COSC 120: HW#4
Dr. Fred Park, Whittier College

1. **Image Contrast Equalization:**
   (a) Write code that uses a transform of the form $\alpha f + \beta$ where f is the 'pout.tif' image from my website that equalizes the contrast of the image. i.e. the min and max values of the image map to 0 and 255 respectively after the transform. Show visually using the plotting tools described in class that the image contrast is improved.
   (b) Use the automatic method from class to equalize the same image. Which is visually better?
   (c) Apply both methods to the factory and chapel images on my website. Display both results with the histograms next to them in the case of original and equalized images.
2. **The Birthday Problem:** Write code to find the minimum number of people needed in a room having the same birthday with probability greater than 50%. How many are needed to have a probability greater than 80%, 90%, 99%, 100%?
3. **Letter Count:** Using a dictionary, write code to find the letter counts in the word 'Mississippi'? Make sure your code generalizes to arbitrary words.
4. **Word Counts:** Copy and paste pages 15-30 from the book Dracula found here: `http://www.gutenberg.org/ebooks/345` using the EPUB (no images) version into a text document called 'dracula.txt'. Write python code that reads the text document and counts the unique words and stores the words and counts into a dictionary. Output the words and counts into a text file called 'dracula_words.txt'.
5. **Sorting Revisited:** Recall the bubble sort algorithm we implemented in class.
   (a) What was the complexity of this algorithm in the worst case scenario? Can you think of a way to improve the algorithm?
   (b) Implement the merge sort algorithm:
   - Given a list L, if the length is 0 or 1, it is already sorted.
   - If the list has more than one element, split the list into two lists, and then each split list into two more, and on and on until you reach the base case where you have lists of 2 elements each. Keep track of these. Sort the values of each of the 2 element lists and merge to obtain lists of 4 elements each which you then sort. Continue this process until the full list is sorted. For example, the merging process can be viewed on the example below:

     ```
     [14 33 27 10 35 18 42 44]
     [14 33 27 10] [35 18 42 44]
     [14 33] [27 10] [35 18] [42 44]
     [14 33] [10 27] [18 35] [42 44]
     [10 14 27 33] [18 35 42 44]
     [10 14 18 27 33 35 42 44]
     ```

     Note how the list is subdivided until lists of length 2 are found and then those are sorted and then merged and then sorted and the process continues until the final sorted list is obtained. Implement the merge sort recursively.
   (c) Time and test your merge sort versus the bubble sort on randomly generated lists of length 10, 100, 1000. Which one is faster and by how much? Use the time module to do this.