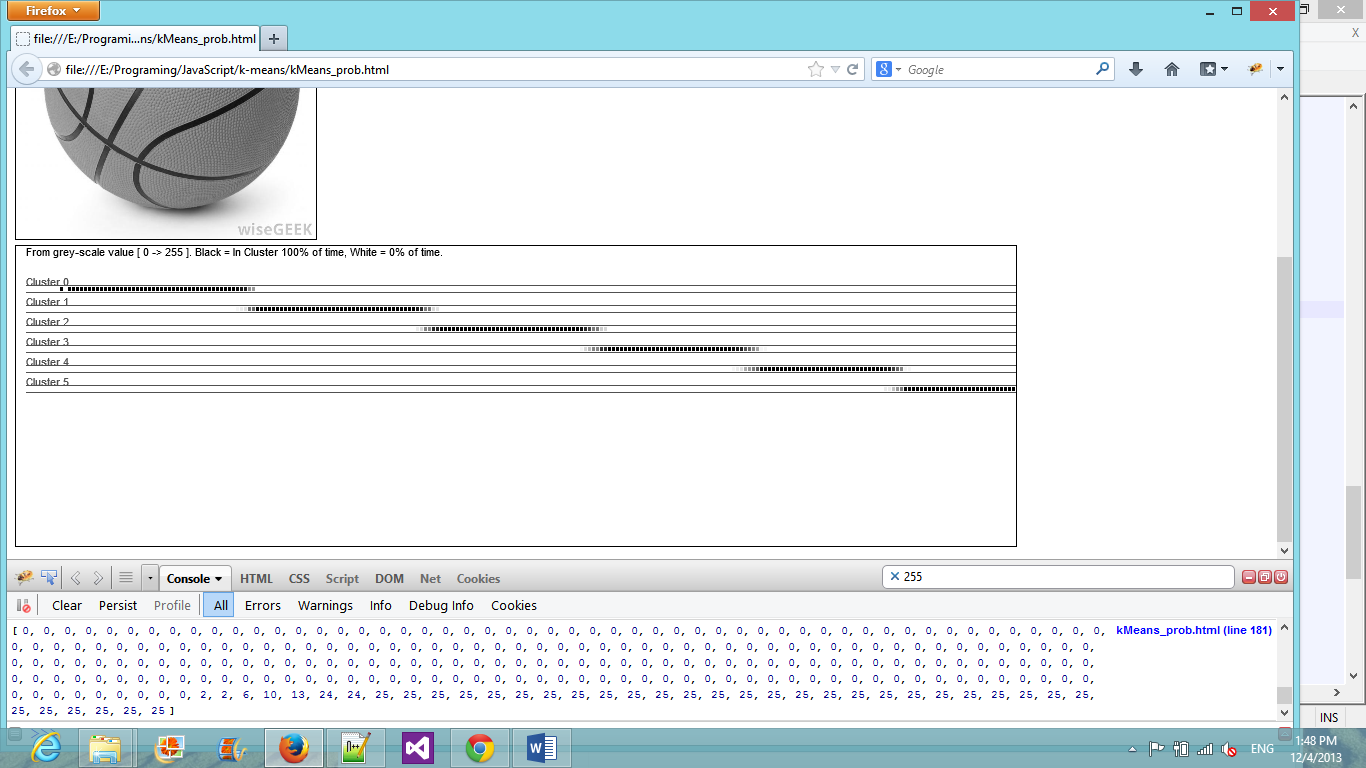
Runtime is approx. 5 seconds on a standard computer for entire run. (Including computation of visuals)

**Probabilistic K-means:**

Probabilistic k-means is also implemented in JavaScript but in a much simpler fashion to save time for multiple computation of k-means (in which is also more accurate since we are removing duplicate values from the data points). Also I have shown details on the entire computations are shown in the console. I am only using grey-scale images and am not outputting an image; but a visualization of probabilities of each points based by intensity as a “value line”, I believe this will be more clear for the data. I’m also using a simpler image, in this case a basketball. The computation will also be more in-place for clusters and not separated into separate data-structures as much as the original k-means. The code will run a basic k-means 25 times and the clusters will be separated and counted based on intensity values which is done my sorting the cluster array by intensity. The data points probability counter is a 2D array of K arrays each with 256 elements (initialized as all 0’s). This applies for all possible grey-scale values and array 0 is least intense while array K is most intense. Then finally we use the indexes on the probability counter matrix and increment the indexes based on which cluster the data was in, in this case it’s determined linearly since we sorted the clusters.

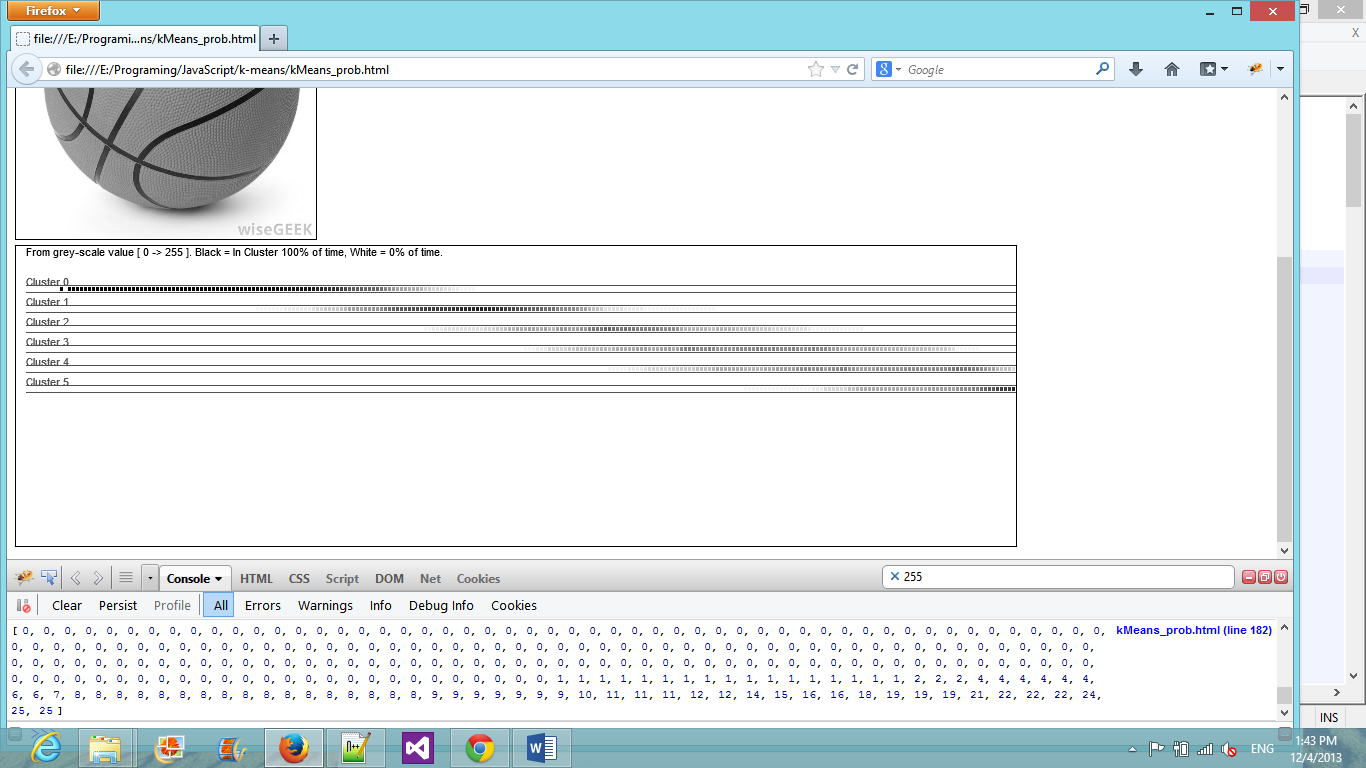
In this program I implemented k-means better and data structures sizes were more compact, each cluster only holds a maximum of around 256/K elements while in multi-colors it is 256^3 /K elements (not including pixels with the same colors). Otherwise as in the first k-means is was: P^3 /K (P= Number of pixels in image). Therefore my probabilistic k-means will be able to work better with pictures with a great number of pixels while the other program performance will drop.

For the computation of the probabilistic k-means, since the algorithm will repeat to save time we will convert to grey-scale value and compute the k-means on the liner data plot. Along with that K clusters are used and here’s the output of the probabilistic k-means for each cluster. White means the pixel is used 100% of cluster n and black being the pixel was 0% belonging to cluster n for 1…n clusters.



Runtime is approx. 4 seconds on a standard computer for entire run. (Including computation of visuals and console log). Notice that k-means is run 25 times and probability is computed.

To show the difference of the k-means avching convergence VS not. This is the program run while not adjusting the centroids. While we can see the general pattern the high probablity beforehand is dropped significantly.



Notes on running the program:

* Firefox is highly recommended to run both programs in. This will also run on Chrome and IE10+(although images are not hidden in IE).
* Link to K-Means : <http://cs.iupui.edu/~spdwiecz/JavaScript/k-means/kMeans.html>
* Link to Probabilistic K-Means: <http://cs.iupui.edu/~spdwiecz/JavaScript/k-means/kMeans_prob.html>