Rio Sonoyama
CS146, Homework 7

1.

(a) See
Since $f(x) = |x|$ is differentiable at $x \in (-\infty,0) \cup (0,\infty)$, subgradient of f at such x is simply the derivative of f at x. Namely,

$$\partial f(x) = \begin{cases} \{-1\} & \text{for } x < 0 \\ \{1\} & \text{for } x > 0 \end{cases}.$$

Now consider the case where $x = 0$. By definition of subgradient, a subgradient of f at $x = 0$ is $g \in \mathbb{R}$ such that

$$f(z) \geq f(0) + g(z - 0) = f(0) + gz \quad \forall z \in \text{dom} f.$$

That is,

$$\begin{aligned} |z| &\geq gz \\ z^2 &\geq g^2 z^2 \\ z^2(1 - g^2) &\geq 0. \end{aligned}$$

But since $z^2 \geq 0$, $1 - g^2 \geq 0 \implies g \in [-1,1]$.
Hence, subdifferential of f is given by:

$$\partial f(x) = \begin{cases} \{-1\} & \text{for } x < 0 \\ [-1,1] & \text{for } x = 0. \\ \{1\} & \text{for } x > 0 \end{cases}$$

(b) Note that subdifferential is additive. Thus, using the result from part (a), we have

$$\begin{aligned} \partial f(y_m) &= \sum_{j=1}^{N_k} \partial |y_j - y_m| \\ &= \{-1\} + \cdots + \{-1\} + [-1,1] + \{1\} + \cdots + \{1\} \\ &= \{-1\} + \{1\} + \cdots + \{-1\} + \{1\} + [-1,1] \\ &= \{0\} + \cdots + \{0\} + [-1,1] \\ &= [-1,1]. \end{aligned}$$

where y_m is the median of $\{y_1, \ldots, y_{N_k}\}$.
Thus, $0 \in \partial f(y_m)$, therefore, by the theorem given, y_m is a minimizer of f. Hence, the median of $\{y_1, \ldots, y_{N_k}\}$ minimizes f.

(c) Step1: For each x_n, assign k such that $k = \arg\min_j |x_n - x_j^*|$ where x_j^* is the median of cluster j, and set $r_{nk} = 1$ if cluster k is assigned to cluster x_n and $r_{nk} = 0$ otherwise.

Step2: For all $j \in \{1,2,\ldots,K\}$, update the median as follows: $x_j^* := \text{median}\{x_n : r_{nk} = 1\}$.

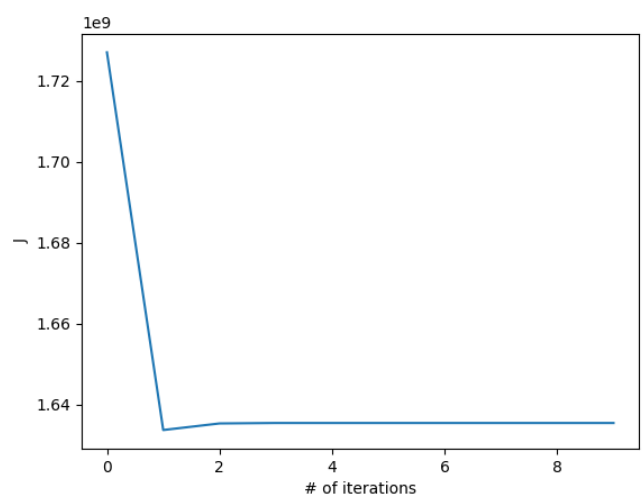
By using L1 norm instead of L2 norm, we can possibly save a small amount of run time because computing median is faster than computing mean.

3.

(a) The figure is shown below:

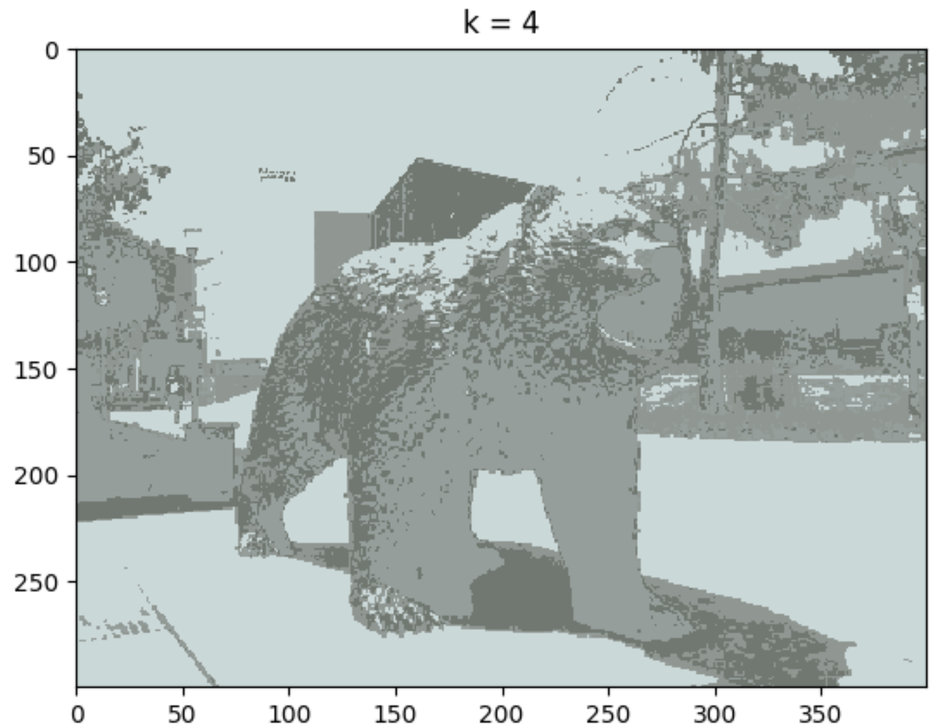
(b) Values of *J* v.s. # of iterations are shown below:

# of iterations	J [e+09]
1	1.72698702
2	1.63371479
3	1.63534069
4	1.63545065
5	1.63545065
6	1.63545065
7	1.63545065
8	1.63545065
9	1.63545065
10	1.63545065

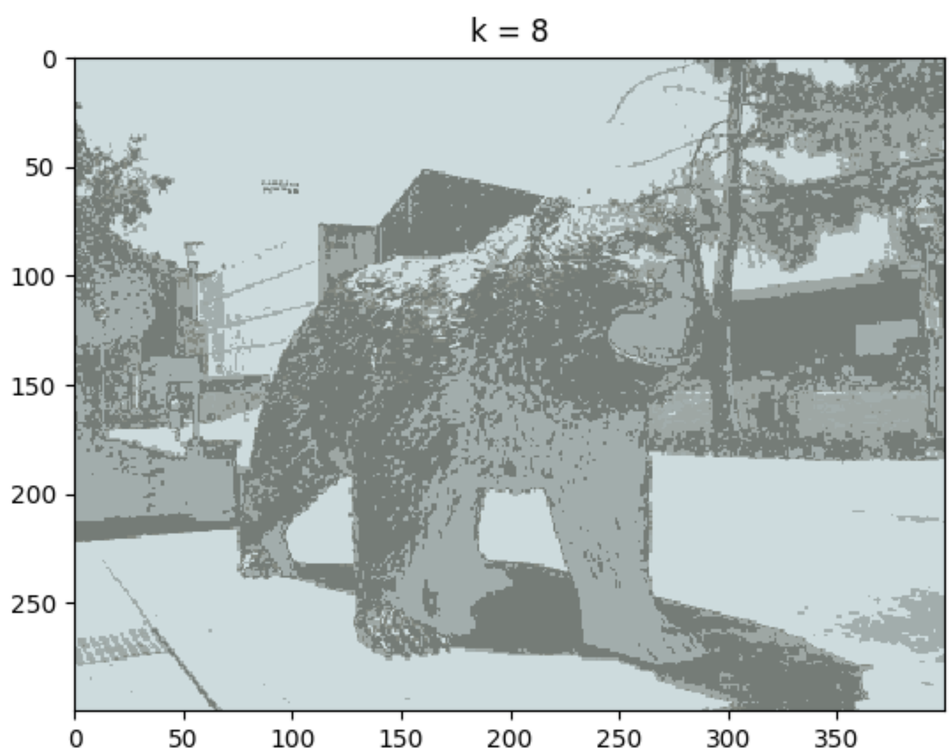


From the data above, we can see that K-means converges relatively fast.

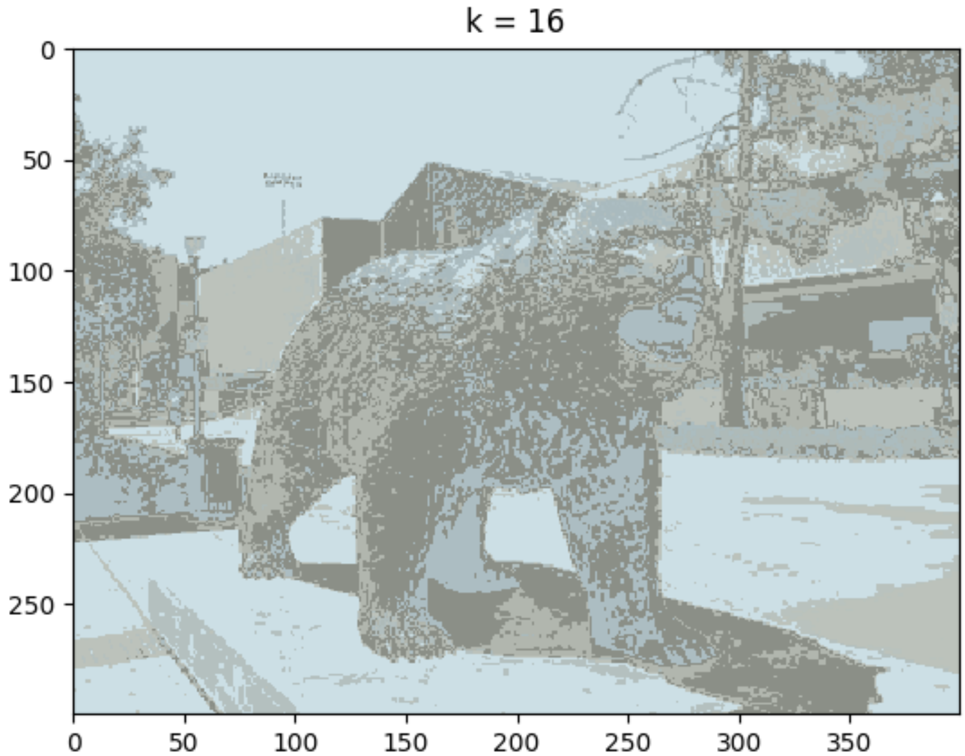
(c) For $k = 4$, value of the objective function after the last iteration is 1.63545065e+09.



For $k = 8$, value of the objective function after the last iteration is 1.65448955e+09.



For $k = 16$, value of the objective function after the last iteration is 2.14143802e+09.



Clearly, the larger k gives you clearer image. This makes sense because the larger k means more colors is used.

(d) Each pixel of the original image requires $3 \times 8 = 24$ bits. Since there are $300 \times 400 = 120000$ many pixels in the original image, the total amount of pixel needed is $24 \times 120000 = 2880000$ bits.
For $k = 4$, $4 \times 8 = 32$ bits are used to store mean colors. To store the index of cluster when there are 4 possible colors, we need 2 bits for each pixel. Thus, $32 + 2 \times 120000 = 240032$ bits are used in total. The compression ratios is $2880000/240032 \approx 11.998$.
For $k = 8$, $8 \times 8 = 64$ bits are used to store mean colors. To store the index of cluster when there are 8 possible colors, we need 3 bits for each pixel. Thus, $32 + 2 \times 120000 = 360064$ bits are used in total. The compression ratios is $2880000/360064 \approx 7.9986$.
For $k = 16$, $16 \times 8 = 128$ bits are used to store mean colors. To store the index of cluster when there are 16 possible colors, we need 4 bits for each pixel. Thus, $128 + 4 \times 120000 = 480128$ bits are used in total. The compression ratios is $2880000/480128 \approx 5.9984$.

Python script:

```
import numpy as np
import matplotlib.pyplot as plt


def read_png(filename):
    return plt.imread(filename)[:, :, :3] * 255


def min_dist(node, nodes):
    dist = np.sum((nodes - node) ** 2, axis=1) ** (0.5)
    return [np.argmin(dist), np.min(dist)]


def max_dist(node, nodes):
    dist = np.sum((nodes - node) ** 2, axis=1) ** (0.5)
    return [np.argmax(dist), np.max(dist)]


def remove(r, x):
    temp = []
    for i in x:
        if i[0] == r[0] and i[1] == r[1] and i[2] == r[2]:
            continue
        temp.append(i)
    return np.array(temp)


def initial_mu(x, m, k):
    mu = np.zeros((k, 3))
    mu[0] = np.array(m)
    x = remove(mu[0], x)
    mu[1] = x[max_dist(mu[0], x)[0]]
    x = remove(mu[1], x)

    for i in range(2, k):
        max_ = 0
        for p in x:
            if max_ == 0:
                c = mu[:i]
                max_ = min_dist(p, mu[:i])[1]
                val = p
                continue
            if min_dist(p, mu[:i])[1] > max_:
                max_ = min_dist(p, mu[:i])[1]
                val = p
        mu[i] = val
        remove(val, x)
    return mu


def k_means(k, iter, mu, x_val):
    j = np.zeros(iter)
    r = np.zeros((x_val.shape[0], k))

    for it in range(iter):

        # Assign cluster
        for i in range(x_val.shape[0]):
            c = min_dist(x_val[i], mu)[0]
            r[i][c] = 1

        # Update mean
```

```
        n = np.zeros(3)
        d = 0
        for i in range(k):
            for m in range(x_val.shape[0]):
                if r[m][i] == 1:
                    n += x_val[m]
                    d += 1
            mu[i] = n/d

        # Compute J
        for m in range(r.shape[0]):
            j[it] += np.sum((x_val[m] - mu[np.argmax(r[m])]) ** 2, axis=0)

    return j, r

def compress(x, r,mu,row,col):
    n = 0
    for i in range(row):
        for j in range(col):
            x[i,j] = mu[np.argmax(r[n])]
            n += 1

    return x


if __name__ == '__main__':
    m = [147, 200, 250]
    k = 16

    x = read_png('UCLA_Bruin.png')
    row = x.shape[0]
    col = x.shape[1]
    x_val = np.concatenate(x, axis=0)

    mu = initial_mu(x_val,m,k)

    j, r = k_means(k=k, iter=10, mu=mu, x_val=x_val)
    print (j)
    x_ = compress(x, r,mu,row,col)


    plt.imshow(x/255)
    plt.title('k = 4')
    plt.show()
    #plt.savefig('3_2_k16')

    plt.plot(j)
    # plt.show()
    plt.ylabel('J')
    plt.xlabel('# of iterations')
    plt.savefig('3_2')

    exit()
```