Bi624 Problem Set #1

Alexa Dowdell

Thu Oct 5 21:38:32 2017

SF-Seq Quality Assessment Assignment

Data

We were assigned two of the demultiplexed file pairs found in `/projects/bgmp/Bi624/PS1_file_assignments.tsv` . Specifically, I was assigned libraries: `32_4G_both` and `22_3H_both` . The associated demultiplexed .fastq files for the assigned libraries found in `/projects/bgmp/2017_sequencing/demultiplexed/` were copied to the approapriate working directory. Each library was associated with two files:

22_3H_both:

```
22_3H_both_S16_L008_R1_001.fastq.gz

22_3H_both_S16_L008_R2_001.fastq.gz
```

32_4G_both:

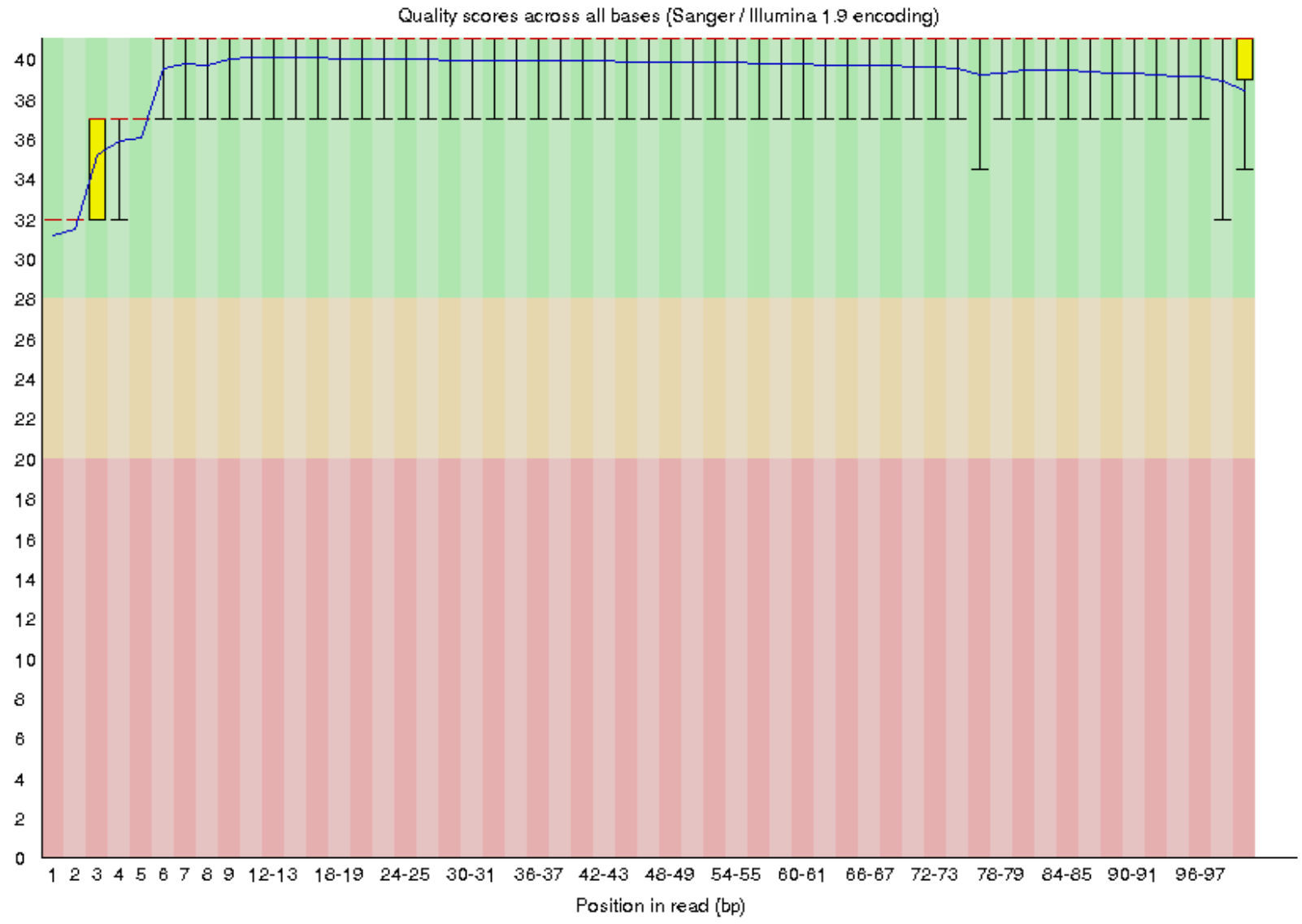
```
32_4G_both_S23_L008_R1_001.fastq.gz

32_4G_both_S23_L008_R2_001.fastq.gz
```

Part 1 – SF-Seq read quality score distributions

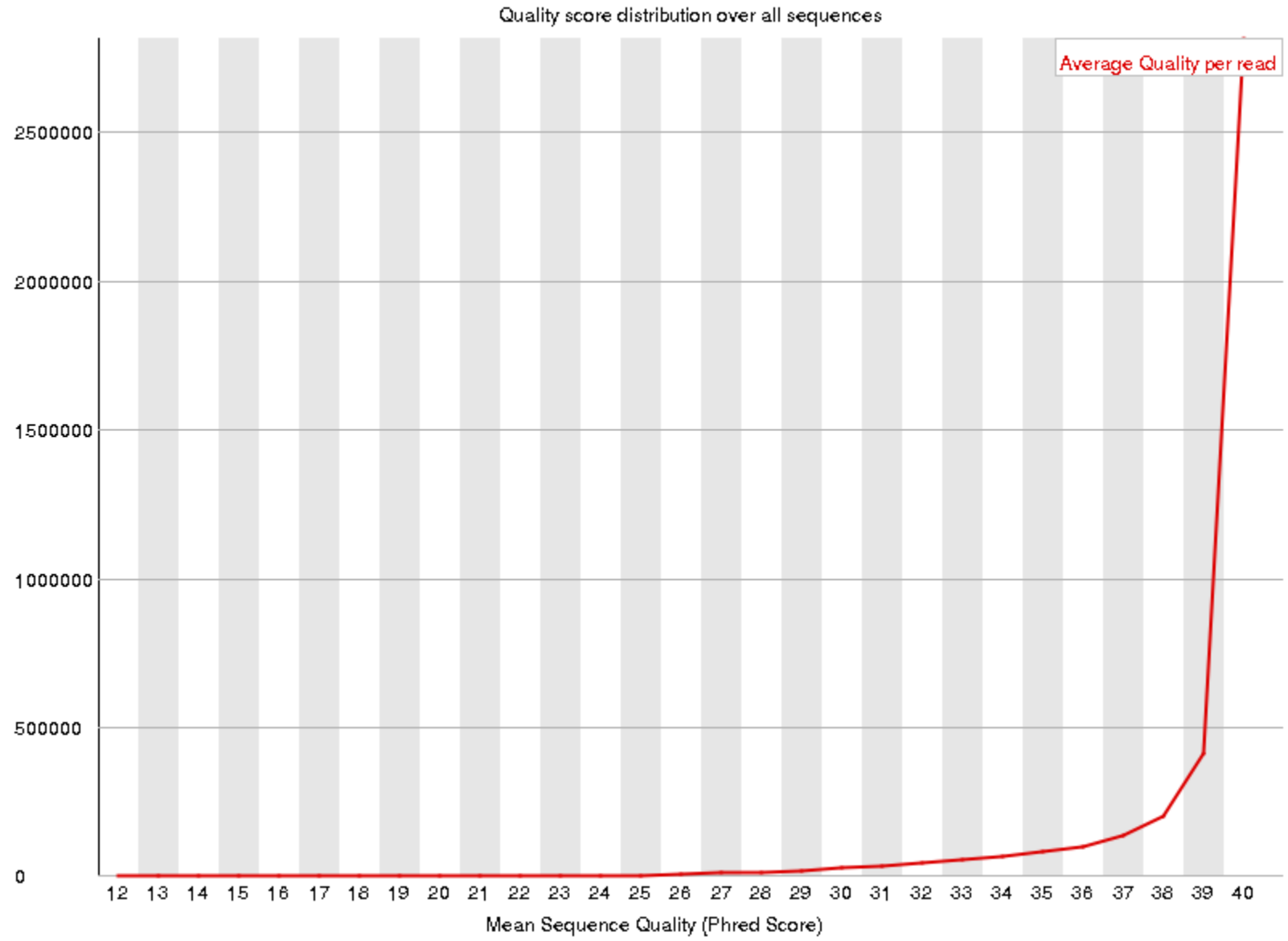
Using FastQC on Talapas, produce plots of quality score distributions for forward and reverse reads. Also, produce plots of the per-base N content, and comment on whether or not they are consistent with the quality score plots.

22_3H_both R1

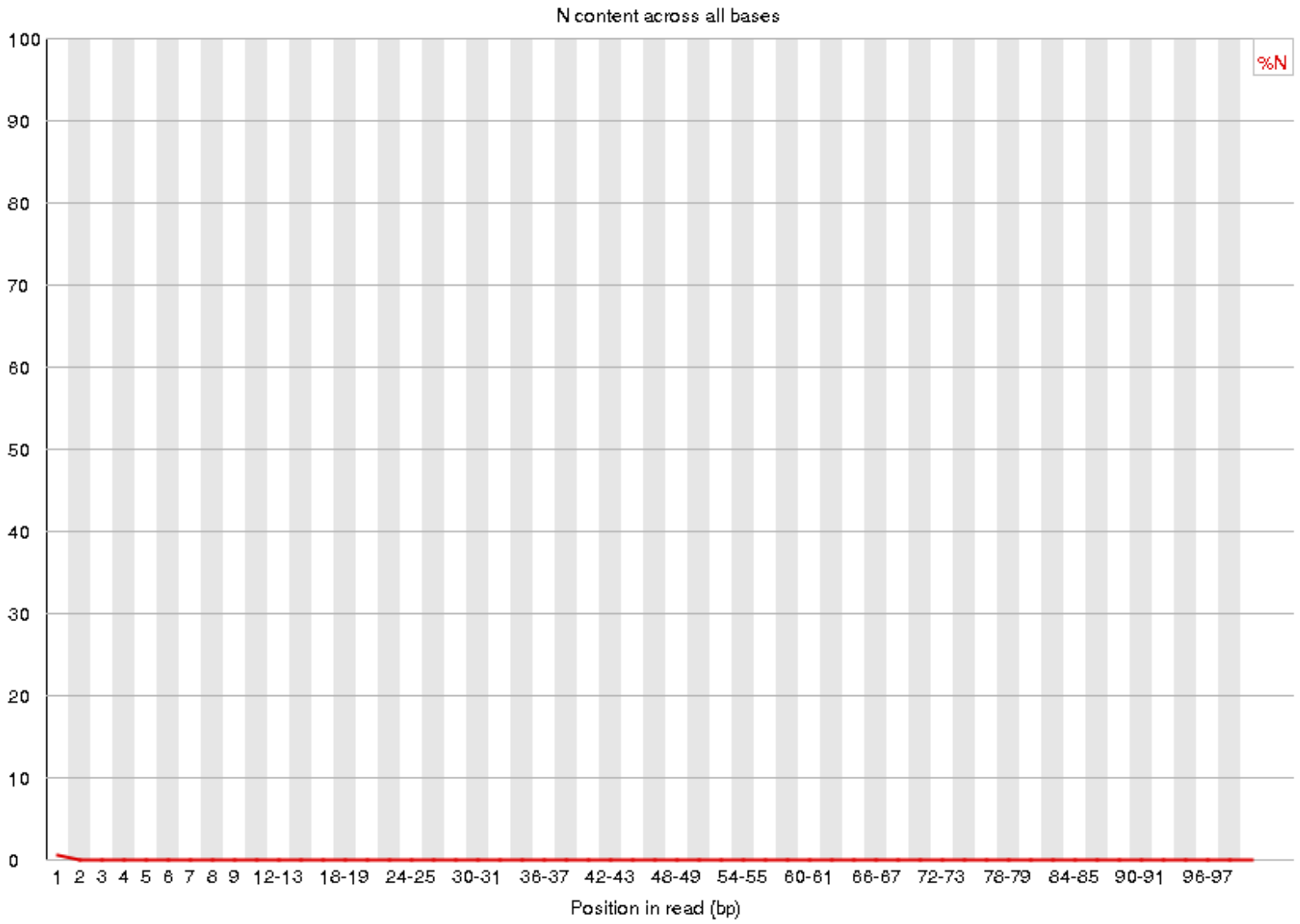
22_3H_both R1: Per base sequence quality



22_3H_both R1: Per sequence quality scores

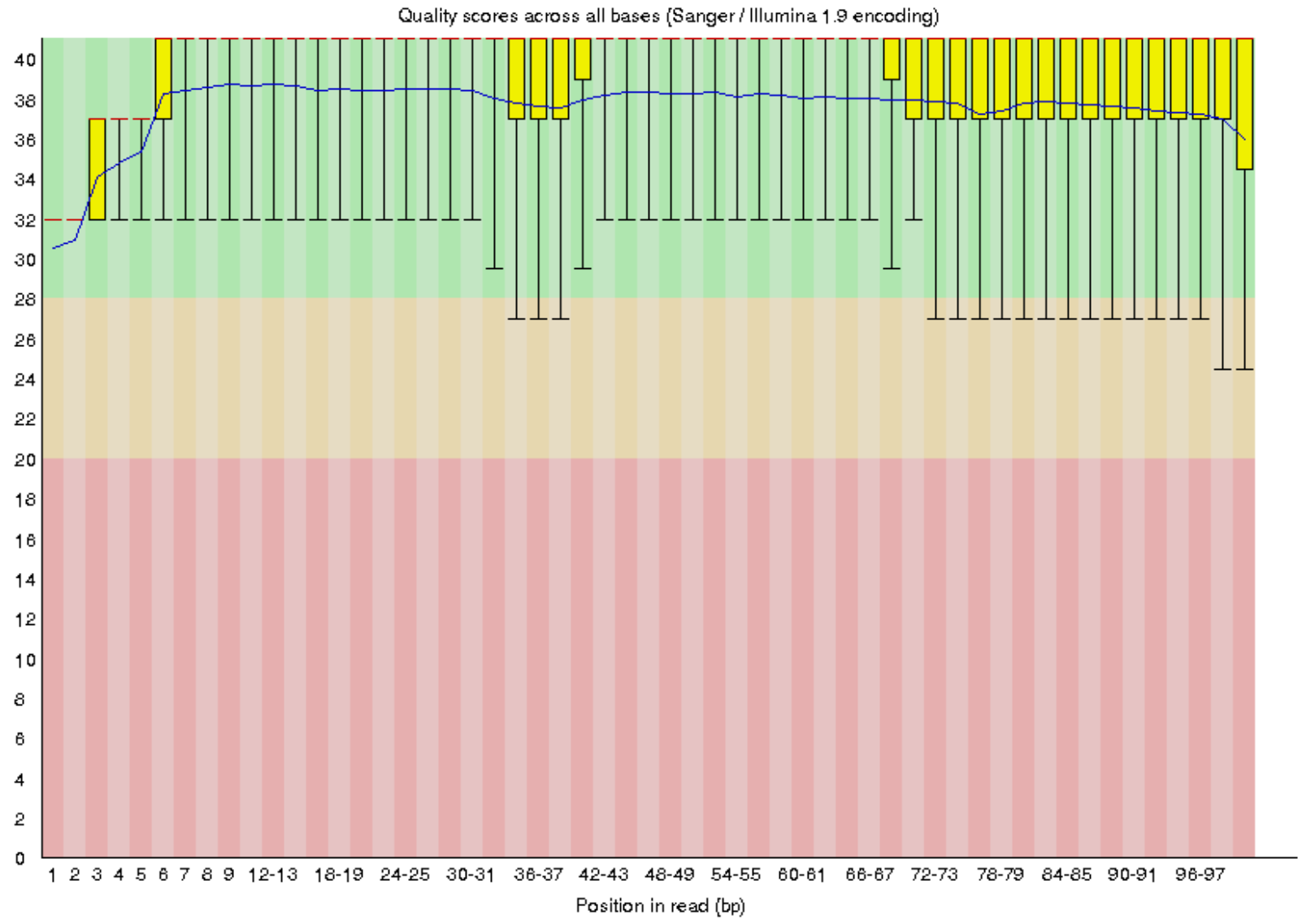


22_3H_both R1: Per-base N Content

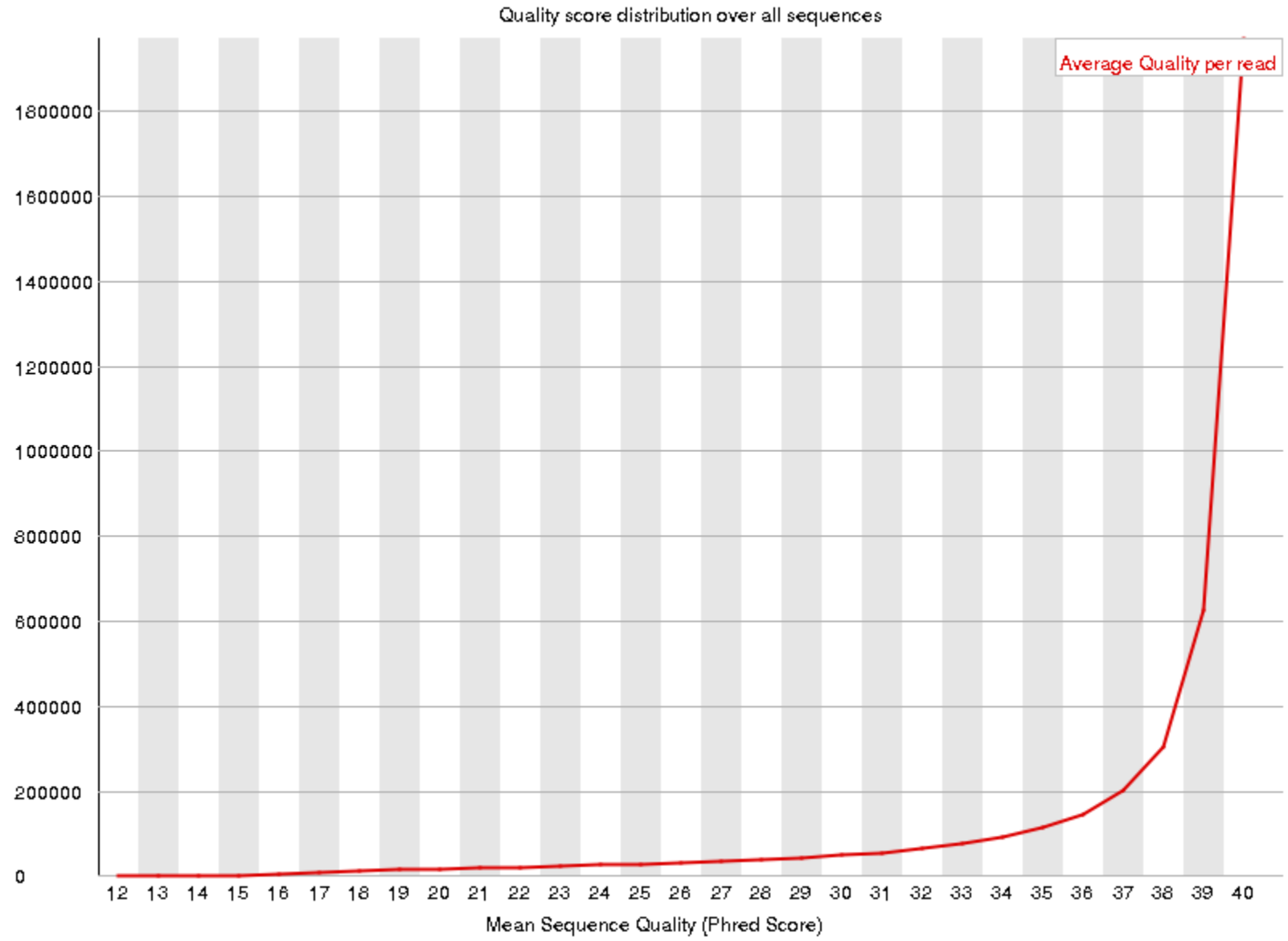


22_3H_both R2

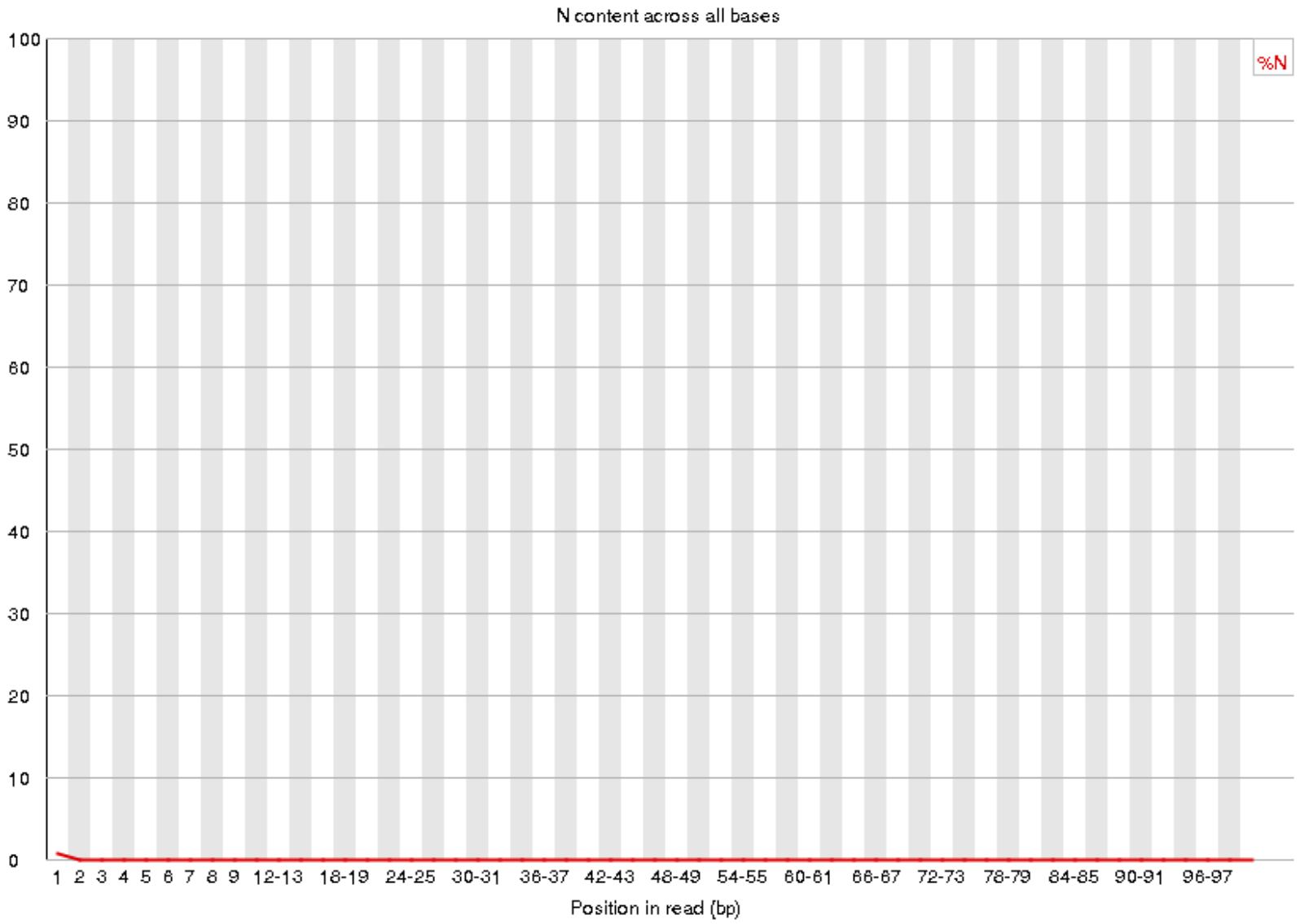
22_3H_both R2: Per base sequence quality



22_3H_both R2: Per sequence quality scores



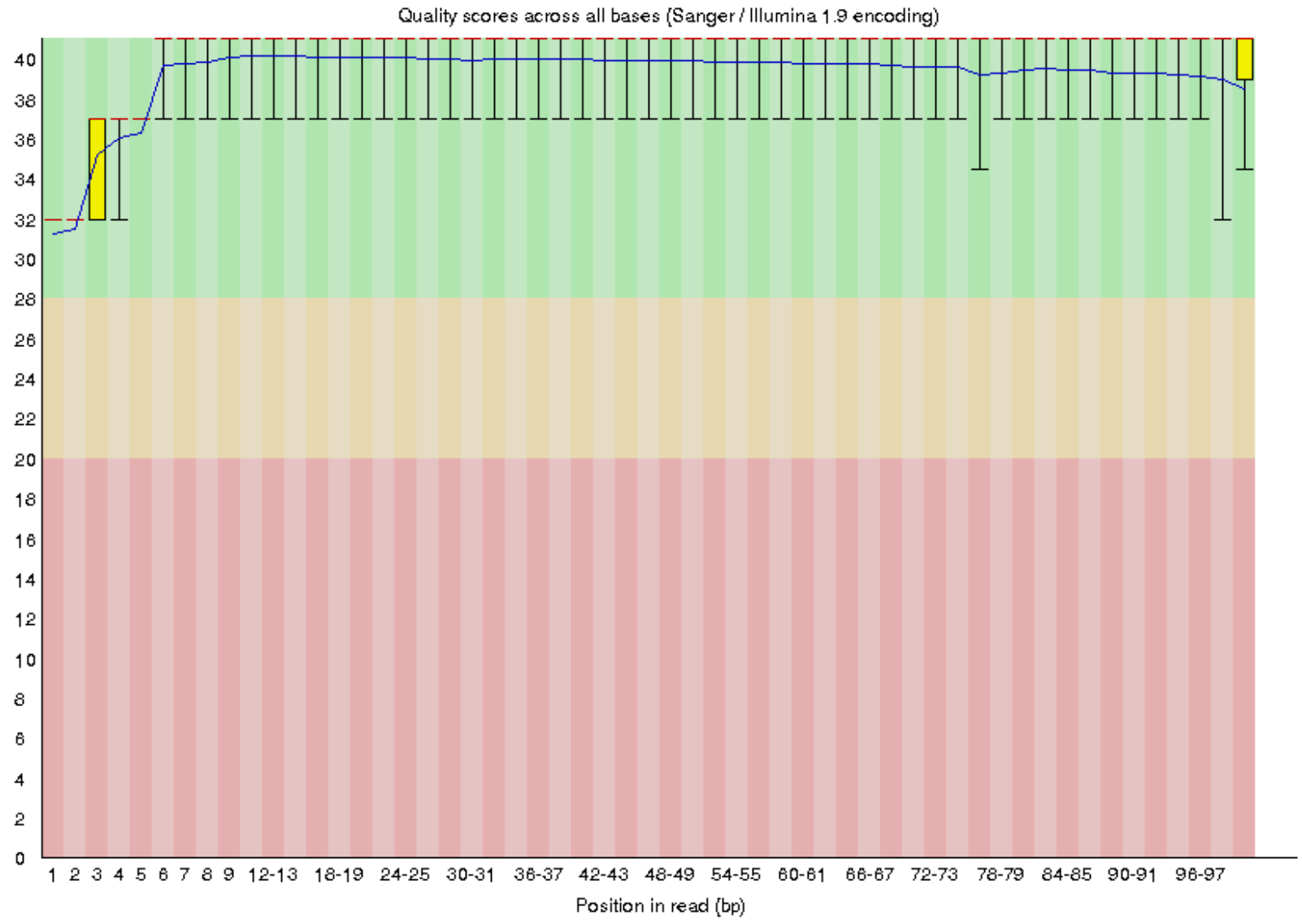
22_3H_both R2: Per-base N Content



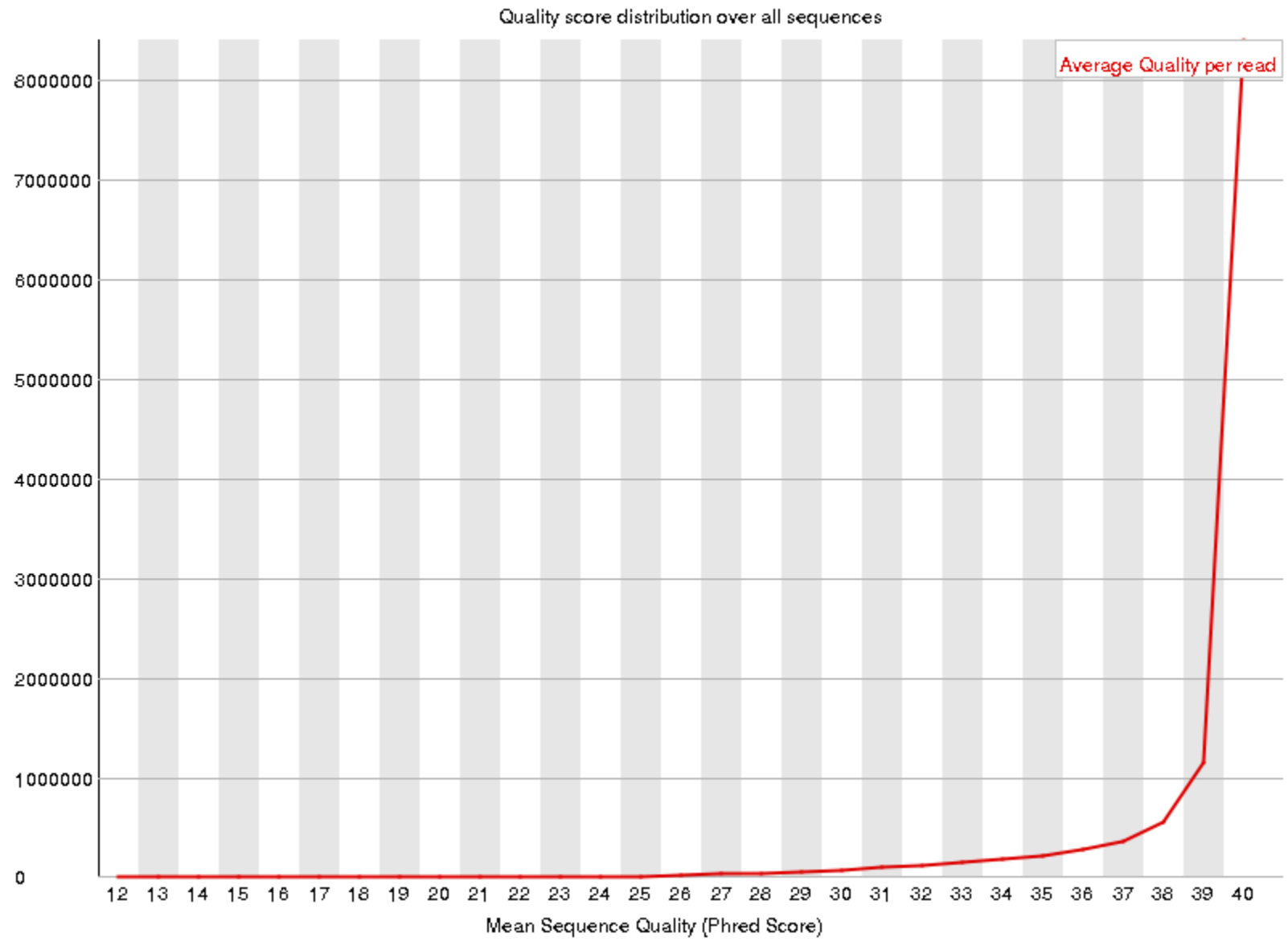
In the 22_3H_both library, the reverse reads in comparison to the forward (R1) reads has a pretty consistently lower quality score across majority of the base pairs. It is evident in both the forward and reverse reads that there is a significant drop in average quality score for the first 7 to 9 bases compared to the rest of the base positions. Likely the consistently lower average quality score seen for base position 1 through 8 corresponds to sequencing error. This trend would also be viewable on the per sequence quality scores graph as graphed levels of frequency below most abundant, at mean quality scores of between 31 and approximately 38. Furthermore the lower average quality scores can also be seen in the per base N content graph with the slightly raised level of N % within the 0 and 2 read positions. Higher average quality scores correspond with lower non-existent per base N content % therefore it makes sense the lower quality scores at the beginning single digit bp spike N content.

32_4G_both R1

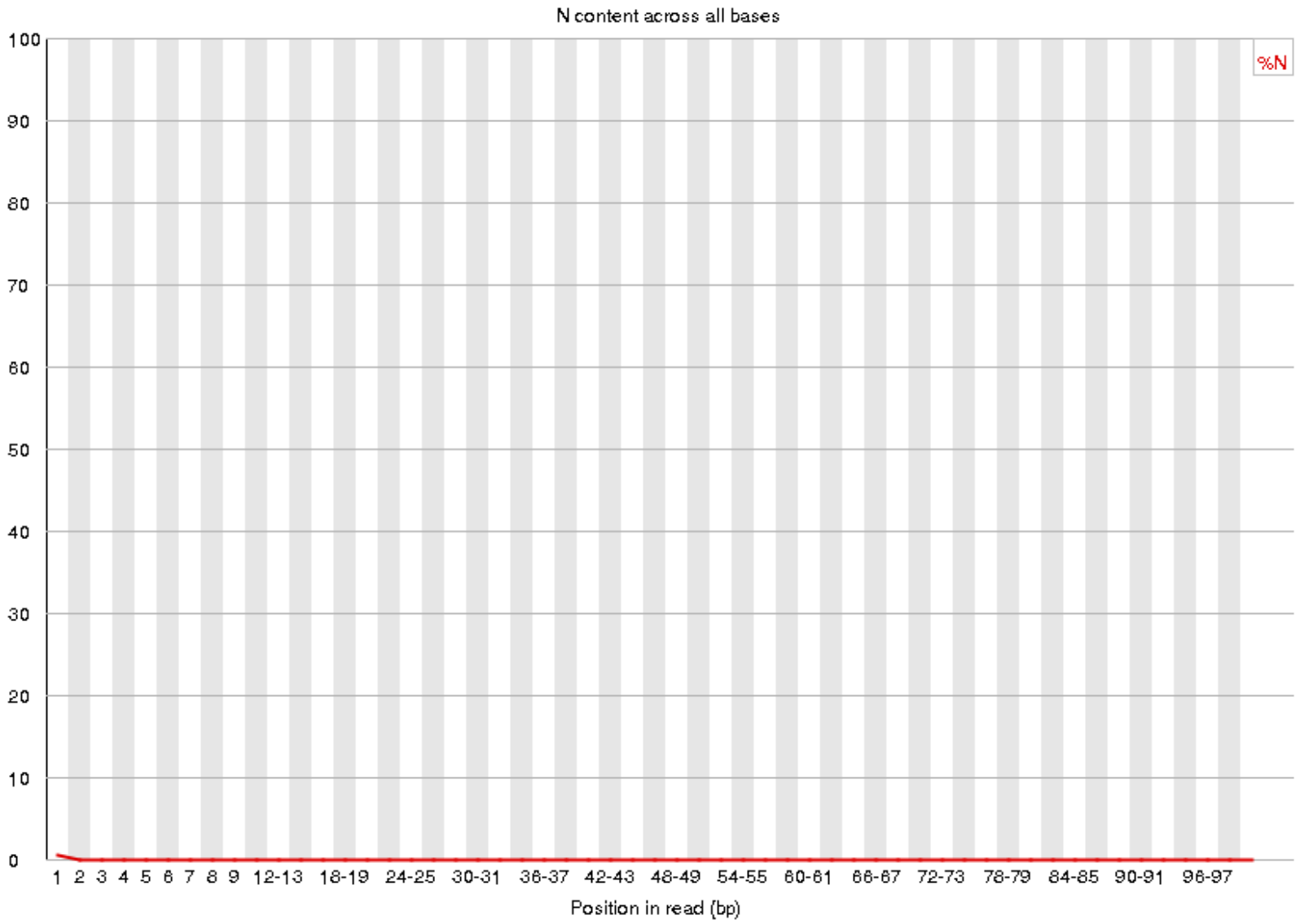
32_4G_both R1: Per base sequence quality



32_4G_both R1: Per sequence quality scores

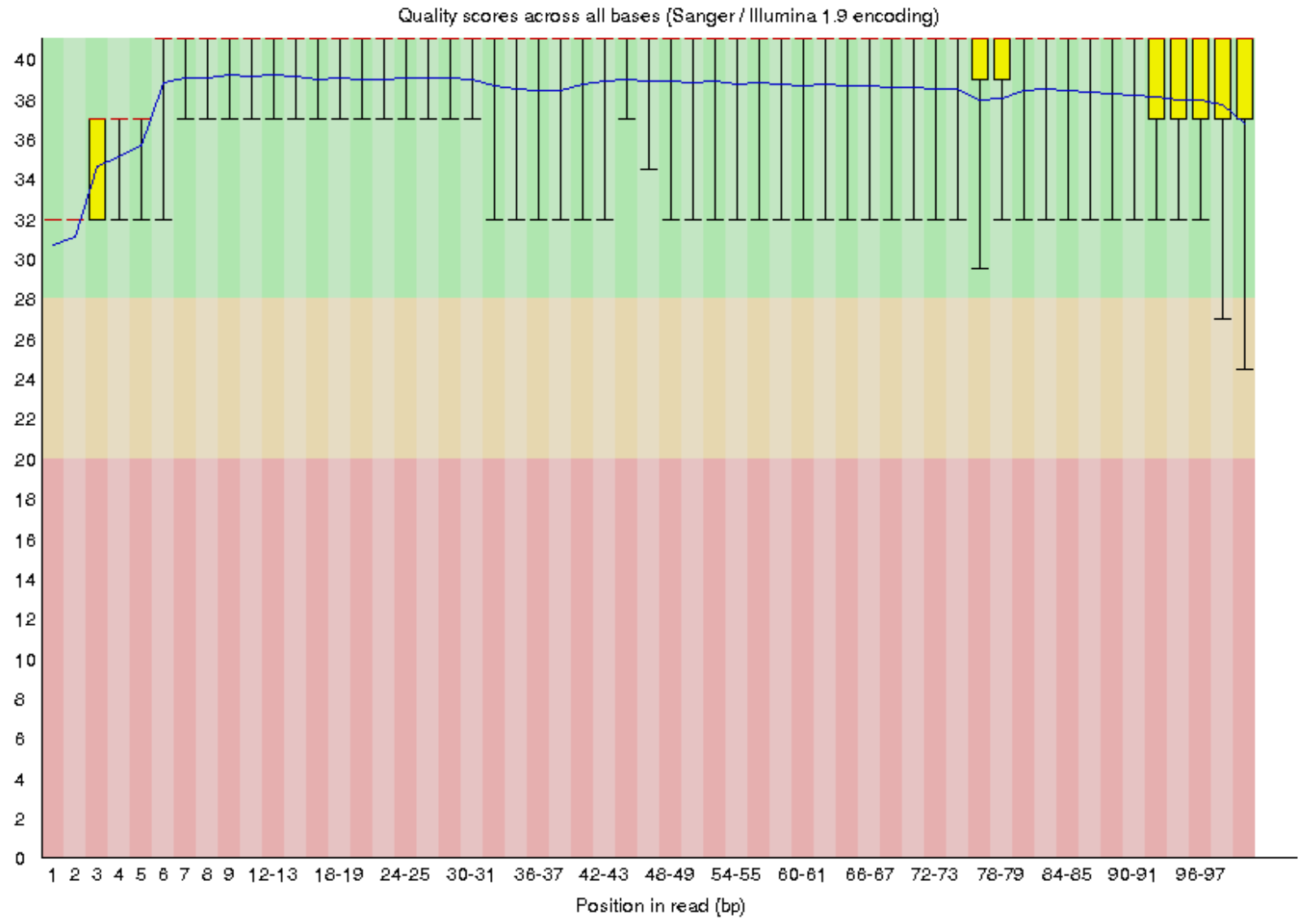


32_4G_both R1: Per-base N Content

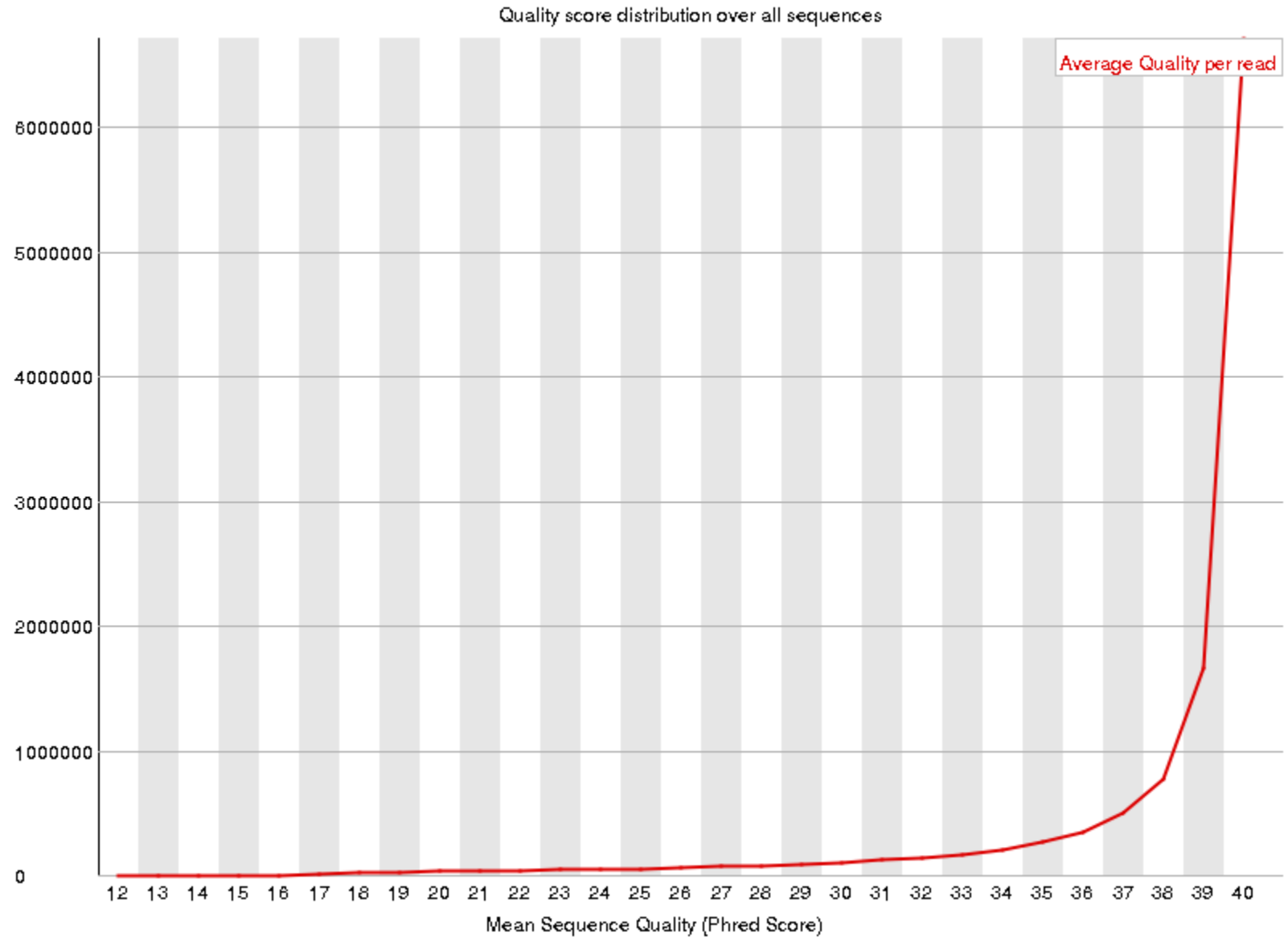


32_4G_both R2

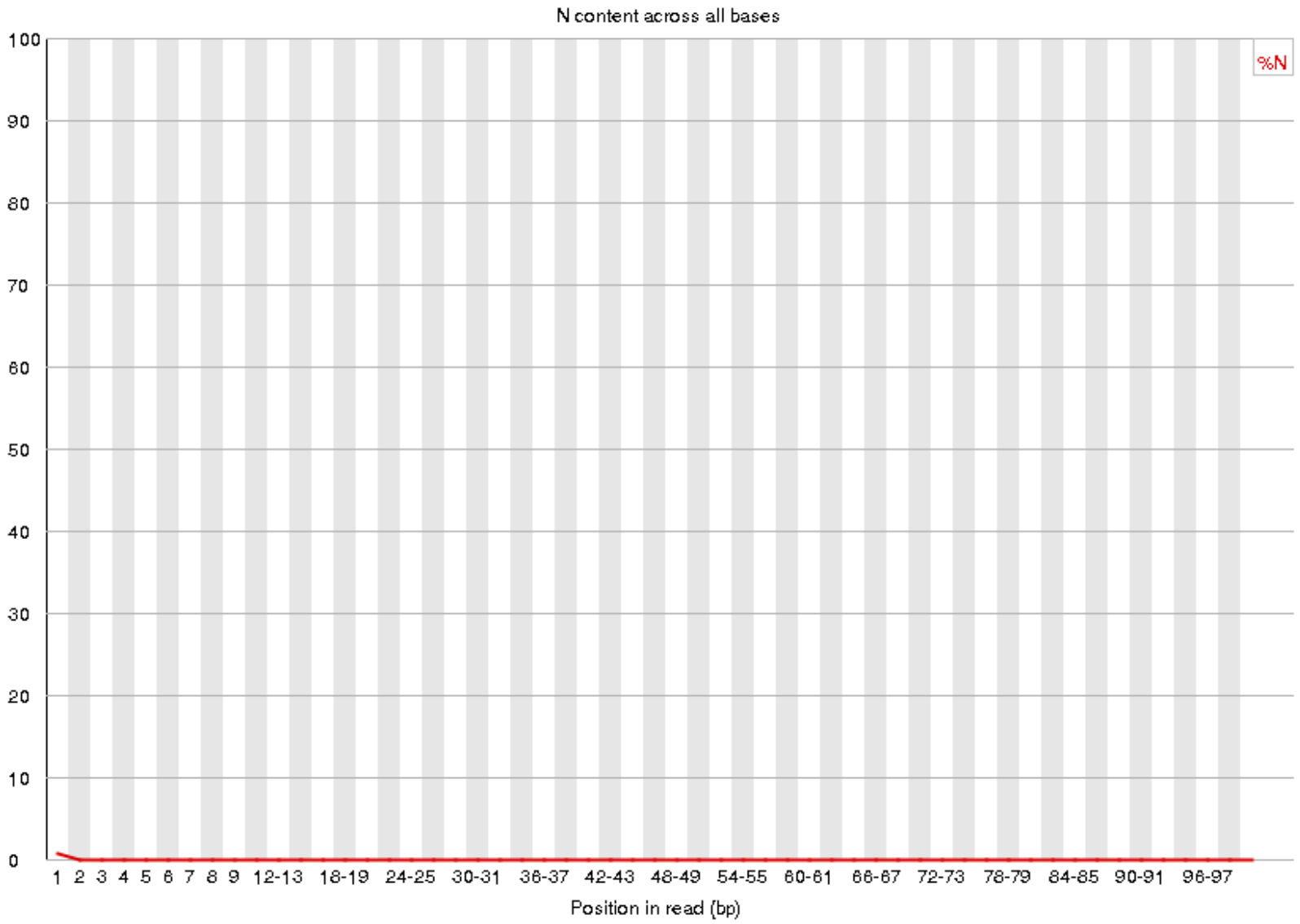
32_4G_both R2: Per base sequence quality



32_4G_both R2: Per sequence quality scores



32_4G_both R2: Per-base N Content



Likewise, in the 32_4G_both library the reverse reads are consistently about a single average quality score point lower than the forward reads across majority of the base positions. But similar to the 22_3H_both library, in both the forward and reverse reads there is a significant drop in average quality score for the first 7 to 8 bases compared to the rest of the base positions. The trend can be seen similarly in the per sequence quality scores distribution in the presence of frequency levels greater than zero for average quality scores between 31 and 37. Again, the lower average quality scores can also be seen in the per base N content graph with the slightly raised level of N % within the 0 and 2 read positions.

Run your quality score plotting script from the index hopping assignment. Describe how the FastQC quality score distribution plots compare to your own. If different, propose an explanation. Also, does the runtime differ? If so, why?

Two seperate scripts were written to produce distributions for per base sequence quality plots and per sequence quality score plots: `part1Hist1.py` and `part2FreqDist.py`.

Per base sequence quality

The per base sequence quality plots Python script `part1Hist1.py` was written to take in one argument for a file, and was hard coded to write out a tab seperated distribution with the name specified. The output file contains two columns, each base position and the mean quality score at each base position. Almost identical scripts were run for both libraries on all four files just with slightly different output file names corresponding to the file passed in. Below is `part1Hist1.py`:

```
#!usr/bin/env python
import argparse

parser = argparse.ArgumentParser(description="Quality Index Swapping")
parser.add_argument("-f", "--filepath", help="File", required=True, type=str)
args = parser.parse_args() #sets arguments as call-able

f = args.filepath

mean_scores = [0.0]*101 #initialize a vector 101 in length to floats of 0.0

def convert_phred(letter): #function to convert a ASCII character to its correspondin
g phred score
    n = ord(letter)-33
    return n

i = 0 #initialize a counter
with open(f, "r") as fh: #open the file to read in
    for line in fh:
        line = line.strip("\n")
        i+=1
        if i%4 == 0: #if the line is a quality score line
            j = 0
            for char in line: #for each specific base in that line
                score = convert_phred(char) #convert the base to a phred score
                mean_scores[j] = mean_scores[j] + score #add the mean score to the su
m for that base position
                j+=1
        if i%5000000: #for debugging on Talapas, print progress
            print("On line: ", i, flush = True)

for i in range(101): #take each mean sum at each index position and divide
# by the total number of quality score lines. (total wc -l of file / 4)
    mean_scores[i] = mean_scores[i]/4050899

with open("22_3H_R1_dist_output.tsv", "w") as output:
    output.write("Base Position" + "\t" + "Mean Q_score" + "\n")
    for i in range(101):
        output.write(str(i) + "\t" + str(mean_scores[i]) + "\n")
```

Four shell scripts were submitted, one for each of the forward and reverse reads in each of the two libraries. An example of one of the shell scripts is below:

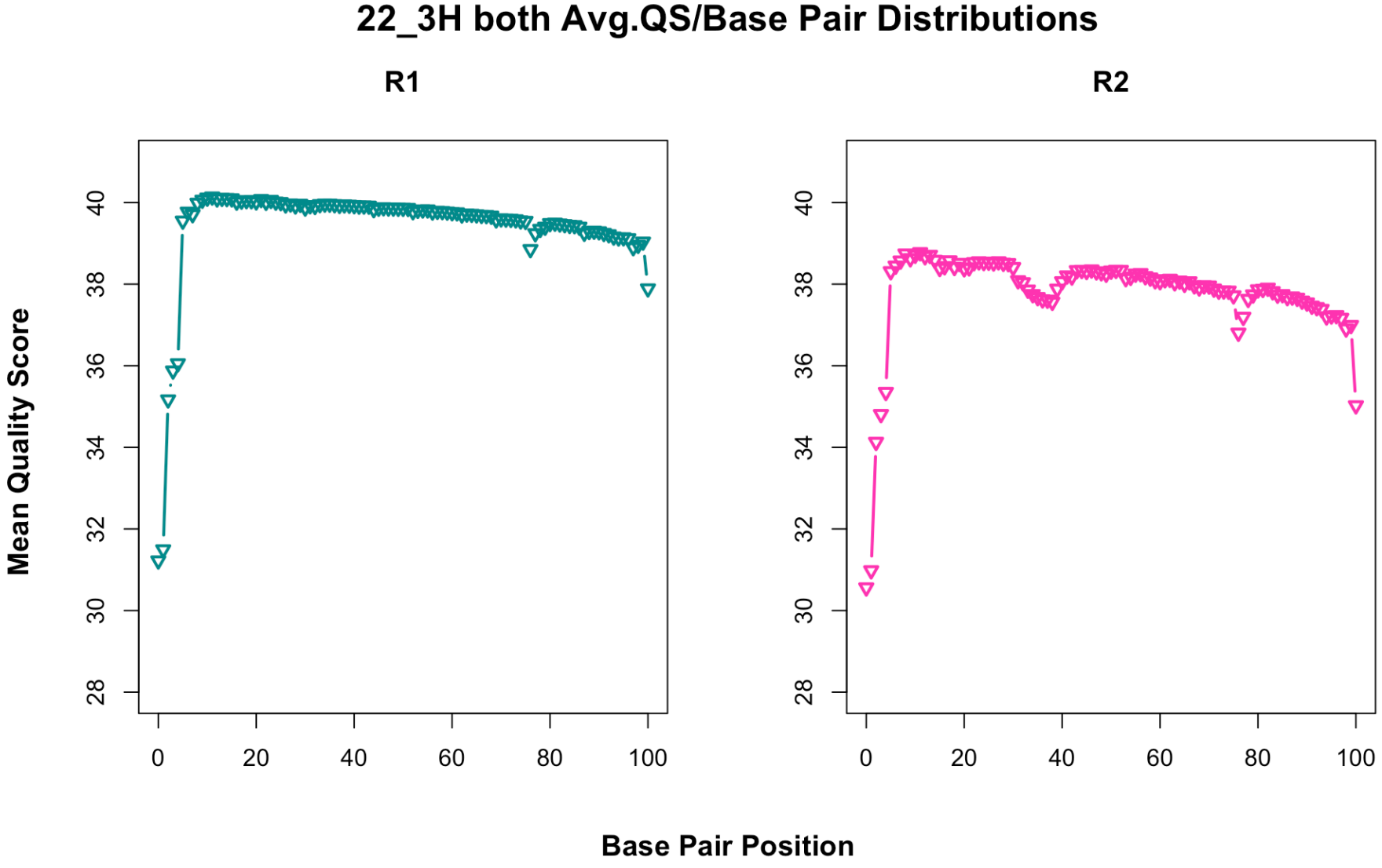
```
#!/bin/bash

#SBATCH --partition=short        ### Indicate want long node
#SBATCH --job-name=AD_R1     ### Job Name
#SBATCH --output=AD_R1.out         ### File in which to store job output
#SBATCH --error=AD_R1.err          ### File in which to store job error messages
#SBATCH --time=0-02:00:00   ### Wall clock time limit in Days-HH:MM:SS
#SBATCH --nodes=1                ### Node count required for the job
#SBATCH --ntasks-per-node=28     ### Nuber of tasks to be launched per Node

ml easybuild GCC/6.3.0-2.27 OpenMPI/2.0.2 Python/3.6.1

python part1Hist1.py -f 22_3H_both_S16_L008_R1_001.fastq
```

Note: The original demultiplexed libraries had a .gz extension and the `gunzip` function was used to decompress the .fastq files so the Python script above would run properly. Heading the first 20 lines of one of the output files `22_3H_R1_dist_output.tsv` looks as follows:

```
Base Position   Mean Q_score
0   31.222923109166633
1   31.49547372077161
2   35.169197750919
3   35.87730723476443
4   36.05591277392993
5   39.5506106175444
6   39.77021841324605
7   39.70922479183016
8   39.98892369323451
9   40.06332273404002
10  40.11517838385998
11  40.142210654968196
12  40.09451605680616
13  40.109002470809564
14  40.10371549623923
15  40.09659041116552
16  40.016267500127746
17  40.04215878993774
18  40.05327138494443
```

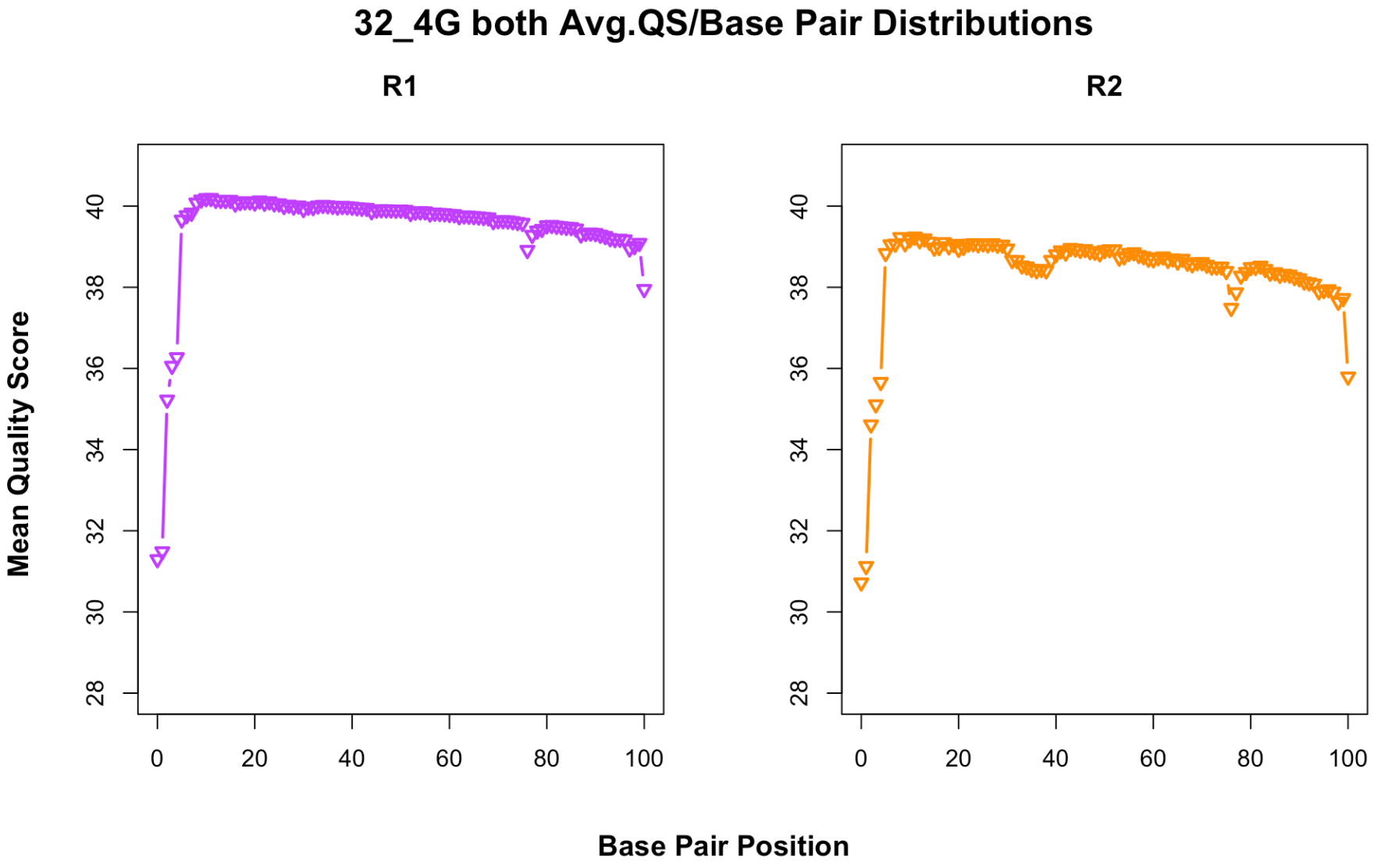
22_3H_both R1 and R2: Per base sequence quality

```
R1_22_3H <- read.table("/Users/adowdell/sf-seq-qaa-alexadowdell/Part1/22_3H_R1_dist_o
utput.tsv", head = TRUE,
    sep = "\t")

R2_22_3H <- read.table("/Users/adowdell/sf-seq-qaa-alexadowdell/Part1/22_3H_R2_dist_o
utput.tsv", head = TRUE,
    sep = "\t")

par(mfrow = c(1, 2), oma = c(0, 1, 1, 0))
plot(R1_22_3H$Mean.Q_score ~ R1_22_3H$Base.Position, main = "R1", pch = 6, cex = 0.9,
type = "b", lwd = 2,
    col = "cyan4", xlab = "", ylab = "", ylim = c(28, 41))

plot(R2_22_3H$Mean.Q_score ~ R2_22_3H$Base.Position, main = "R2", pch = 6, cex = 0.9,
type = "b", lwd = 2,
    col = "maroon1", xlab = "", ylab = "", ylim = c(28, 41))

mtext("22_3H both Avg.QS/Base Pair Distributions", side = 3, font = 2, outer = TRUE,
line = -0.5, cex = 1.5)
mtext("Mean Quality Score", side = 2, font = 2, outer = TRUE, line = -0.5, cex = 1.2)
mtext("Base Pair Position", side = 1, font = 2, outer = TRUE, line = -1, cex = 1.2)
```



32_4G_both R1 and R2: Per base sequence quality

```
R1_32_4G <- read.table("/Users/adowdell/sf-seq-qaa-alexadowdell/Part1/32_4G_R1_dist_o
utput.tsv", head = TRUE,
    sep = "\t")

R2_32_4G <- read.table("/Users/adowdell/sf-seq-qaa-alexadowdell/Part1/32_4G_R2_dist_o
utput.tsv", head = TRUE,
    sep = "\t")

par(mfrow = c(1, 2), oma = c(0, 1, 1, 0))
plot(R1_32_4G$Mean.Q_score ~ R1_32_4G$Base.Position, main = "R1", pch = 6, cex = 0.9,
type = "b", lwd = 2,
    col = "darkorchid1", xlab = "", ylab = "", ylim = c(28, 41))

plot(R2_32_4G$Mean.Q_score ~ R2_32_4G$Base.Position, main = "R2", pch = 6, cex = 0.9,
type = "b", lwd = 2,
    col = "darkorange", xlab = "", ylab = "", ylim = c(28, 41))

mtext("32_4G both Avg.QS/Base Pair Distributions", side = 3, font = 2, outer = TRUE,
line = -0.5, cex = 1.5)
mtext("Mean Quality Score", side = 2, font = 2, outer = TRUE, line = -0.5, cex = 1.2)
mtext("Base Pair Position", side = 1, font = 2, outer = TRUE, line = -1, cex = 1.2)
```



Per sequence quality scores

A second Python script `part2FreqDist.py` was created to produce quality score frequency distributions in each of the four files. The script takes four file arguments and is flexible to change the name of the output file corresponding to each of the files that is passed in. Each output file is a two column tab seperated distribution with an average quality score column and a frequency column containing the number of times that particular average quality score was passed. The `part2FreqDist.py` script looks like the following:

```
#!usr/bin/env python

import argparse

parser = argparse.ArgumentParser(description="Quality Index Swapping")
parser.add_argument("-f1", "--filepath1", help="R1 file", required=True, type=str)
parser.add_argument("-f2", "--filepath2", help="R2 file", required=True, type=str)
parser.add_argument("-f3", "--filepath3", help="R3 file", required=True, type=str)
parser.add_argument("-f4", "--filepath4", help="R4 file", required=True, type=str)

args = parser.parse_args() #sets arguments as call-able

f1 = args.filepath1
f2 = args.filepath2
f3 = args.filepath3
f4 = args.filepath4



# f1 = "./1294_S1_L008_R1_001_sample.fastq"
# f4 = "./1294_S1_L008_R4_001_sample.fastq"

def convert_phred(char):
    n = ord(char) - 33
    return n

def read_frequency(file, index, n):
    phredscore_dict = {}

    if index == True:
        length = 101

    # add up quality scores
    with open(file) as fh:
        i = 0  # line counter
        for line in fh:
            i += 1
            line = line.strip("\n")
            if i % 4 == 0:  # picks out only quality score lines
                sum1 = 0
                for char in line:
                    sum1 += ord(char) - 33
                mean = int(sum1 / length)
                if mean in phredscore_dict:
                    phredscore_dict[mean] += 1
```

```
                else:
                    phredscore_dict[mean] = 1
    with open("R" + str(n) + "_part2Plot.txt", "w") as output:
        for key in phredscore_dict:
            output.write(str(key) + "\t" + str(phredscore_dict[key])+"\n")

read_frequency(f1, True,1)
read_frequency(f2, True,2)
read_frequency(f3, True,3)
read_frequency(f4, True,4)
```

A shell script was created to assign the R1 and R2 for each of the two libraries as arguments in the above Python script. For the purpose of this script the 22_3H library R1 corresponded to `-f1` and R2 corresponded to `-f2` , the 32_4G library corresponded to `-f3` and `-f4` respectively.

The following shell script `part2FreqDist.srun` was submitted:

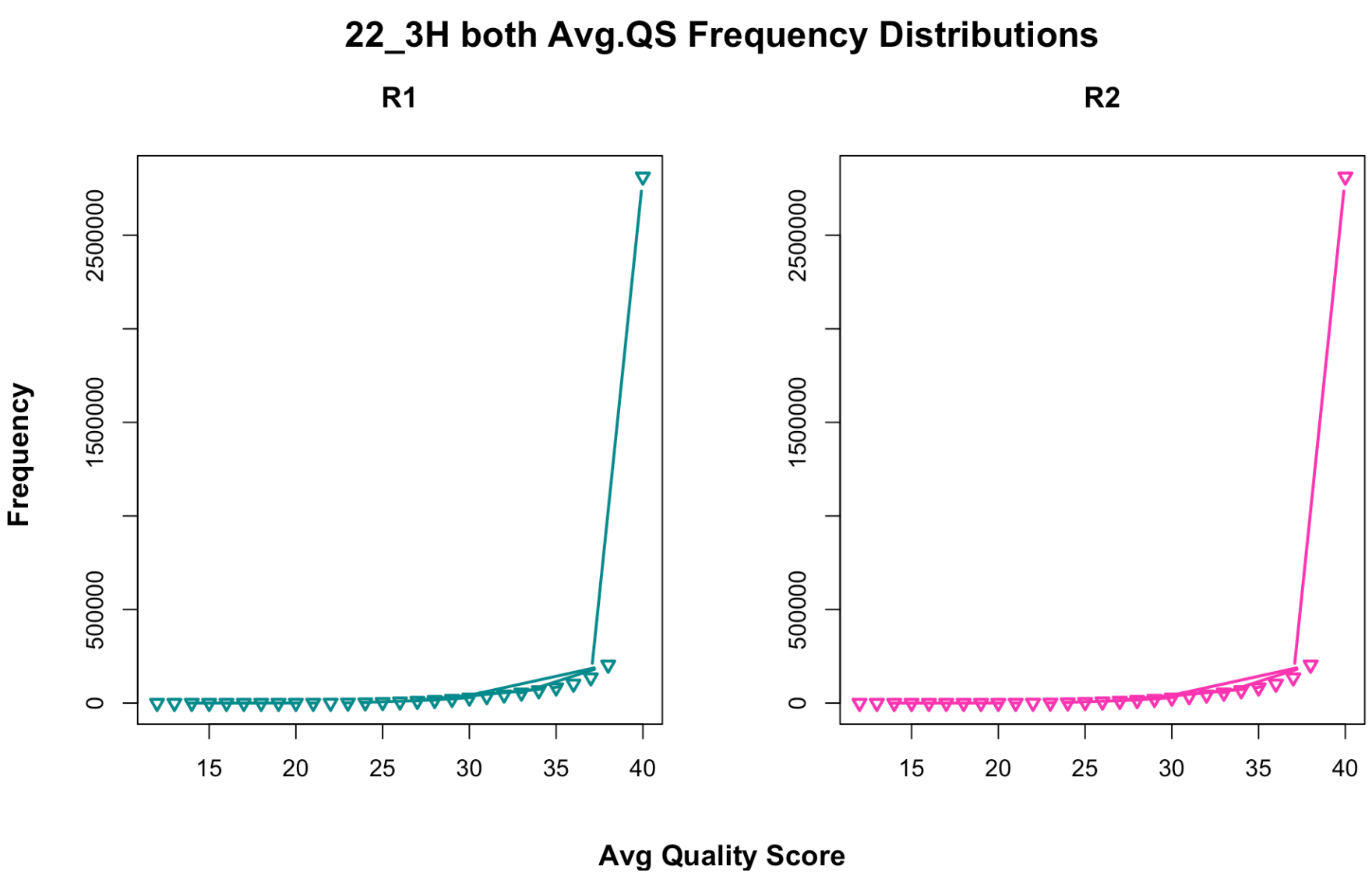
```
#!/bin/bash

#SBATCH --partition=long       ### Indicate want long node
#SBATCH --job-name=AD_freq2     ### Job Name
#SBATCH --output=AD_freq2.out         ### File in which to store job output
#SBATCH --error=AD_freq2.err          ### File in which to store job error messages
#SBATCH --time=2-00:00:00   ### Wall clock time limit in Days-HH:MM:SS
#SBATCH --nodes=1              ### Node count required for the job
#SBATCH --ntasks-per-node=28     ### Nuber of tasks to be launched per Node

ml easybuild GCC/6.3.0-2.27 OpenMPI/2.0.2 Python/3.6.1
pip list installed | grep numpy

python part2FreqDist.py -f1 22_3H_both_S16_L008_R1_001.fastq  -f2 22_3H_both_S16_L008
_R2_001.fastq  \
-f3 32_4G_both_S23_L008_R1_001.fastq -f4 32_4G_both_S23_L008_R2_001.fastq
```

The output files were renamed to be more descriptive of the forward and reverse reads for each of the two libraries

```
mv R1_part2Plot.txt R1_22_3H_part2Plot.tsv

mv R2_part2Plot.txt R2_22_3H_part2Plot.tsv

mv R3_part2Plot.txt R1_32_4G_part2Plot.tsv

mv R4_part2Plot.txt R2_32_4G_part2Plot.tsv
```

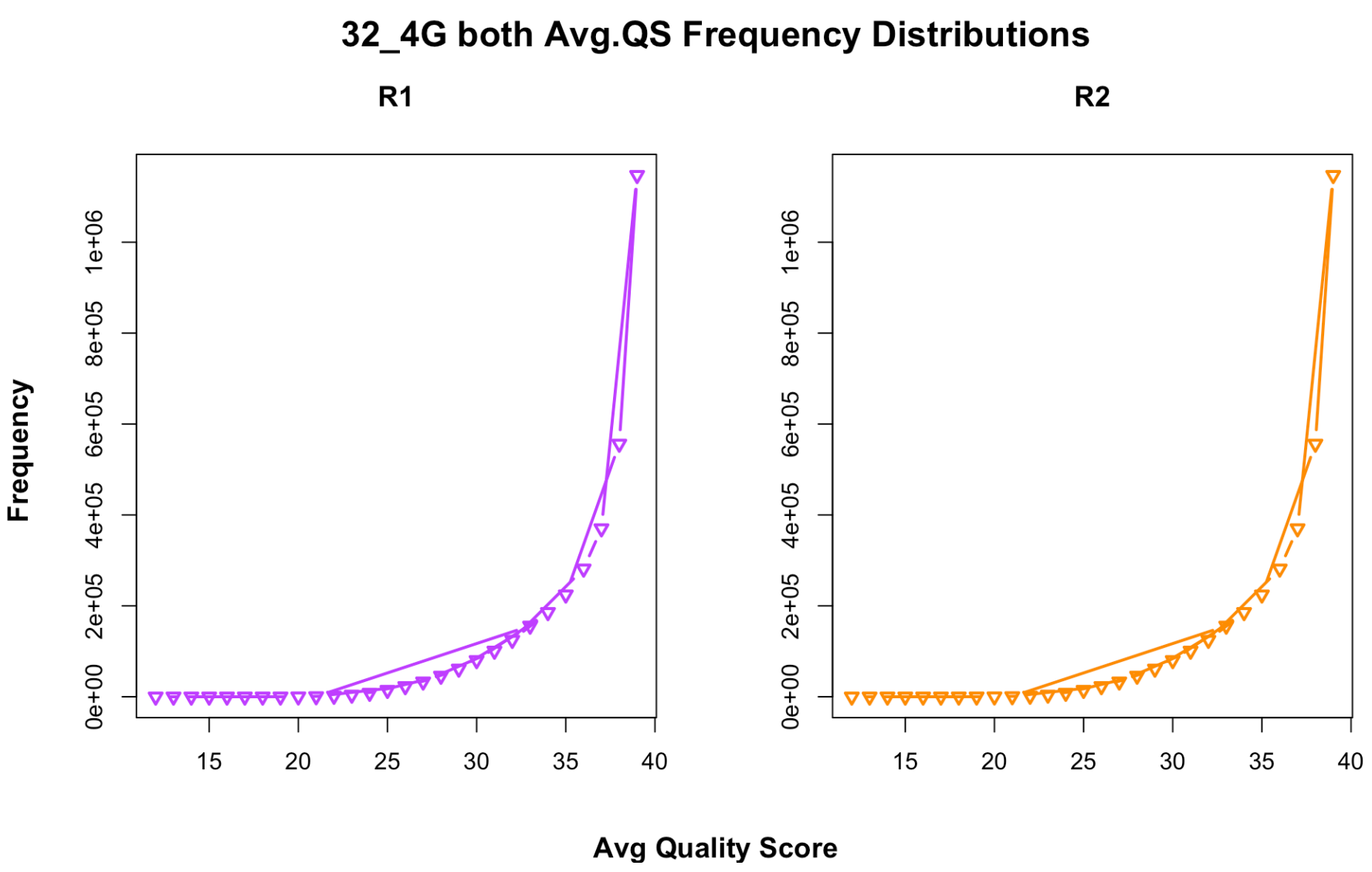
22_3H_both R1 and R2: Per sequence quality scores

```
R1_counts22 <- read.table("/Users/adowdell/sf-seq-qaa-alexadowdell/Part1/R1_22_3H_par
t2Plot.tsv", sep = "\t",
    header = TRUE)
R2_counts22 <- read.table("/Users/adowdell/sf-seq-qaa-alexadowdell/Part1/R1_22_3H_par
t2Plot.tsv", sep = "\t",
    header = TRUE)

headers <- c("QS", "Counts")  #assign column headers
colnames(R1_counts22) <- headers
colnames(R2_counts22) <- headers

par(mfrow = c(1, 2), oma = c(0, 1, 1, 0))
plot(R1_counts22$Counts ~ R1_counts22$QS, main = "R1", pch = 6, cex = 0.9, lwd = 2, c
ol = "cyan4", xlab = "",
    ylab = "", type = "b")

plot(R2_counts22$Counts ~ R2_counts22$QS, main = "R2", pch = 6, cex = 0.9, lwd = 2, c
ol = "maroon1",
    xlab = "", ylab = "", type = "b")

mtext("22_3H both Avg.QS Frequency Distributions", side = 3, font = 2, outer = TRUE,
line = -0.5, cex = 1.5)
mtext("Frequency", side = 2, font = 2, outer = TRUE, line = -0.5, cex = 1.2)
mtext("Avg Quality Score", side = 1, font = 2, outer = TRUE, line = -1, cex = 1.2)
```



32_4G_both R1 and R2: Per sequence quality scores

```
R1_counts32 <- read.table("/Users/adowdell/sf-seq-qaa-alexadowdell/Part1/R1_32_4G_par
t2Plot.tsv", sep = "\t",
    header = TRUE)
R2_counts32 <- read.table("/Users/adowdell/sf-seq-qaa-alexadowdell/Part1/R1_32_4G_par
t2Plot.tsv", sep = "\t",
    header = TRUE)

headers <- c("QS", "Counts")  #assign column headers
colnames(R1_counts32) <- headers
colnames(R2_counts32) <- headers

par(mfrow = c(1, 2), oma = c(0, 1, 1, 0))
plot(R1_counts32$Counts ~ R1_counts32$QS, main = "R1", pch = 6, cex = 0.9, lwd = 2, c
ol = "darkorchid1",
    xlab = "", ylab = "", type = "b")

plot(R2_counts32$Counts ~ R2_counts32$QS, main = "R2", pch = 6, cex = 0.9, lwd = 2, c
ol = "darkorange",
    xlab = "", ylab = "", type = "b")

mtext("32_4G both Avg.QS Frequency Distributions", side = 3, font = 2, outer = TRUE,
line = -0.5, cex = 1.5)
mtext("Frequency", side = 2, font = 2, outer = TRUE, line = -0.5, cex = 1.2)
mtext("Avg Quality Score", side = 1, font = 2, outer = TRUE, line = -1, cex = 1.2)
```



Overall, in both libraries for forward and reverse reads, the FastQC quality distribution plots and the ones personally generated with a handwritten Python script match very closely. In the per base sequence quality distributions, consistent average quality scores above 38 across majority of the basepairs are displayed, as well as the drop off in average quality score for approximately the first seven base positions. Therefore, the Python script ran to generate the quality score plots that was written specifically for the index hopping script has some validity since it agrees with the FastQC output.

Clearly, FastQC runs much faster than the Python scripts written personally. The FastQC returned output for each of the two libraries in less than 10 minutes. On the other hand the 22_3H_both library had a runtime of approximately 12 minutes using my personal Python script to generate the per sequence quality plot distributions. The 32_4G_both took approximately 37 minutes with the script to generate the same distribution. Additionally, a second Python script to generate the per sequence quality scores for all four distributions took 8 minutes to complete. The difference in runtime using the Python script between libraries can be attributed to the size the libraries to begin with. 22_3H_both library contained 4,050,899 reads as oppose to the 32_4G library which had 11,820,174 reads. The difference in runtime between FastQC and my personal Python scripts likely has a lot to do with the lack of optimization in my Python scripts. The FastQC program is likely much more efficient in its algorithms for generating the statistics and graphs. FastQC is also run with the underlying programming language being Java.

Part 2 – Adaptor trimming comparison

Look into the adaptor trimming options for cutadapt, process_shortreads, and Trimmomatic (all on Talapas), and briefly describe the differences. Pick one of these to properly trim adapter sequences. Use default settings. What proportion of reads (both forward and reverse) was trimmed?

cutadapt

- Only equipped to handle a single file
- Has options to cut read lengths, trim adapters, trim excess repetitive bases off the ends of reads
- Only takes a single adapter as a parameter, but there is an added parameter to accomodate Illumina Truseq adapters in paired end data
- Must specify two output file names corresponding to the paired end files being trimmed

process_shortreads

- Equipped to process paired end reads (two files) simultaneously
- Has quality filtering parameters to discard reads with low quality scores and/or reads with more mismatched bases than specified, or those that dont meet a specific length requirement.
- Various trimming parameters to remove bases instead of discarding reads and filter for the Illumina chastity filter utilizing a sliding window process.
- Only an output directory must be specified, outputs four files for paired end data, automatically named similarly to the input file names.
- Automatically discards reads with length <30 bp

Trimmomatic

- Similar to process_shortreads, equipped to take in two files simultaneously
- Various quality filtering parameters to discard reads completely if below a quality score cutoff or minimum length
- Various trimming parameters to not discard reads, but trim from the beginning or end of a read in the sliding window from the 5' end.
- Must specify four output file names explicitly for paired end reads, or provide a base name from all of the output file extensions to add to.

Before choosing one techinque, lets identify the known forward and reverse adapter sequences for our particular Illumina TruSeq reads:

Forward adapter (R1): GATCGGAAGAGCACACGTCTGAACTCCAGTCAC

Reverse adapter (R2): GATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT

Choosing one technique to trim adapter sequences: cutadapt

Two new shell scripts `cutadapt22.srun` and `cutadapt32.srun` were created to utilize the cutadapt adapter trimming technique. The scripts were very similar passing in the correct forward and reverse paired end files to the corresponding library. The appropriate modules for cutadapt, which were found using `module spider` are called and loaded. Then the `cutadapt` technique is called which is passed the known forward Illumina Truseq adapter sequence under `-a` and concatenation of the forward and reverse adapter sequences under `-A`. The `-o` parameter is passed the location of the output file for R1 with a name specified `trimmed_22_3H.R1.fastq` (in this case for the 22_3H library). `-p` corresponded to the location of the output file for the paired end R2 read with a specified name `trimmed_22_3H.R2.fastq`. Then the actual library paired end forward and reverse read files are passed in as additional parameters. Below are the `cutadapt22.srun` and `cutadapt32.srun` shell scripts that were run:

22_3H_both

```
#!/bin/bash
# SBATCH --partition=short         ### Indicate want long node
# SBATCH --job-name=AD_cut     ### Job Name
# SBATCH --output=AD_cut.out   ### File in which to store job output
# SBATCH --error=AD_cut.err    ### File in which to store job error messages
# SBATCH --time=0-12:00:00          ### Wall clock time limit in Days-HH:MM:SS
# SBATCH --nodes=1                ### Node count required for the job
# SBATCH --ntasks-per-node=28     ### Nuber of tasks to be launched per Node

# Load the proper module
ml easybuild ifort/2017.1.132-GCC-6.3.0-2.27 impi/2017.1.132
ml cutadapt/1.14-Python-2.7.13

cutadapt -a AGATCGGAAGAGCACACGTCTGAACTCCAGTCAC -A AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT \
-o /home/adowdell/AS1/cutadapt/cutadapt22-2/trimmed_22_3H.R1.fastq -p \
/home/adowdell/AS1/cutadapt/cutadapt22-2/trimmed_22_3H.R2.fastq /home/adowdell/AS1/22
_3H_both_S16_L008_R1_001.fastq \
/home/adowdell/AS1/22_3H_both_S16_L008_R2_001.fastq
```

33_4G_both

```
#!/bin/bash
# SBATCH --partition=short         ### Indicate want long node
# SBATCH --job-name=AD_cut2     ### Job Name
# SBATCH --output=AD_cut2.out   ### File in which to store job output
# SBATCH --error=AD_cut2.err    ### File in which to store job error messages
# SBATCH --time=0-12:00:00          ### Wall clock time limit in Days-HH:MM:SS
# SBATCH --nodes=1                ### Node count required for the job
# SBATCH --ntasks-per-node=28     ### Nuber of tasks to be launched per Node

# Load the proper module
ml easybuild ifort/2017.1.132-GCC-6.3.0-2.27 impi/2017.1.132
ml cutadapt/1.14-Python-2.7.13

cutadapt -a AGATCGGAAGAGCACACGTCTGAACTCCAGTCAC -A AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT \
-o /home/adowdell/AS1/cutadapt/cutadapt32-2/trimmed_32_4G.R1.fastq -p \
/home/adowdell/AS1/cutadapt/cutadapt32-2/trimmed_32_4G.R2.fastq /home/adowdell/AS1/32
_4G_both_S23_L008_R1_001.fastq \
/home/adowdell/AS1/32_4G_both_S23_L008_R2_001.fastq
```

Along with the two specified output files whose named were specified under the `-o` and `-p` parameters, a third `slurm-#.out` is outputted containing a summary of the run including statistics for total and discarded reads, as well as the distribution of counts at each read length of what trimming actions were conducted and how many reads were discarded. A quick view of the `slurm-#.out` which was renamed to reflect its corresponding libraries `slurm-22_3H.out` `slurm-32_4G.out` can be found below:

slurm-22_3H.out

```
Trimming 2 adapters with at most 10.0% errors in paired-end mode ...
Finished in 108.34 s (27 us/read; 2.24 M reads/minute).

=== Summary ===

Total read pairs processed:          4,050,899
  Read 1 with adapter:                 153,087 (3.8%)
  Read 2 with adapter:                 186,534 (4.6%)
Pairs written (passing filters):     4,050,899 (100.0%)

Total basepairs processed:   818,281,598 bp
  Read 1:   409,140,799 bp
  Read 2:   409,140,799 bp
Total written (filtered):    815,265,741 bp (99.6%)
  Read 1:   407,701,227 bp
  Read 2:   407,564,514 bp

=== First read: Adapter 1 ===

Sequence: AGATCGGAAGAGCACACGTCTGAACTCCAGTCAC; Type: regular 3'; Length: 34; Trimmed:
153087 times.

No. of allowed errors:
0-9 bp: 0; 10-19 bp: 1; 20-29 bp: 2; 30-34 bp: 3

Bases preceding removed adapters:
  A: 21.1%
  C: 30.8%
  G: 30.2%
  T: 17.7%
  none/other: 0.2%

Overview of removed sequences
length  count   expect  max.err error counts
3   75429   63295.3 0   75429
4   17444   15823.8 0   17444
5   6268    3956.0  0   6268
6   2911    989.0   0   2911
7   2584    247.2   0   2584
8   2382    61.8    0   2382
9   2434    15.5    0   2341 93
10  2338    3.9 1   2184 154
11  2260    1.0 1   2164 96
```

For the 22_3H_both library, the cutadapt technique trimmed 3.8% of the R1 reads that contained the forward adapter and 4.6% of the R2 reads containing the reverse adapter. These correspond to 153,087 reads in R1 and 186,534 reads in R2 out of the 4,050,899 total read pairs processed that were trimmed.

slurm-32_4G.out

```
Trimming 2 adapters with at most 10.0% errors in paired-end mode ...
Finished in 318.03 s (27 us/read; 2.23 M reads/minute).

=== Summary ===

Total read pairs processed:         11,820,174
  Read 1 with adapter:                 631,712 (5.3%)
  Read 2 with adapter:                 725,571 (6.1%)
Pairs written (passing filters):    11,820,174 (100.0%)

Total basepairs processed: 2,387,675,148 bp
  Read 1: 1,193,837,574 bp
  Read 2: 1,193,837,574 bp
Total written (filtered):  2,372,250,702 bp (99.4%)
  Read 1: 1,186,321,445 bp
  Read 2: 1,185,929,257 bp

=== First read: Adapter 1 ===

Sequence: AGATCGGAAGAGCACACGTCTGAACTCCAGTCAC; Type: regular 3'; Length: 34; Trimmed:
631712 times.

No. of allowed errors:
0-9 bp: 0; 10-19 bp: 1; 20-29 bp: 2; 30-34 bp: 3

Bases preceding removed adapters:
  A: 18.2%
  C: 32.1%
  G: 34.3%
  T: 14.4%
  none/other: 1.1%

Overview of removed sequences
length  count   expect  max.err error counts
3   228316  184690.2    0   228316
4   62985   46172.6 0   62985
5   30175   11543.1 0   30175
6   20018   2885.8  0   20018
7   18438   721.4   0   18438
8   17141   180.4   0   17141
9   16390   45.1    0   16058 332
10  15591   11.3    1   14990 601
11  14537   2.8 1   14071 466
```

For the 32_4G_both library, the cutadapt technique trimmed 5.3% of the R1 reads that contained the forward adapter and 6.1% of the R2 reads containing the reverse adapter. These correspond to 631,712 reads in R1 and 725,571 reads in R2 out of the 11,820,174 total read pairs processed that were trimmed.

Sanity check: Use your Unix skills to search for the adapter sequences in your datasets and confirm the expected sequence orientations.

22_3H_both

```
cat 22_3H_both_S16_L008_R1_001.fastq | grep "AGATCGGAAGAGCACACGTCTGAACTCCAGTCAC" | wc
-l

6857

cat 22_3H_both_S16_L008_R2_001.fastq | grep "GATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT" | wc -
l

8087
```

32_4G_both

```
cat 32_4G_both_S23_L008_R1_001.fastq | grep "AGATCGGAAGAGCACACGTCTGAACTCCAGTCAC" | wc
-l

33003

cat 32_4G_both_S23_L008_R2_001.fastq | grep "GATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT" | wc -
l

43839
```

Plot the trimmed read length distributions for both forward and reverse reads (on the same plot). If necessary, consult Assignment 5 (Block 1) from Bi 623 to refresh your memory.

Some Unix manipulation was done to the R1/R2 output files: `trimmed_22_3H.R1.fastq`, `trimmed_22_3H.R2.fastq`, `trimmed_32_4G.R1.fastq` and `trimmed_32_4G.R2.fastq` which contain the remaining reads: post-trimming. First, the files were narrowed down to only contain sequence lines. Then, the corresponding length of each sequence line was calculated and the frequency of each sequence length was recorded. What resulted were four new files `trimmed_cutadapt_22_3H.R1.tsv`, `trimmed_cutadapt_22_3H.R2.tsv`, `trimmed_cutadapt_32_4G.R1.tsv` and `trimmed_cutadapt_32_4G.R2.tsv` which contain two columns, counts and sequence length.

```
cat trimmed_22_3H.R1.fastq | grep -A 1 "^@" | grep -v "^--$" | grep -v "^@" | awk '{p
rint  length($0)}' | sort -n | uniq -c > trimmed_cutadapt_22_3H.R1.tsv

cat trimmed_22_3H.R2.fastq | grep -A 1 "^@" | grep -v "^--$" | grep -v "^@" | awk '{p
rint  length($0)}' | sort -n | uniq -c > trimmed_cutadapt_22_3H.R2.tsv

cat trimmed_32_4G.R1.fastq | grep -A 1 "^@" | grep -v "^--$" | grep -v "^@" | awk '{p
rint  length($0)}' | sort -n | uniq -c > trimmed_cutadapt_32_4G.R1.tsv

cat trimmed_32_4G.R2.fastq | grep -A 1 "^@" | grep -v "^--$" | grep -v "^@" | awk '{p
rint  length($0)}' | sort -n | uniq -c > trimmed_cutadapt_32_4G.R2.tsv
```

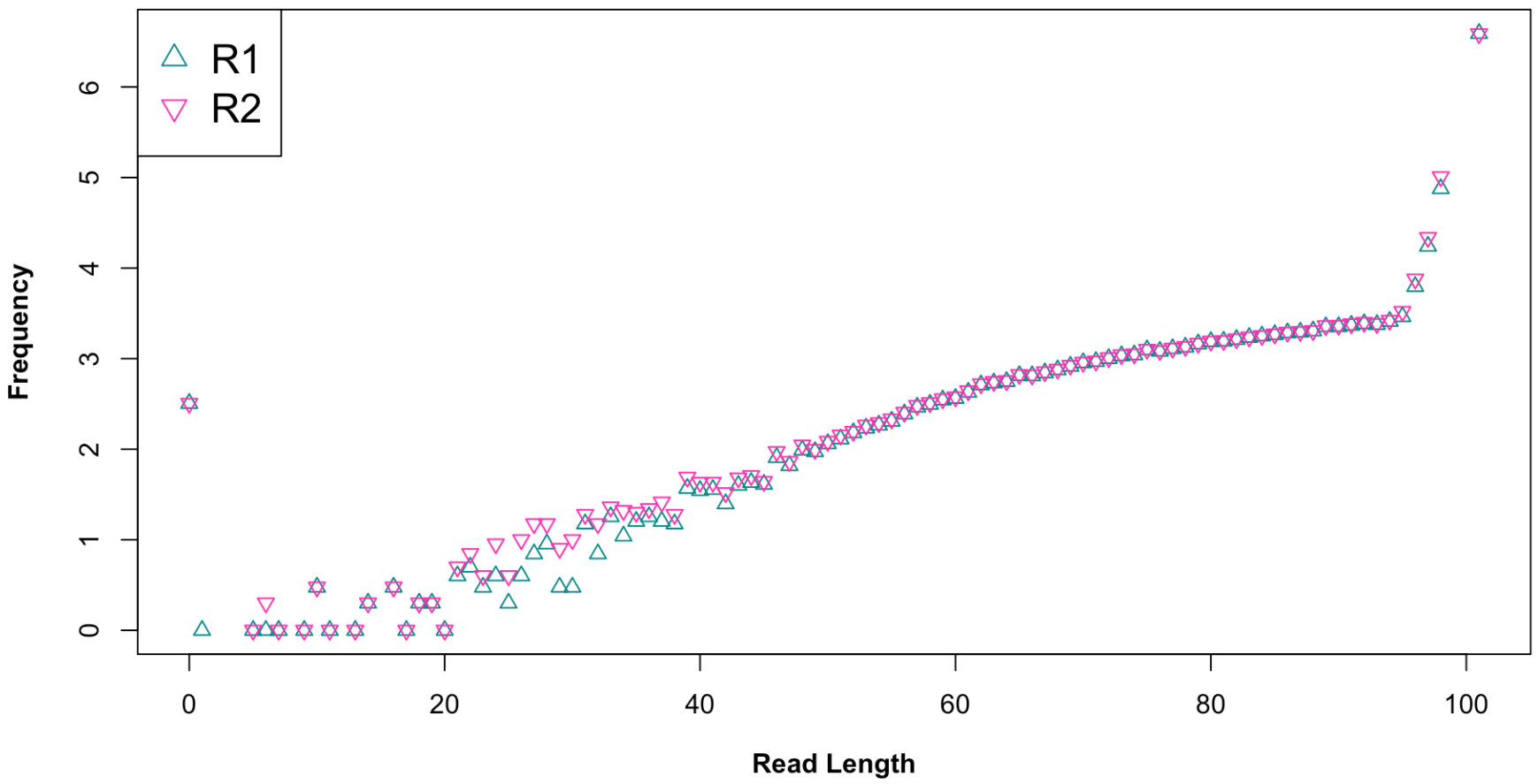
The four files `trimmed_cutadapt_22_3H.R1.tsv`, `trimmed_cutadapt_22_3H.R2.tsv`, `trimmed_cutadapt_32_4G.R1.tsv` and `trimmed_cutadapt_32_4G.R2.tsv` were slightly reformatted in TextWrangler to only contain one space as a seperator, and then read into R as data frames and assigned column headers `Frequency` and `Read_Length`.

```
R1_cutA_22 <- read.table("/Users/adowdell/sf-seq-qaa-alexadowdell/Part2/Cutadapt/trim
med_cutadapt_22_3H.R1.tsv",
    sep = " ", header = FALSE)
R2_cutA_22 <- read.table("/Users/adowdell/sf-seq-qaa-alexadowdell/Part2/Cutadapt/trim
med_cutadapt_22_3H.R2.tsv",
    sep = " ", header = FALSE)
R1_cutA_32 <- read.table("/Users/adowdell/sf-seq-qaa-alexadowdell/Part2/Cutadapt/trim
med_cutadapt_32_4G.R1.tsv",
    sep = " ", header = FALSE)
R2_cutA_32 <- read.table("/Users/adowdell/sf-seq-qaa-alexadowdell/Part2/Cutadapt/trim
med_cutadapt_32_4G.R2.tsv",
    sep = " ", header = FALSE)

column_headers <- c("Frequency", "Read_Length")
colnames(R1_cutA_22) <- column_headers
colnames(R2_cutA_22) <- column_headers
colnames(R1_cutA_32) <- column_headers
colnames(R2_cutA_32) <- column_headers
```

Using the imported distributions for R1 and R2 we now want to plot the distributions of trimmed read lengths for each of the two libraries 22_3H and 32_4G to visualize any dips that may appear in the graph.

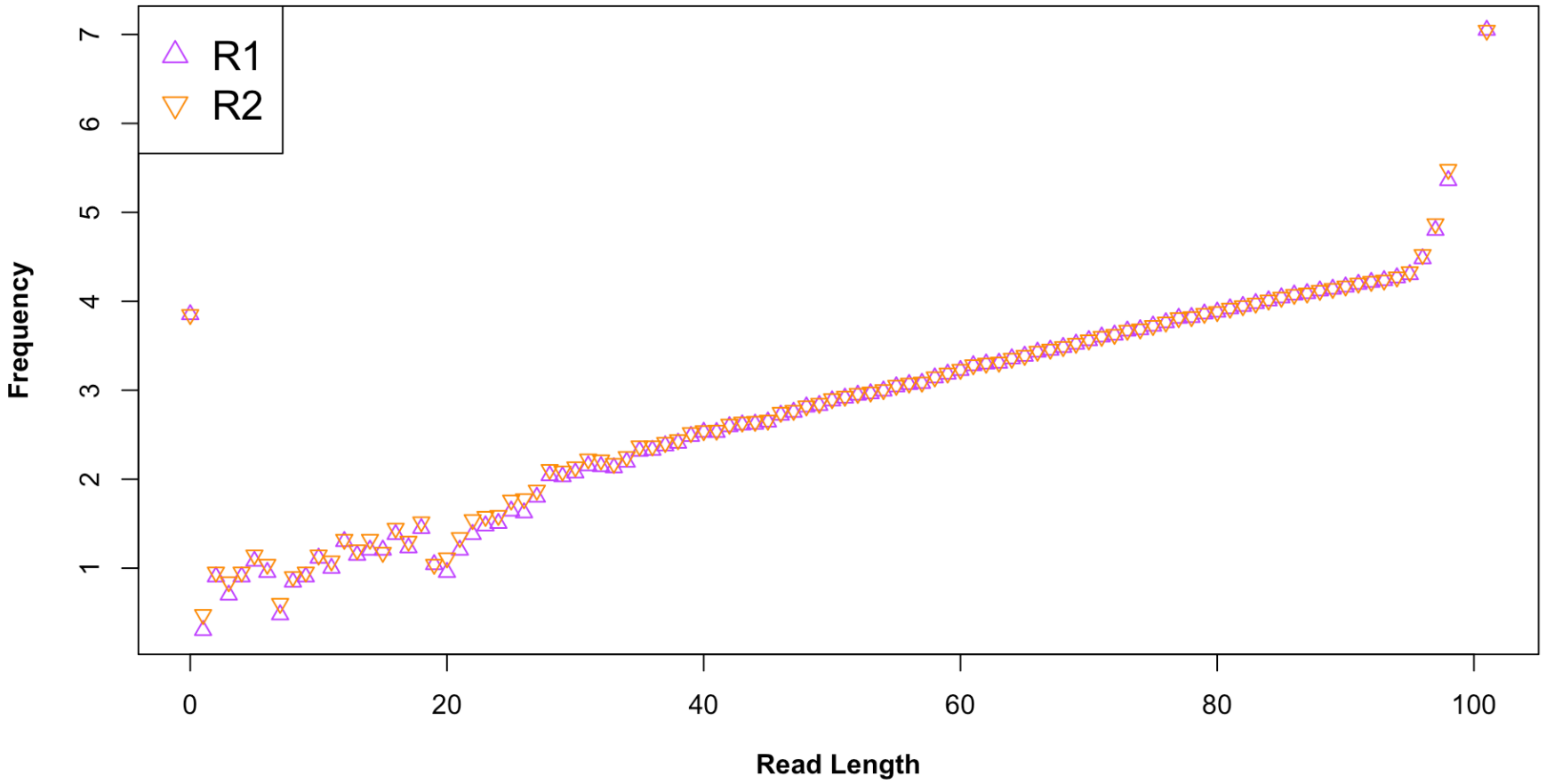
Distribution of Trimmed Cutadapt Reads: 22_3H_both

```
par(mfrow = c(1, 1), oma = c(0, 1, 1, 0))
plot(R1_cutA_22$Read_Length, log10(R1_cutA_22$Frequency), xlab = expression(bold("Rea
d Length")), ylab = expression(bold("Frequency")),
    col = "cyan4", pch = 2)
points(R2_cutA_22$Read_Length, log10(R2_cutA_22$Frequency), xlab = expression(bold("R
ead Length")), ylab = expression(bold("Frequency")),
    col = "maroon1", pch = 6)

mtext("22_3H_both Distribution of Trimmed Cutadapt Reads", side = 3, font = 2, outer
= TRUE, line = -1,
    cex = 1.7)

legend("topleft", c("R1", "R2"), pch = c(2, 6), col = c("cyan4", "maroon1"), cex = 1.
5)
```



Distribution of Trimmed Cutadapt Reads: 32_4G_both

```
par(mfrow = c(1, 1), oma = c(0, 1, 1, 0))
plot(R1_cutA_32$Read_Length, log10(R1_cutA_32$Frequency), xlab = expression(bold("Rea
d Length")), ylab = expression(bold("Frequency")),
    col = "darkorchid1", pch = 2)
points(R2_cutA_32$Read_Length, log10(R2_cutA_32$Frequency), xlab = expression(bold("R
ead Length")), ylab = expression(bold("Frequency")),
    col = "darkorange", pch = 6)

mtext("32_4G_both Distribution of Trimmed Cutadapt Reads", side = 3, font = 2, outer
= TRUE, line = -1,
    cex = 1.7)

legend("topleft", c("R1", "R2"), pch = c(2, 6), col = c("darkorchid1", "darkorange"),
cex = 1.5)
```



As a backup, and out of curiosity a more familiar technique `process_shortreads` was also run on both the 22_3H_both and 32_4G_both libraries. Unsurprisingly, the distributions of forward (r1) and reverse (R2) reads for both techniques graph identically. Differences were seen in the proportion of trimmed reads in each library of R1 and R2 reads between the `cutadapt` and `process_shortreads` techniques. These differences can be attributed to differences in the overall structure of trimming adapters noted in the beginning of this section two “Adapter Contamination.”

Choosing one technique to trim adapter sequences: process_shortreads

A new shell script was constructed to trim the adapter sequences, which loads Stacks 2.0 as well as all of the other associated modules. The `process_shortreads` command was called and the appropriate parameters were passed. `-1` and `-2` signify the files passed are paired end files R1 and R2. `-i` indicates input type is a fastq. `-y` specifies output format type as a fastq. `-o` is the output directory for the files created. `--adapter_mm` is the number of allowed mismatches before the read is discarded. Then `adapter_1` and `adapter_2` are the known Illumina adapter sequences appended to the reads. `-q` discards reads with low quality scores.

Seperate, but similar scripts `trimming22.srun` and `trimming32.srun` were created for the 22_3H and 32_4G libraries respectively.

The `trimming22.srun` script:

```
#!/bin/bash
# SBATCH --partition=gpu         ### Indicate want long node
# SBATCH --job-name=AD_ps22     ### Job Name
# SBATCH --output=AD_ps22.out   ### File in which to store job output
# SBATCH --error=AD_ps22.err    ### File in which to store job error messages
# SBATCH --time=0-05:00:00          ### Wall clock time limit in Days-HH:MM:SS
# SBATCH --nodes=1                ### Node count required for the job
# SBATCH --ntasks-per-node=28     ### Nuber of tasks to be launched per Node

# Load the proper module
ml slurm easybuild intel/2017a Stacks/1.46

process_shortreads -1 22_3H_both_S16_L008_R1_001.fastq -2 22_3H_both_S16_L008_R2_001.
fastq \
-i fastq  -y fastq -o ~/AS1/trim22 -c --adapter_mm 2 --adapter_1 AGATCGGAAGAGCACACGTC
TGAACTCCAGTCAC \
--adapter_2 AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT -q
```

The `trimming32.srun` script:

```
#!/bin/bash
# SBATCH --partition=gpu         ### Indicate want long node
# SBATCH --job-name=AD_ps32     ### Job Name
# SBATCH --output=AD_ps32.out   ### File in which to store job output
# SBATCH --error=AD_ps32.err    ### File in which to store job error messages
# SBATCH --time=0-05:00:00          ### Wall clock time limit in Days-HH:MM:SS
# SBATCH --nodes=1                ### Node count required for the job
# SBATCH --ntasks-per-node=28     ### Nuber of tasks to be launched per Node

# Load the proper module
ml slurm easybuild intel/2017a Stacks/1.46

process_shortreads -1 32_4G_both_S23_L008_R1_001.fastq -2 32_4G_both_S23_L008_R2_001.
fastq \
-i fastq  -y fastq -o ~/AS1/trim32 -c --adapter_mm 2 --adapter_1 AGATCGGAAGAGCACACGTC
TGAACTCCAGTCAC \
--adapter_2 AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT -q
```

Five files were outputted for each of the trimming scripts that called `process_shortreads` . For the 22_3H library the following files were created:

```
22_3H_both_S16_L008_R1_001.1.fq
22_3H_both_S16_L008_R1_001.fastq.rem.1.fq
22_3H_both_S16_L008_R2_001.2.fq
22_3H_both_S16_L008_R2_001.fastq.rem.2.fq
process_shortreads.log
```

The `process_shortreads.log` which was renamed to be `process_shortreads_22_3H.log` contains the summary from the run which is quite informative, as shown below:

```
process_shortreads -1 22_3H_both_S16_L008_R1_001.fastq -2 22_3H_both_S16_L008_R2_001.
fastq -i fastq -y fastq -o /home/adowdell/AS1/trim22 -c --adapter_mm 2 --adapter_1 AG
ATCGGAAGAGCACACGTCTGAACTCCAGTCAC --adapter_2 AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT -q
process_shortreads version 1.46 executed 2017-09-28 15:09:53

File    Retained Reads  Adapter Seq Low Quality Ambiguous Barcodes  Trimmed Reads   O
rphaned paired-end reads   Total
22_3H_both_S16_L008_R1_001.fastq    7959166 228522  66336   0   152226  0   8101798

Total Sequences 8101798
Reads containing adapter sequence   228522
Ambiguous Barcodes  0
Low Quality 66336
Trimmed Reads   152226
Orphaned Paired-ends    0
Retained Reads  7959166
```

Using the summary from `process_shortreads` the proportion of trimmed reads can be calculated.

$$152226/8101798 = 0.01878$$

Therefore, 0.018789 is the proportion of trimmed reads in the 22_3H library, or 1.88% of the total reads were trimmed in the library.

Likewise, a `process_shortreads.log` which was renamed to be `process_shortreads_32_4G.log` was generated from the `trimming32.srun` containing the summary of trimmed adapter sequence from the 32_4G library:

```
process_shortreads -1 32_4G_both_S23_L008_R1_001.fastq -2 32_4G_both_S23_L008_R2_001.
fastq -i fastq -y fastq -o /home/adowdell/AS1/trim32 -c --adapter_mm 2 --adapter_1 AG
ATCGGAAGAGCACACGTCTGAACTCCAGTCAC --adapter_2 AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT -q
process_shortreads version 1.46 executed 2017-09-28 15:11:43

File    Retained Reads  Adapter Seq Low Quality Ambiguous Barcodes  Trimmed Reads   O
rphaned paired-end reads   Total
32_4G_both_S23_L008_R1_001.fastq    23231813    967412  197019  0   755896  0   23640
348

Total Sequences 23640348
Reads containing adapter sequence   967412
Ambiguous Barcodes  0
Low Quality 197019
Trimmed Reads   755896
Orphaned Paired-ends    0
Retained Reads  23231813
```

Using the summary from `process_shortreads` the proportion of trimmed reads can be calculated.

$$755896/23640348 = 0.03197$$

Therefore, 0.03197 is the proportion of trimmed reads in the 32_4G library, or 3.19% of the total reads were trimmed in the library.

Plot the trimmed read length distributions for both forward and reverse reads (on the same plot). If necessary, consult Assignment 5 (Block 1) from Bi 623 to refresh your memory.

A bit of Unix manipulation was done to the R1/R2 output files: `22_3H_both_S16_L008_R1_001.1.fq`, `22_3H_both_S16_L008_R2_001.2.fq`, `32_4G_both_S23_L008_R1_001.1.fq` and `32_4G_both_S23_L008_R2_001.2.fq` which contain the remaining reads: post-trimming. Four new files were created `22_3H_both_R1_trimmed_dist.tsv`, `22_3H_both_R2_trimmed_dist.tsv`, `32_4G_both_R1_trimmed_dist.tsv` and `32_4G_both_R2_trimmed_dist.tsv` which contain two columns, counts and sequence length.

```
cat 22_3H_both_S16_L008_R1_001.1.fq | grep -A 1 "^@" | grep -v "^--$" | grep -v "^@"
| awk '{print  length($0)}' | sort -n | uniq -c > 22_3H_both_R1_trimmed_dist.tsv

cat 22_3H_both_S16_L008_R2_001.2.fq | grep -A 1 "^@" | grep -v "^--$" | grep -v "^@"
| awk '{print  length($0)}' | sort -n | uniq -c > 22_3H_both_R2_trimmed_dist.tsv

cat 32_4G_both_S23_L008_R1_001.1.fq | grep -A 1 "^@" | grep -v "^--$" | grep -v "^@"
| awk '{print  length($0)}' | sort -n | uniq -c > 32_4G_both_R1_trimmed_dist.tsv

cat 32_4G_both_S23_L008_R2_001.2.fq | grep -A 1 "^@" | grep -v "^--$" | grep -v "^@"
| awk '{print  length($0)}' | sort -n | uniq -c > 32_4G_both_R2_trimmed_dist.tsv
```

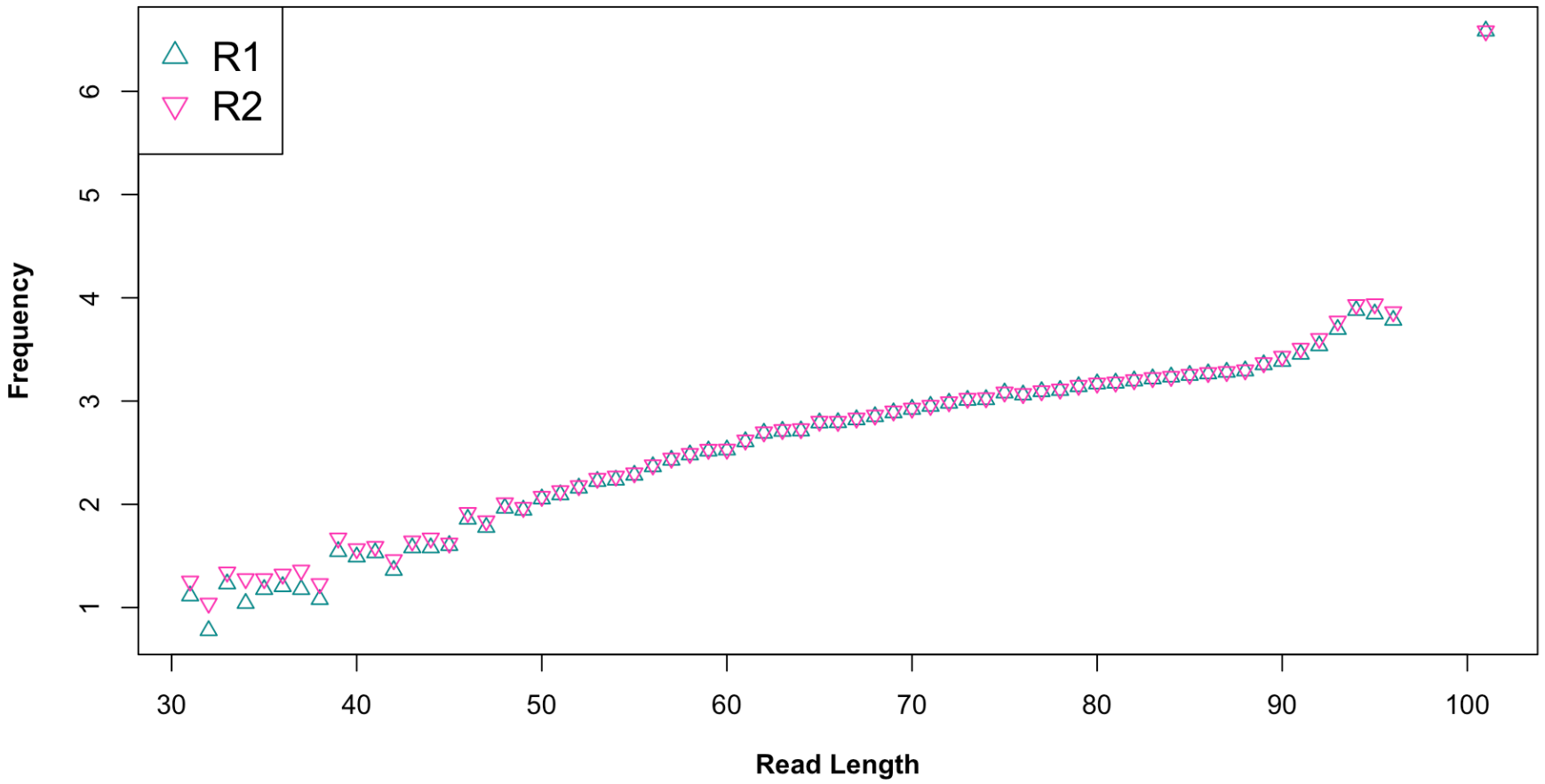
The four files `22_3H_both_R1_trimmed_dist.tsv`, `22_3H_both_R2_trimmed_dist.tsv`, `32_4G_both_R1_trimmed_dist.tsv` and `32_4G_both_R2_trimmed_dist.tsv` were read into R as data frames and assigned column headers `Frequency` and `Read_Length`.

```
R1_trimmed_22 <- read.table("/Users/adowdell/sf-seq-qaa-alexadowdell/Part2/Process_sh
ortreads/22_3H_both_R1_trimmed_dist.tsv",
    sep = " ", header = FALSE)
R2_trimmed_22 <- read.table("/Users/adowdell/sf-seq-qaa-alexadowdell/Part2/Process_sh
ortreads/22_3H_both_R2_trimmed_dist.tsv",
    sep = " ", header = FALSE)
R1_trimmed_32 <- read.table("/Users/adowdell/sf-seq-qaa-alexadowdell/Part2/Process_sh
ortreads/32_4G_both_R1_trimmed_dist.tsv",
    sep = " ", header = FALSE)
R2_trimmed_32 <- read.table("/Users/adowdell/sf-seq-qaa-alexadowdell/Part2/Process_sh
ortreads/32_4G_both_R2_trimmed_dist.tsv",
    sep = " ", header = FALSE)

column_headers <- c("Frequency", "Read_Length")
colnames(R1_trimmed_22) <- column_headers
colnames(R2_trimmed_22) <- column_headers
colnames(R1_trimmed_32) <- column_headers
colnames(R2_trimmed_32) <- column_headers
```

Using the imported distributions for R1 and R2 we now want to plot the distributions of trimmed read lengths for each of the two libraries 22_3H and 32_4G to visualize any dips that may appear in the graph.

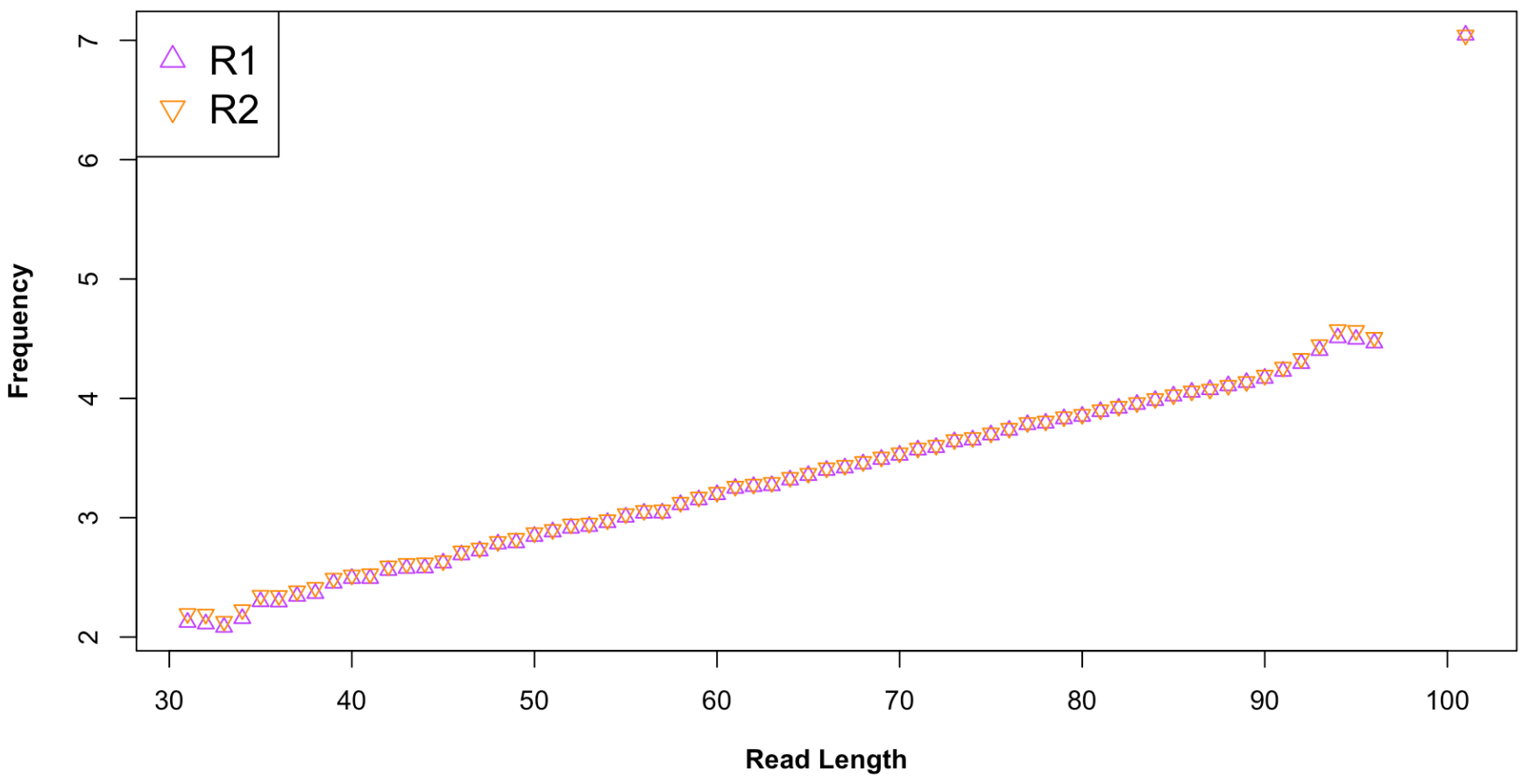
Distribution of Trimmed Process_shortreads: 22_3H_both

```
par(mfrow = c(1, 1), oma = c(0, 1, 1, 0))
plot(R1_trimmed_22$Read_Length, log10(R1_trimmed_22$Frequency), xlab = expression(bol
d("Read Length")),
    ylab = expression(bold("Frequency")), col = "cyan4", pch = 2)
points(R2_trimmed_22$Read_Length, log10(R2_trimmed_22$Frequency), xlab = expression(b
old("Read Length")),
    ylab = expression(bold("Frequency")), col = "maroon1", pch = 6)

mtext("22_3H_both Distribution of Trimmed Process_shortreads", side = 3, font = 2, ou
ter = TRUE, line = -1,
    cex = 1.7)

legend("topleft", c("R1", "R2"), pch = c(2, 6), col = c("cyan4", "maroon1"), cex = 1.
5)
```



Distribution of Trimmed Process_shortreads: 32_4G_both

```
par(mfrow = c(1, 1), oma = c(0, 1, 1, 0))
plot(R1_trimmed_32$Read_Length, log10(R1_trimmed_32$Frequency), xlab = expression(bol
d("Read Length")),
    ylab = expression(bold("Frequency")), col = "darkorchid1", pch = 2)
points(R2_trimmed_32$Read_Length, log10(R2_trimmed_32$Frequency), xlab = expression(b
old("Read Length")),
    ylab = expression(bold("Frequency")), col = "darkorange", pch = 6)

mtext("32_4G_both Distribution of Trimmed Process_shortreads", side = 3, font = 2, ou
ter = TRUE, line = -1,
    cex = 1.7)

legend("topleft", c("R1", "R2"), pch = c(2, 6), col = c("darkorchid1", "darkorange"),
cex = 1.5)
```



Briefly describe whether the adaptor trimming results are consistent with the insert size distributions for your libraries. The size distribution information is in the Fragment Analyzer trace file on Github.

The adapter trimming results are consistent with the insert size distributions for libraries 22_3H_both and 32_4G_both. For between 50 bp and 1000 bp the average read lengths for libraries 22_3H_both and 32_4G_both were 411 bp and 420 bp, respectively. Since 411 and 420 bp are both average read lengths significantly larger than the specified insert length of 100, we dont expect to see much adapter contamination which is consistent with our results. The probability of sequencing adapter and having adapter contamination is very unusual given an average read length approximately 300 bp longer than our insert length to buffer sequencing error before an adapter is unexpectedly sequenced.

Part 3 – rRNA reads and strand-specificity

Find publicly available mouse rRNA sequences and generate a gsnap database from them. Align the SF-Seq reads to your mouse rRNA database and report the proportion of reads that likely came from rRNAs.

The mouse rRNA sequences were not explicitly available on Ensembl. The mouse non coding RNA fasta was downloaded `Mus_musculus.GRCm38.ncrna.fa.gz` as a starting point. The file was gunzipped and two functions which I personally added to the `~/bin` directory and exported the PATH in my `.bash_profile` were used. `FastaToTbl` converts a fasta file format to a tab delimited format where each record is transformed into one line (sequence_namesequence). Then, the rRNA sequences could be grepped by `transcript_biotype: rRNA`. The file was then formatted back into normal fasta format using `TblToFasta` and the rRNA sequences were finally outputted as `Mus_musculus.GRCm38.rRNAonly.fa`. The rRNA only fasta was gzipped in preparation for gmapping.

```
gunzip Mus_musculus.GRCm38.ncrna.fa.gz

FastaToTbl Mus_musculus.GRCm38.ncrna.reformat.fa | grep "transcript_biotype: rRNA" |
TblToFasta > Mus_musculus.GRCm38.rRNAonly.fa

gzip Mus_musculus.GRCm38.rRNAonly.fa
```

Below are the commands underlying both `FastaToTbl` and `TblToFasta`:

`FastaToTbl` command:

```
#!/usr/bin/awk -f
{
        if (substr($1,1,1)==">")
        if (NR>1)
                    printf "\n%s\t", substr($0,2,length($0)-1)
        else
            printf "%s\t", substr($0,2,length($0)-1)
        else
                printf "%s", $0
}END{printf "\n"}
```

`TblToFasta` command:

```
#! /usr/bin/awk -f
{
  sequence=$NF

  ls = length(sequence)
  is = 1
  fld  = 1
  while (fld < NF)
  {
     if (fld == 1){printf ">"}
     printf "%s " , $fld
     if (fld == NF-1){
        printf "\n"
      }
      fld = fld+1
  }
  while (is <= ls){
    printf "%s\n", substr(sequence,is,60)
    is=is+60
  }
}
```

A new shell script was made called `mouseDB.srun` which loads the gmap-gsnap modules and executes the `gmap_build` command:

```
#!/bin/bash

#SBATCH --partition=gpu        ### Indicate want long node
#SBATCH --job-name=AD_GMAP     ### Job Name
#SBATCH --output=AD_mouseDB.out         ### File in which to store job output
#SBATCH --error=AD_mouseDB.err          ### File in which to store job error messages
#SBATCH --time=0-02:00:00   ### Wall clock time limit in Days-HH:MM:SS
#SBATCH --nodes=1               ### Node count required for the job
#SBATCH --ntasks-per-node=28     ### Nuber of tasks to be launched per Node

module use /projects/ebb/modules/all/Core
module load gmap-gsnap/2017-09-11
gmap_build -D ~/AS1/Part3 -d mouseDB -g Mus_musculus.GRCm38.rRNAonly.fa.gz
```

After the rRNA database `mouseDB` was created, two similar shell scripts were constructed, one for each of the libraries `mouseGsnap22trim.srun` and `mouseGsnap32trim.srun` . Each script takes in the corresponding trimmed library files that were outputted during the `cutadapt` adapter trimming process. Its important to note the `--allow-pe-name-mismatch` is used to prevent mismatching errors to fail the jobs. `-N 1` should be noted as important in marking nonsplicing sites for RNA seq data. Forgetting the `-N` parameter in RNA seq data makes gmap/gsnap default to thinking its DNA seq data which is incorrect.

The scripts look like the following:

`mouseGsnap22trim.srun`

```
#!/bin/bash
#SBATCH --partition=short        ### Indicate want long node
#SBATCH --job-name=AD_22bt     ### Job Name
#SBATCH --output=AD_22bt.out         ### File in which to store job output
#SBATCH --error=AD_22bt.err          ### File in which to store job error messages
#SBATCH --time=0-24:00:00   ### Wall clock time limit in Days-HH:MM:SS
#SBATCH --nodes=1                ### Node count required for the job
#SBATCH --ntasks-per-node=28     ### Nuber of tasks to be launched per Node

module use /projects/ebb/modules/all/Core
module load gmap-gsnap/2017-09-11

gsnap -D ~/AS1/Part3 -d mouseDB -t 26 --allow-pe-name-mismatch --no-sam-headers \
--ordered -N 1 -A sam ~/AS1/cutadapt/cutadapt22-2/trimmed_22_3H.R1.fastq \
~/AS1/cutadapt/cutadapt22-2/trimmed_22_3H.R2.fastq --split-output \
~/AS1/Part3-Trimmed/22_3H_R1R2_gsnapTrim.sam
```

mouseGsnap32trim.srun

```
#!/bin/bash
#SBATCH --partition=short        ### Indicate want long node
#SBATCH --job-name=AD_32bt     ### Job Name
#SBATCH --output=AD_32bt.out         ### File in which to store job output
#SBATCH --error=AD_32bt.err          ### File in which to store job error messages
#SBATCH --time=0-24:00:00   ### Wall clock time limit in Days-HH:MM:SS
#SBATCH --nodes=1                ### Node count required for the job
#SBATCH --ntasks-per-node=28     ### Nuber of tasks to be launched per Node

module use /projects/ebb/modules/all/Core
module load gmap-gsnap/2017-09-11

gsnap -D ~/AS1/Part3 -d mouseDB -t 26 --allow-pe-name-mismatch --no-sam-headers \
--ordered -N 1 -A sam ~/AS1/cutadapt/cutadapt32-2/trimmed_32_4G.R1.fastq \
~/AS1/cutadapt/cutadapt32-2/trimmed_32_4G.R2.fastq --split-output \
~/AS1/Part3-Trimmed/32_4G_R1R2_gsnapTrim.sam
```

The output files with the prefixes `22_3H_R1R2_gsnapTrim.sam` and `32_4G_R1R2_gsnapTrim.sam` were directed towards the `Part3-Trimmed` directory. Since the proportion of rRNA reads is of interest to us, the `.nomapping` file corresponding to each library is of most interest. We can determine the proportion of rRNA reads by finding the number of reads that did not map to the mouse rRNA database out of the total number of reads for the particular library and subtracting that fraction from 1.

22_3H_both

```
(wc -l 22_3H_both_S16_L008_R1_001.fastq) / 4 = 4050899
(wc -l 22_3H_both_S16_L008_R2_001.fastq) / 4 = 4050899

grep -v "^@" 22_3H_R1R2_gsnapTrim.sam.nomapping | wc -l
8081907
```

$$1 - (8081907/(4050899 + 4050899))$$

$$1 - 0.9975 = 0.0025$$

The 22_3H_both library has a total of 8101798 reads, in which 8081907 reads did not map to rRNA sequences.Therefore, 0.25% of reads mapped to rRNA sequences.

32_4G_both

```
wc -l 32_4G_both_S23_L008_R1_001.fastq / 4 = 11820174
wc -l 32_4G_both_S23_L008_R1_001.fastq / 4 = 11820174

grep -v "^@" 32_4G_R1R2_gsnapTrim.sam.nomapping | wc -l
23373693
```

$$1 - (23373693/(11820174 + 11820174))$$

$$1 - 0.9887 = 0.0113$$

The 32_4G_both library has a total of 23640348 reads, in which 23373693 reads did not map. Therefore, 1.13% of reads mapped to rRNA sequences.

Demonstrate convincingly that the SF-Seq data are from “strand-specific” RNA-Seq libraries. There are a number of possible strategies to address this problem, but you need only implement one. Report your evidence in numeric and graphical (e.g. a plot) forms.

To confirm the SF-Seq data are from “strand-specific” RNA-Seq libraries is to identify patterns of poly-A tail homopolymers. Since the libraries were prepped using poly-A tail selection to select mRNA sequences with poly-A tails, we expect to see reads in either the R1 or R2 file containing an abundance of A’s. Likewise, the read file for the same library that does not contain an abundance of reads with poly-A tailing will contain an abundance of poly-T sequences.

First we’ll look at the 22_3H_both library for poly-A and poly-T tail homopolymers.

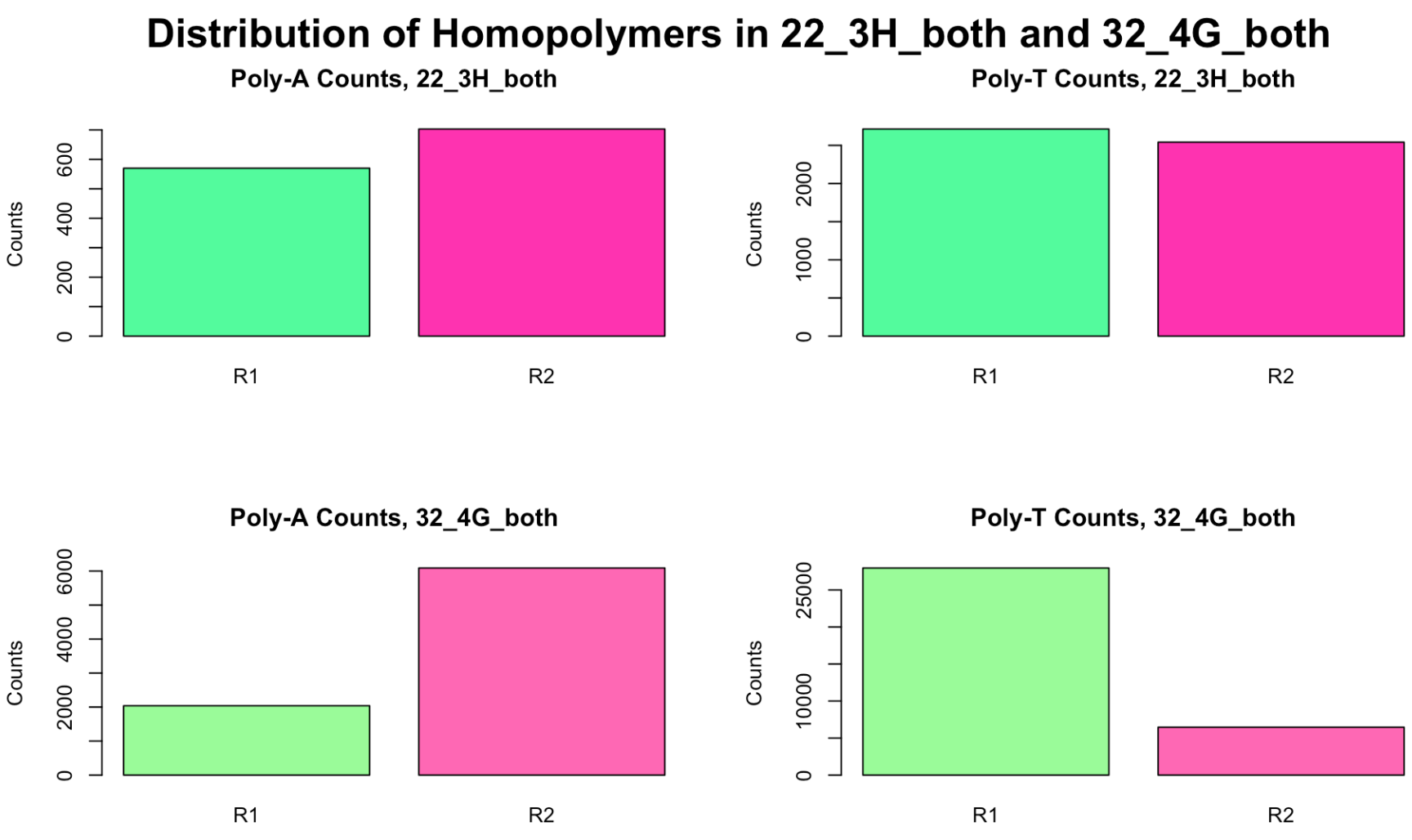
```
grep -A 1 "^@K00" trimmed_22_3H.R1.fastq | grep "AAAAAAAAAAAAAAAAAAAAAAAAA" | wc -l
570

grep -A 1 "^@K00" trimmed_22_3H.R2.fastq | grep "AAAAAAAAAAAAAAAAAAAAAAAAA" | wc -l
703

grep -A 1 "^@K00" trimmed_22_3H.R1.fastq | grep "TTTTTTTTTTTTTTTTTTTTTTTTT" | wc -l
2714

grep -A 1 "^@K00" trimmed_22_3H.R2.fastq | grep "TTTTTTTTTTTTTTTTTTTTTTTTT" | wc -l
2542
```

Then, the same process was conducted for the 32_4G_both library.

```
grep -A 1 "^@K00" trimmed_32_4G.R1.fastq | grep "AAAAAAAAAAAAAAAAAAAAAAAAA" | wc -l
2039

grep -A 1 "^@K00" trimmed_32_4G.R2.fastq | grep "AAAAAAAAAAAAAAAAAAAAAAAAA" | wc -l
6090

grep -A 1 "^@K00" trimmed_32_4G.R1.fastq | grep "TTTTTTTTTTTTTTTTTTTTTTTTT" | wc -l
27965

grep -A 1 "^@K00" trimmed_32_4G.R2.fastq | grep "TTTTTTTTTTTTTTTTTTTTTTTTT" | wc -l
6462
```

The counts for both libraries were added into R in the form of lists for comparison.

```
polyA_count22 <- c(570, 703)
polyT_count22 <- c(2714, 2542)
counts223H <- as.data.frame(cbind(polyA_count22, polyT_count22))

polyA_count32 <- c(2039, 6090)
polyT_count32 <- c(27965, 6462)
counts324G <- as.data.frame(cbind(polyA_count32, polyT_count32))


par(mfrow = c(2, 2), oma = c(0, 1, 1, 0))
barplot(counts223H$polyA_count22, col = c("seagreen1", "maroon1"), main = "Poly-A Cou
nts, 22_3H_both",
    name = c("R1", "R2"), ylab = "Counts")
barplot(counts223H$polyT_count22, col = c("seagreen1", "maroon1"), main = "Poly-T Cou
nts, 22_3H_both",
    name = c("R1", "R2"), ylab = "Counts")
barplot(counts324G$polyA_count32, col = c("palegreen1", "hotpink"), main = "Poly-A Co
unts, 32_4G_both",
    name = c("R1", "R2"), ylab = "Counts")
barplot(counts324G$polyT_count32, col = c("palegreen1", "hotpink"), main = "Poly-T Co
unts, 32_4G_both",
    name = c("R1", "R2"), ylab = "Counts")
mtext("Distribution of Homopolymers in 22_3H_both and 32_4G_both", side = 3, font = 2
, outer = TRUE,
    line = -1, cex = 1.6)
```



According to the counts for poly-A and poly-T homopolymers in each of the libraries and from the visual representations above, there are patterns regarding the abundance of each type of homopolymer based on whether the file was a forward (R1) or reverse (R2) read. Both the 22_3H_both and 32_4G_both libraries show trends of an increased abundance of poly-As in R2 than R1, and furthermore an increased abundance of poly-Ts in R1 than R2. Although the trend is much more apparent in the 32_4G_both library, the patterns remain consistent and differences can be attributed to a number of things including library size, and differing individual library prep.