Lossless Image Compression

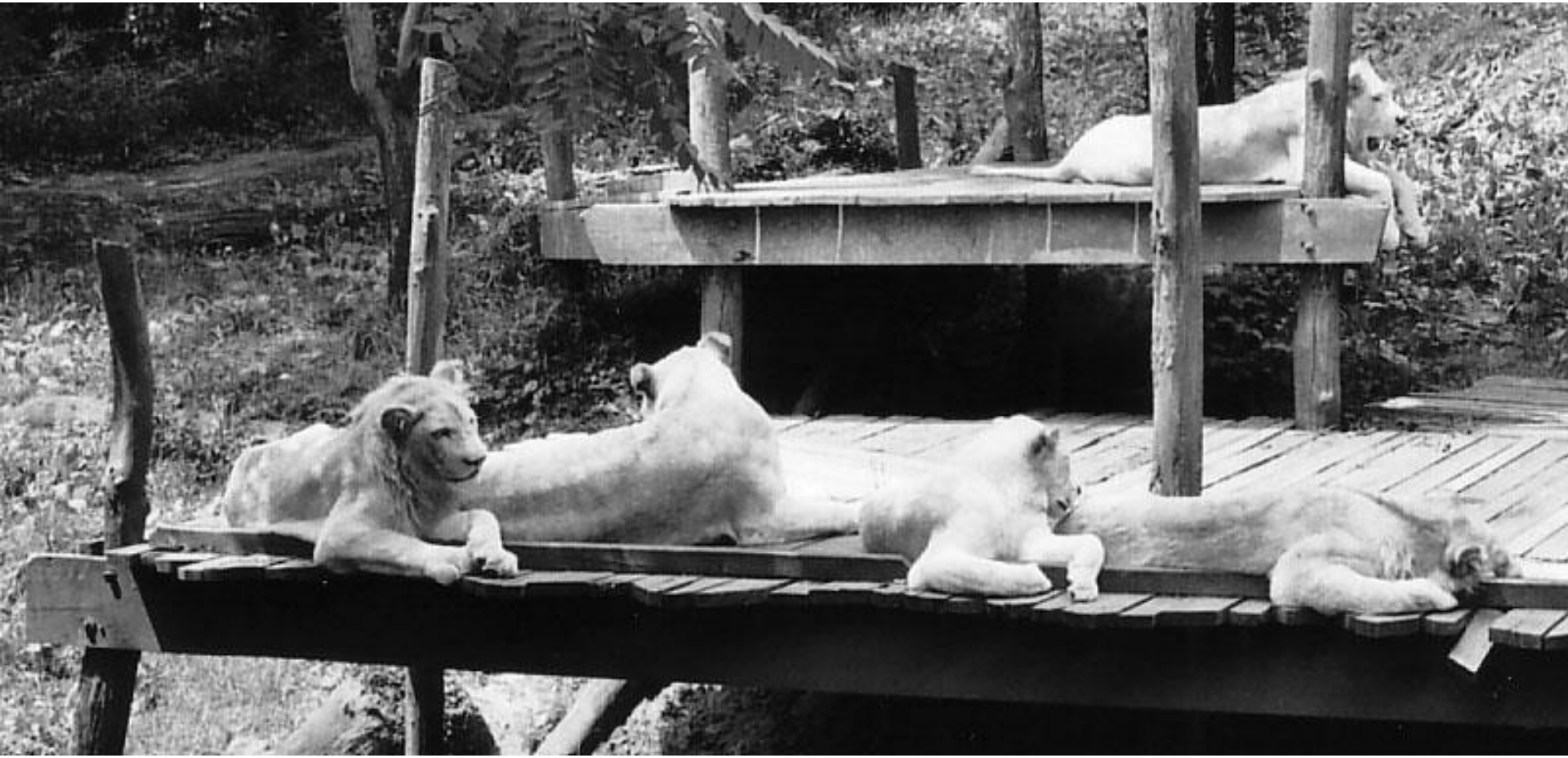
CS6025 Data Encoding

Yizong Cheng

1-22-15

Continuous-Tone Image

Grayscale Image of 2^8 Shades



Bi-leval Image

Raw Images

- Some digital cameras can produce raw images.
- Usually 24 bits (RGB) for each pixel.
- Easy to write a program to display the image.
- Each camera company has its own file format.
  - .cr2 (Canon), .dng (Adobe), .tif (Kodak)
- Windows has .bmp
  - Viewable with Microsoft Office, paint, Photoshop

Windows BMP File Format

- 54 bytes for the header, beginning with “BM”.
- Then height * width * 3 bytes for the image.
- Can be viewed with IE or Windows Media tools.
- Uses little endian for multi-byte parameters.
- Image rows start at bottom.

HexDump of BMP Header

- Highlighted are: (0x424d = “BM”)
  - Image offset (0x36) (14 + 40 = 54)
  - Header length (0x28 = 40 header begins here)
  - Width (0x0200 = 512)
  - Height (0x0200 = 512)
  - Depth (bits per pixel) (0x18 = 24)

```
0       424d 3600 0c00 0000 0000 3600 0000 2800   BM6.....6...(.
10      0000 0002 0000 0002 0000 0100 1800 0000   ................
20      0000 0000 0c00 0000 0000 0000 0000 0000   ................
30      0000 0000 0000 0000 5d24 4f77 3a87 8b39   ........]$Ow:..9
40      8f88 48a4 8d3b 7188 214c 6530 7c80 3285   ..H..;q.!Le0|.2.
```

Reading Dimension from BMP Header

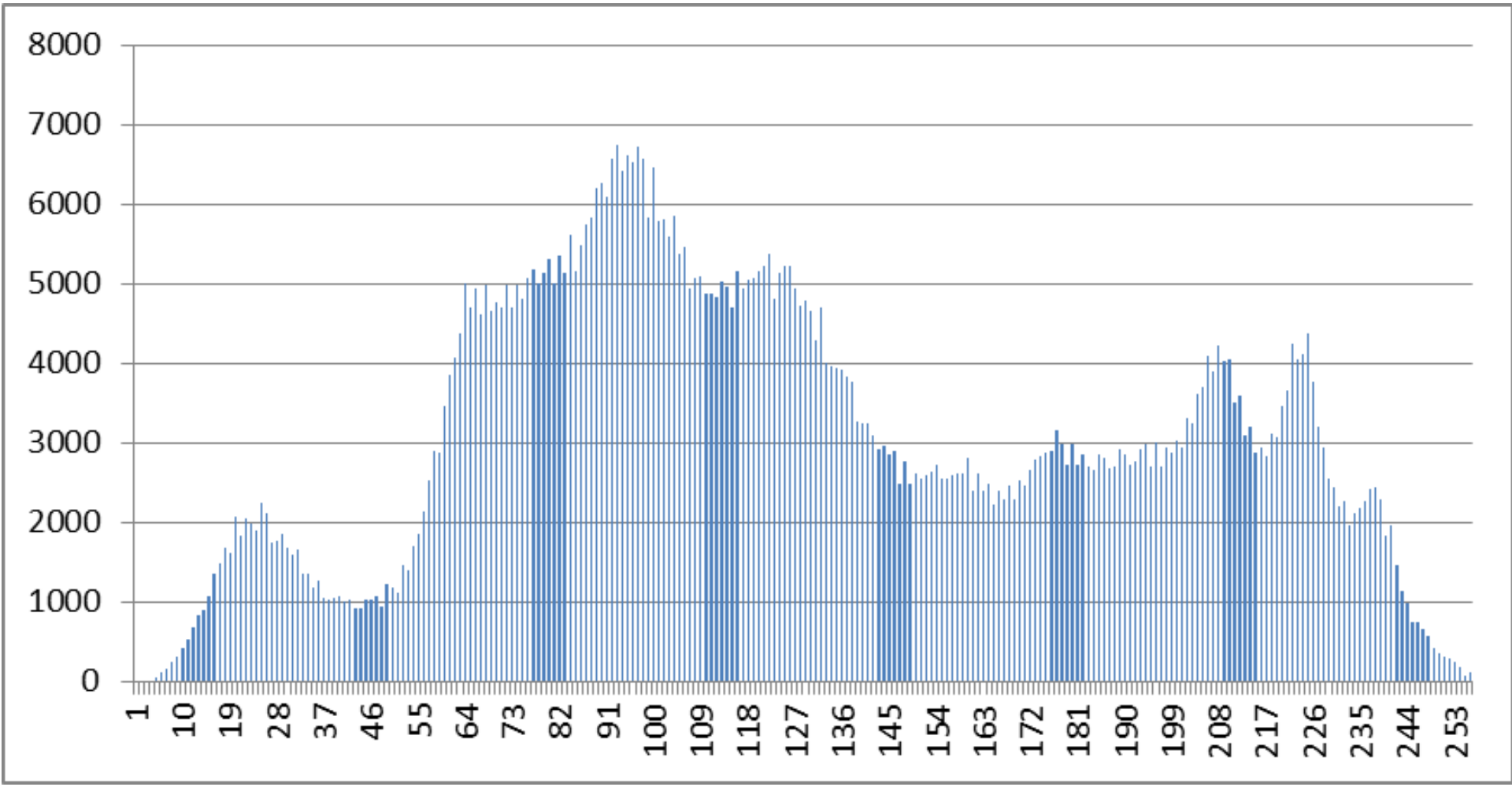
```
void readHeader(){
   byte[] header = new byte[54]; // 54 bytes for header
   try {
      System.in.read(header);
      System.out.write(header);
   } catch (IOException e){
     System.err.println(e.getMessage());
     System.exit(1);
   }
   if (header[0] != 'B' || header[1] != 'M'
      || header[14] != 40 || header[28] != 24)
     System.exit(1);
   int w1 = header[18]; int w2 = header[19];
   if (w1 < 0) w1 += 256; if (w2 < 0) w2 += 256;
   width = w2 * 256 + w1;
   int h1 = header[22]; int h2 = header[23];
   if (h1 < 0) h1 += 256; if (h2 < 0) h2 += 256;
   height = h2 * 256 + h1;
 }
```

LenaRGB.bmp 512x512, 24-bit

Reading Pixel Values

```
void readImage(){
   byte[] image = new byte[height * width * 3];
   raw = new short[height][width][3]; // upside down
   try {
      System.in.read(image);
   } catch (IOException e){
     System.err.println(e.getMessage());
     System.exit(1);
   }
   int index = 0;
   for (int i = 0; i < height; i++)
    for (int j = 0; j < width; j++)
      for (int k = 0; k < 3; k++){
        raw[i][j][k] = (short)image[index++];
        if (raw[i][j][k] < 0) raw[i][j][k] += 256;
   }
 }
```

Lena Pixel Values Entropy = 7.75



JPEG-LS

- Predict x using a, b, and c.
- If c=max(a,b,c), predict x=min(a,b).
- If c=min(a,b,c), predict x=max(a,b).
- Else predict x=a+b-c.

c	b
a	x

Predicting Using Neighboring Values

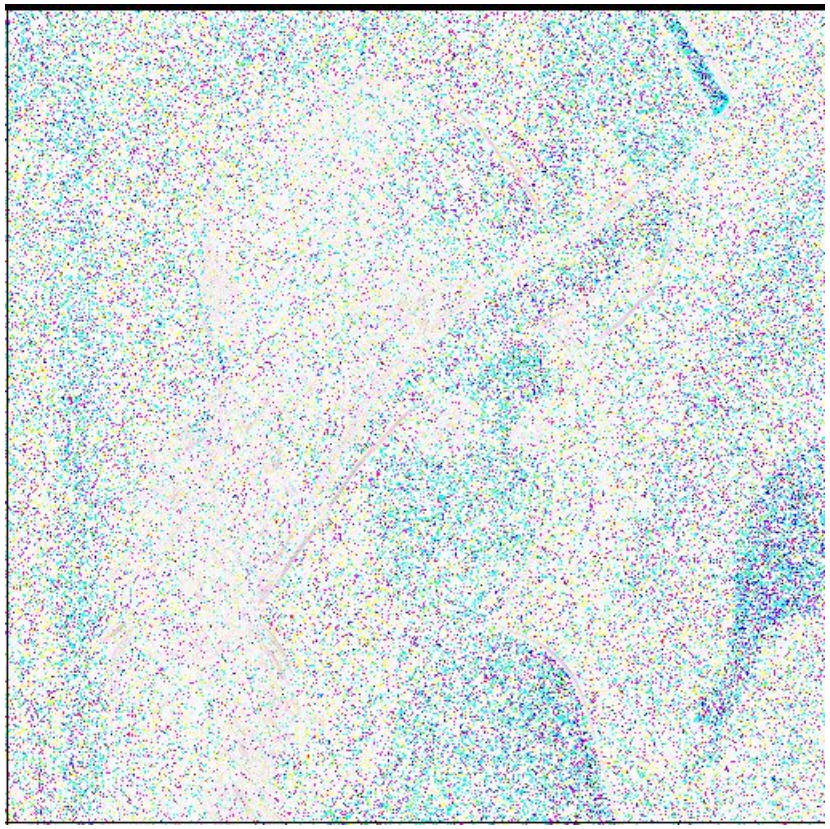
```
int predict(int a, int b, int c){
     int x;
     if ((c >= a) && (c >= b)) x = (a >= b) ? b : a;
     else if ((c <= a) && (c <= b)) x = (a >= b) ? a : b;
     else x = a + b - c;
     return x;
}
```

Prediction Errors for Lena

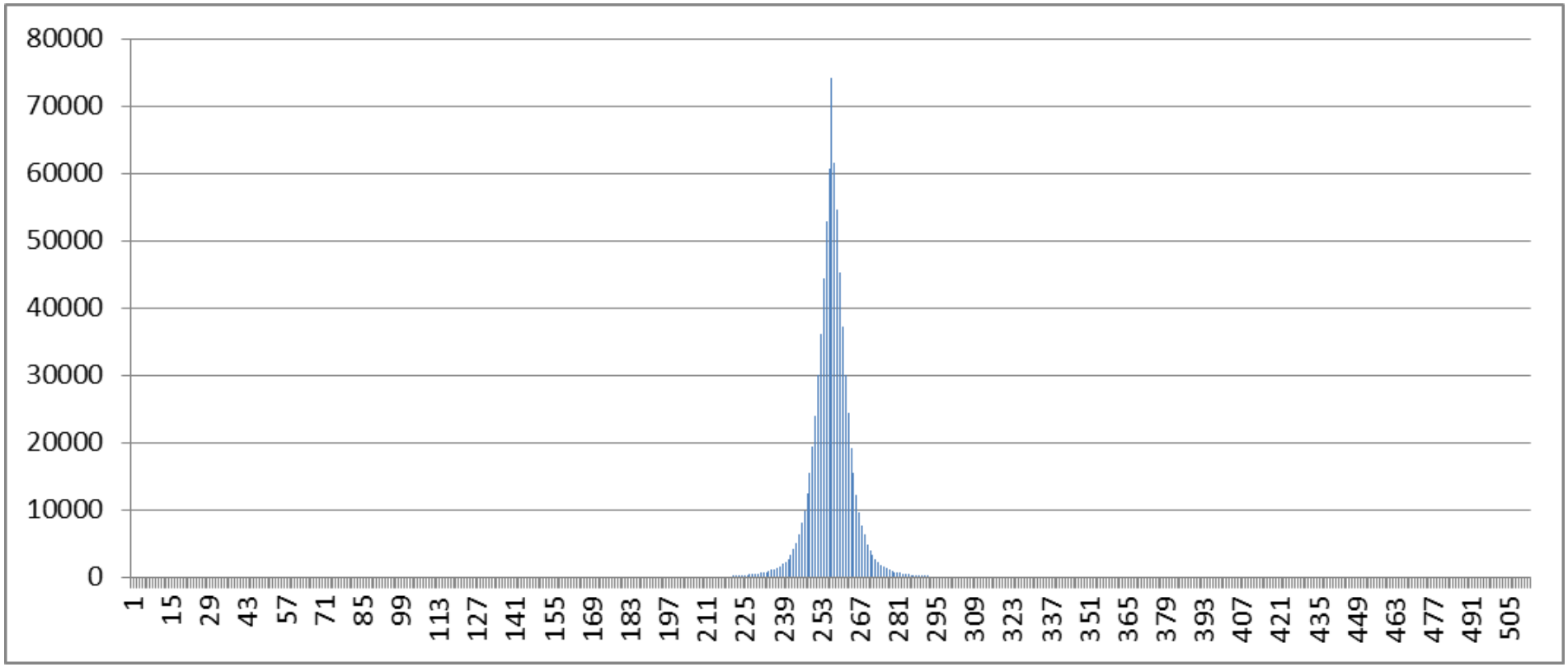
- There are 512 x 512 x 3 = 786,432 pixel values.
- 74130 values (9.4%) are correctly predicted.
  - 2066 of them have a=b=c=x.
- Prediction error = actual value – predicted value.
- Error range [-255, 255]

Lena |Error| from Prediction

256 - Lena Prediction |Errors|



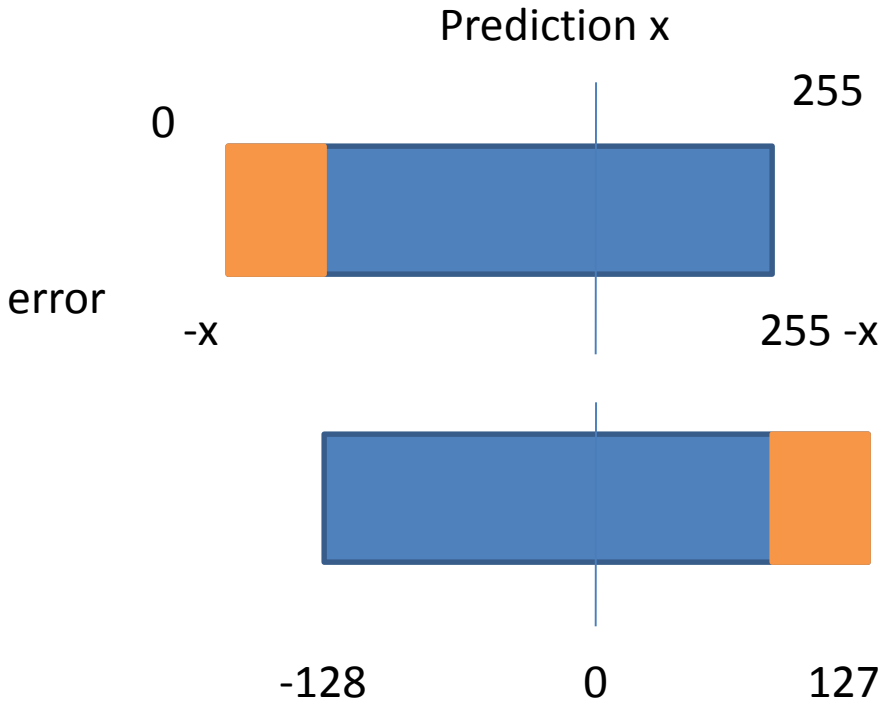
Lena Prediction Error + 256 Distribution



Error Mapping

- int e = value - x;
- if (e > 127) e -= 256;
- else if (e < -128) e += 256;
- value in [0, 255], e in [-x, 255-x]
- If 255-x > 127, -x > -255+127=-128
- Move [128, 255-x] to [-128, -x-1]
- This is e -= 256 when e > 127

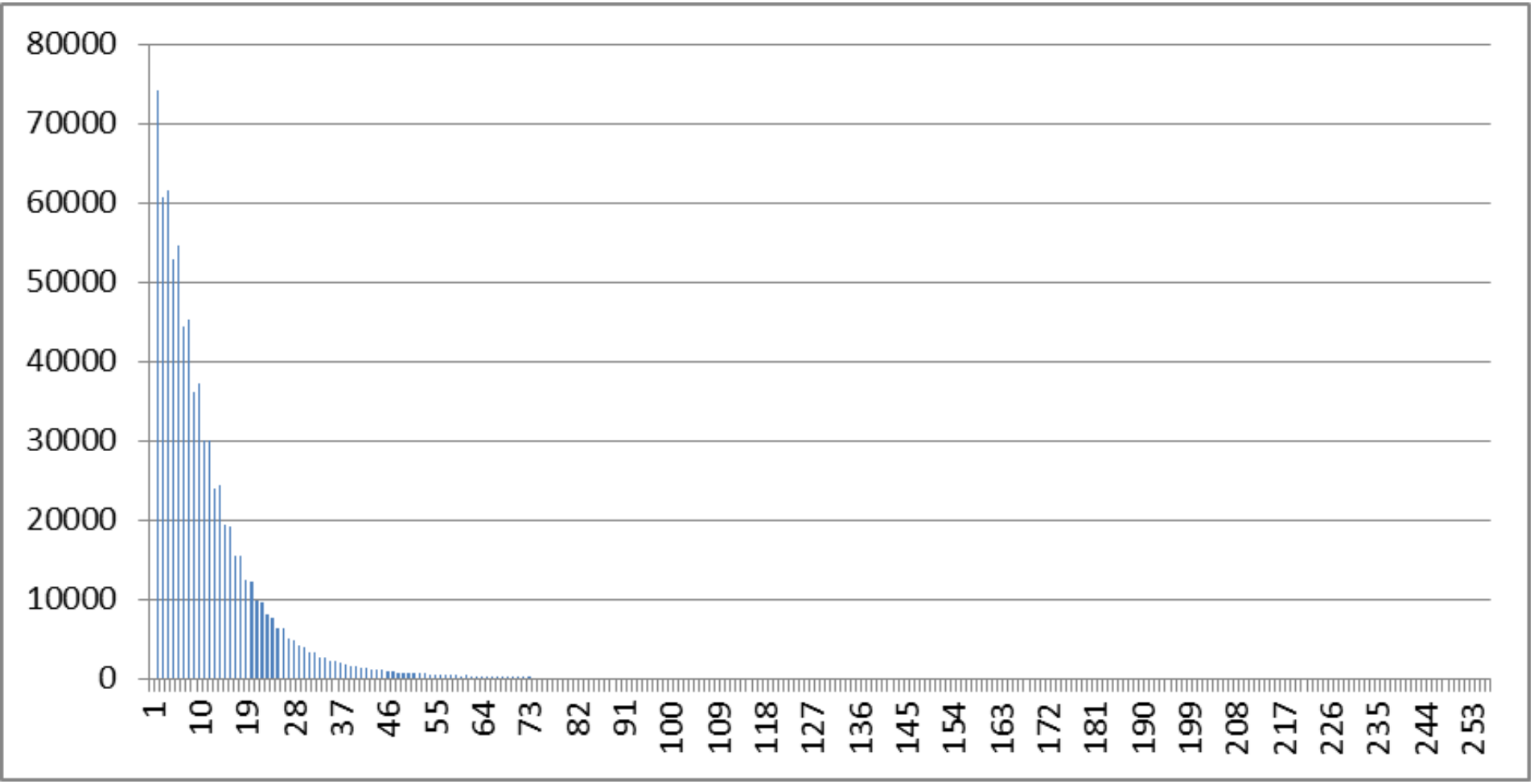
Error Mapping



Interlaced Error Array

- e = (e >= 0) ? e * 2 : -e * 2 - 1;
- If e is nonnegative, e become an even number (multiplied by 2)
- If e is negative, e is mapped to a positive odd number.
- The mapping is 0 -1 1 -2 2 -3 3 ... into 0 1 2 3 4..
- Gamma or C^1 (Fibonacci code) can now be used on these nonnegative integers.

Lena Prediction Error Entropy = 4.83



Mapping Error to [0, 255]

```
// map the prediction error to nonnegatives
  int mapError(int value, int prediction){
     int e = value - prediction;   //  prediction error
     if (e > 127) e -= 256; //  putting error in [-128, 127]
     else if (e < -128) e += 256;
     int error = (e >= 0) ? e * 2 : -e * 2 - 1;
       //  into 0 -1 1 -2 2 array (interlacing)
     return error;
  }
```

Unwrapping the Error

- Reverse error = (e >= 0) ? e * 2 : -e * 2 - 1;
- If e is less than –x (x is the prediction), then add 256 to it.
- If e is greater than 255-x, then subtract 256 from it.
- x+e is the decoded pixel value.

Unmapping Error

```
short unmapError(int error, int predicted){
    int e = 0;
    // Your code to reverse the line in H4A:
    // error = (e >= 0) ? e * 2 : -e * 2 - 1;
    int value = predicted + e;
    if (value > 255) value -= 256;
    else if (value < 0) value += 256;
    return (short)value;
 }
```

Read in the C^1 Codeword

- Read bits in and if the ith bit is 1, add fib[i] to error, until two consecutive 1's are read.
- example: 1001011 → e = fib[0] + fib[4] + fib[6] = 1 + 5 + 13 = 19
- This is performed by int deFib(), to read the next C^1 codeword and return the value (in [0, 255]) it encodes.

Homework 4: due 1-28-15

- Complete H4B and H4C so that H4B is the inverse of H4A and H4C is the inverse of H3A c1code.txt.
- test4.c1 is the result of
  - > java H4A < mystery.bmp > t
  - > java H3A c1code.txt < t > test4.c1
- You may recover mystery.bmp using H4C and then H4B. Submit a report including your code and a jpg version of mystery.bmp.