PSEUDO CODE FOR PROGRAM

```
method create_shingles(tuple)
    name, image <- tuple
    shingles <- empty set
    for each window of dimension (k,k,3) do
        flattened_window <- flatten(window) #convert it to a vector
        shingles.append(flattened_window)
    return shingles

method create_bit_vector(tuple)
    name, image <- tuple
    bit_vec <- bitarray(N)
    bit_vec.setall(0) #since it is sparse
    for each shingle in k_shingle_space do
        if shingle in image do
            set bit to 1 at that position
    return (name, bit_vec)
```

```
method create_signature(tuple)
    name, image <- tuple
    Initialize all M[i] = ∞
    for each bit in image do
        if bit is 1 do
            for each i in M do
                h <- ((a[i] * bit_position + b[i])mod p) mod N
                if h < M[i] do
                    M[i] <- h
    return (name, M)

method create_sketch(tuple)
    name, image <- tuple
    sketch_vector <- vector of all -1's
    for each random_hyper_plane in random_hyper_planes do
        h_v <- image.dot(random_hyper_plane)
        if h_v is non-negative do
            set 1 at position of sketch_vector
    return (name, sketch_vector)
```

method main()

images <- load images from database as numpy arrays

names <- load image names of corresponding images

rdd <- parallelize the images and names

#SHINGLING

rdd.map(create_shingles)

rdd.cache()

shingles <- rdd.flatMap(lambda r: r[1]) #get only the set of shingles

k_shingle_space <- shingles.distinct().collect()

N <- k_shingle_space.size

#MIN HASHING

signature_size = 100

a <- vector_of_random_integers

b <- vector_of_random_integers

p <- find prime number greater than N

min_hash_rdd <- rdd.map(create_bit_vector).map(create_signature)

#LOCALITY SENSISTIVE HASHING

bands <- 20

rows <- 5

min_hash_dict <- empty dictionary

signature_matrix <- min_hash_rdd.collect()

```
for each name, signature in signature_matrix do
    for each band in image do
        if band not in min_hash_dict do
            min_hash_dict[band] = list(name)
        else
            min_hash_dict[band].append(name)
jaccard_candidates <- empty dictionary
for each band in min_hash_dict do
    for each pair in min_hash_dict[band] do
        score <- calculate jaccard_similarity of pair
        if score > 0.8 do
            create group for all similar pairs and put it in jaccard_candidates
display jaccard_candidates using matplotlib

#RANDOM HYPERPLANES
sketch_size = 100
random_hyper_planes <- generate random_hyper_planes as vectors of 1's and -1's
rnd_hyp_rdd <- rdd.map(create_bit_vector).map(create_sketch)
```

```
#LOCALITY SENSISTIVE HASHING
sketch_matrix <- rnd_hyp_rdd.collect()
for each name, sketch in sketch _matrix do
        for each band in image do
                if band not in rnd_hyp_dict do
                        rnd_hyp_dict[band] = list(name)
                else
                        rnd_hyp_dict[band].append(name)
cosine_candidates <- empty dictionary
for each band in rnd_hyp_dict do
        for each pair in rnd_hyp_dict[band] do
                score <- calculate cosine_similarity of pair
                if score > 0.8 do
                        create group for all similar pairs and put it in
cosine_candidates
display cosine_candidates using matplotlib
```