RNA-seq Quality Assessment

Zach Sisson

9/5/2021

Part 1 - Read Quality Score Distributions

Using FastQC to generate distribution plots of the quality scores of the R1 and R2 reads.

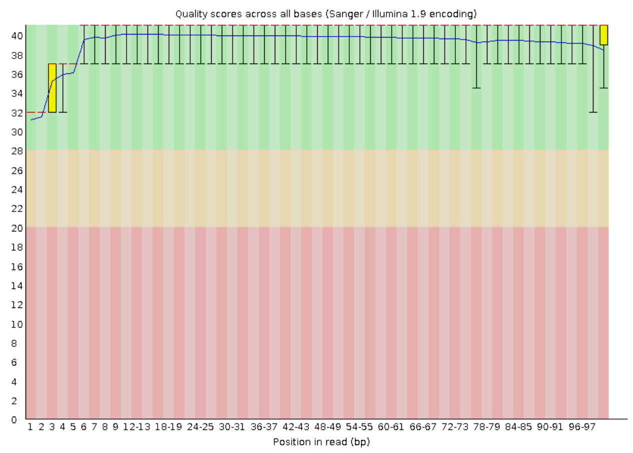
Using the following command, FastQC plots were generated to display the distribution of both mean quality scores per base, as well as distributions of the mean 'N' content per base.

```
cat FastQC.wrapper.sh
```

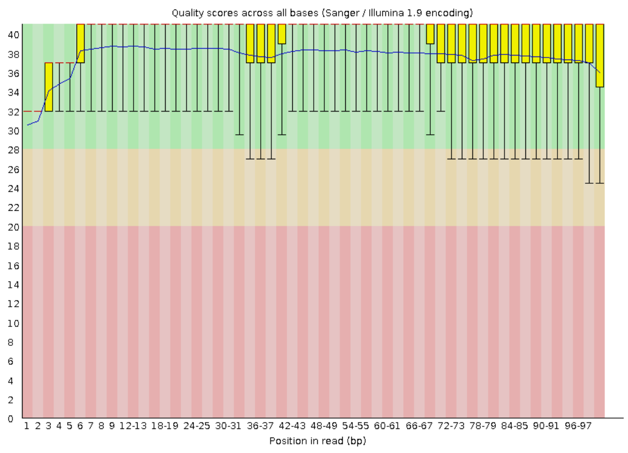
```
## #!/usr/bin/bash
## #SBATCH --account=bgmp
## #SBATCH --partition=bgmp
## #SBATCH --nodes=1
## #SBATCH --ntasks-per-node=1
## #SBATCH --cpus-per-task=1
## #SBATCH --time=0-20:00:00
## #SBATCH --output=FastQC.output.%j
## #SBATCH --error=FastQC.output.err
##
##
## #Files I am using: 23_4A_control_S17_L008 and 22_3H_both_S16_L008
## file1read1='/projects/bgmp/shared/2017_sequencing/demultiplexed/23_4A_control_S17_L008_R1_001.fastq.g
## file1read2='/projects/bgmp/shared/2017_sequencing/demultiplexed/23_4A_control_S17_L008_R2_001.fastq.g
## file2read1='/projects/bgmp/shared/2017_sequencing/demultiplexed/22_3H_both_S16_L008_R1_001.fastq.gz'
## file2read2='/projects/bgmp/shared/2017_sequencing/demultiplexed/22_3H_both_S16_L008_R2_001.fastq.gz'
##
## ml fastqc/0.11.5
##
## /usr/bin/time -v fastqc $file1read1 $file1read2 $file2read1 $file2read2 \
## -o '/home/zsisson2/bgmp/bioinformatics/Bi623/QAA/fastqc_output'
```

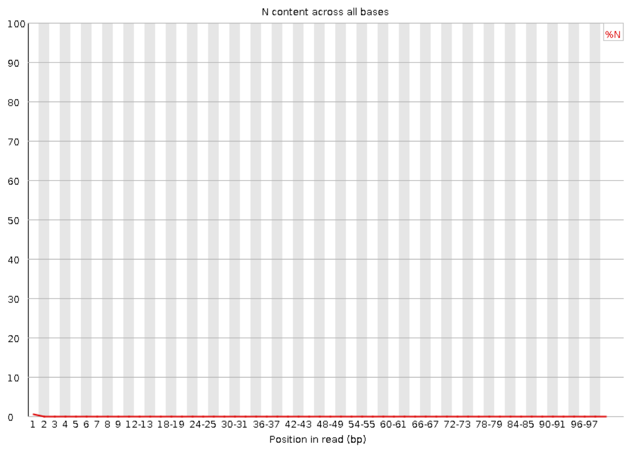
Plots for 22_3H_both_S16

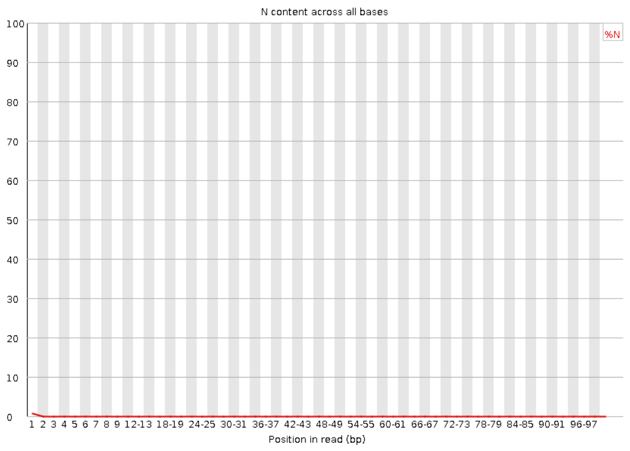
Graph order: Top two represent mean quality score distribution and the bottom two represent mean 'N' content per base. Left two represent Read 1 (forward) and



right two represent Read 2 (reverse).

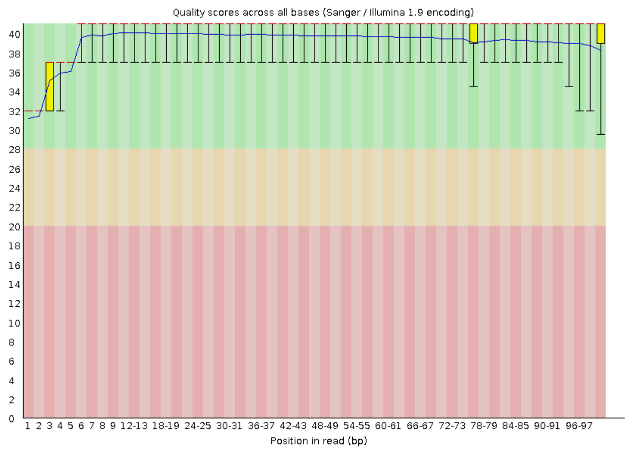


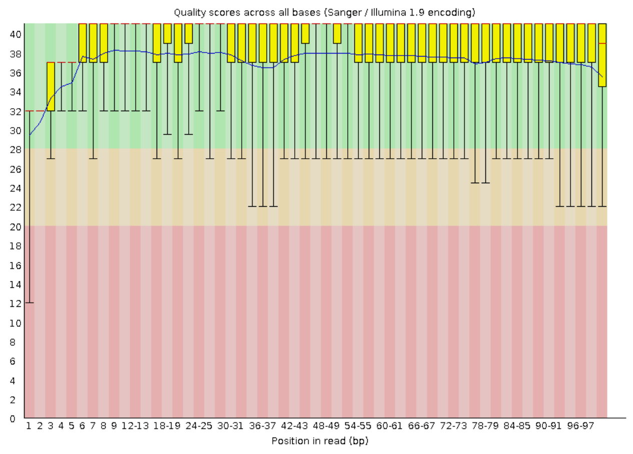


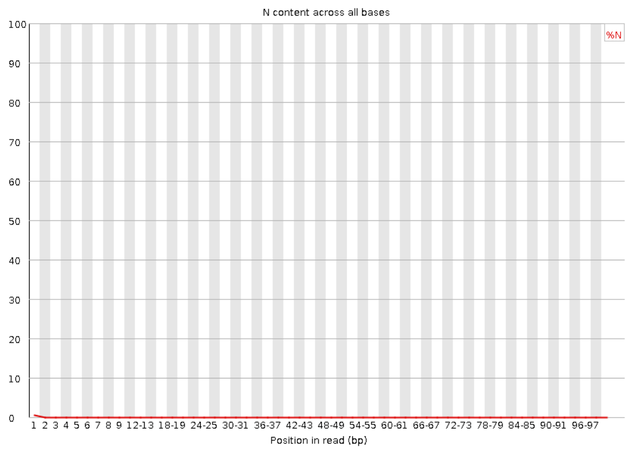


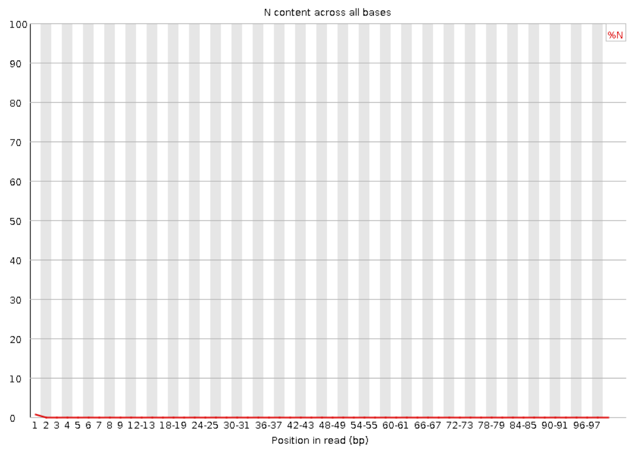
Plots for 23_4A_control_S17:

Graph order: Top two represent mean quality score distribution and the bottom two represent mean ‘N’ content per base. Left two represent Read 1 (forward) and right two represent Read 2 (reverse).







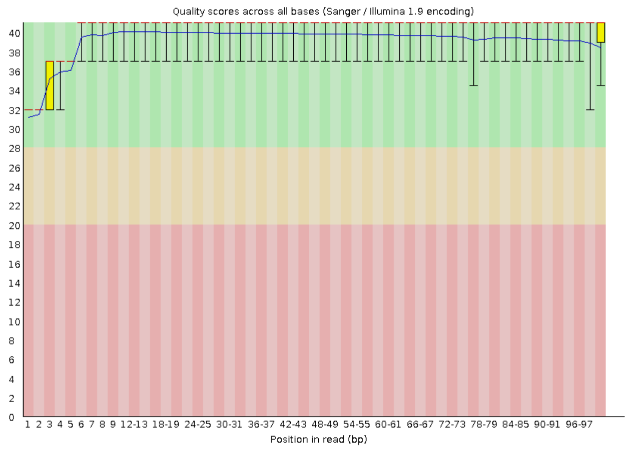


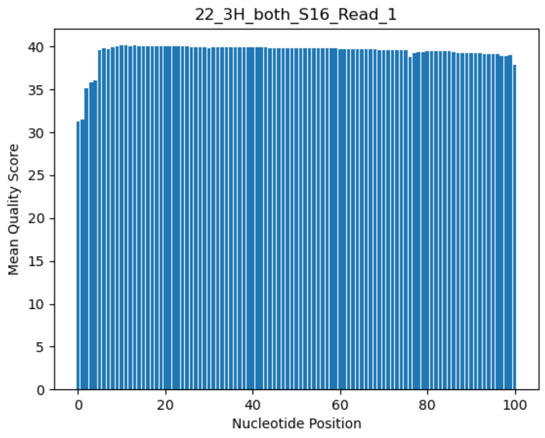
Looking at distribution plots produced by FastQC, it appears that all of the reads have an average quality score above 30, with the majority of values being between 38-40, which is consistent with the distribution plots of ‘n’ content per base which shows almost no ‘N’s’ in the reads.

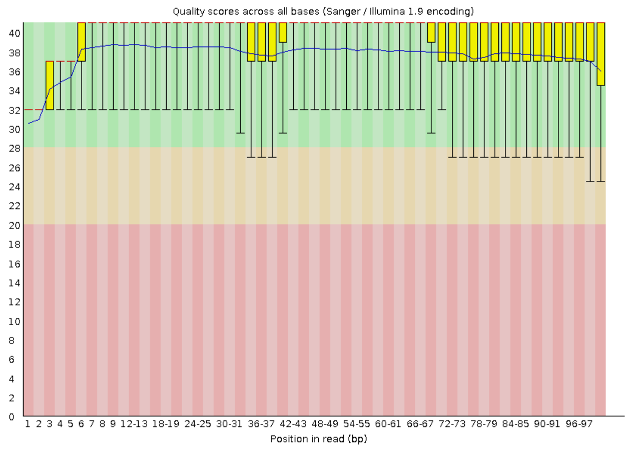
The following is a comparison of the FastQC mean quality score distribution plots with those generated by the script I created. *Note* that mean quality score plots represent the distribution of the mean quality score, based on Phred + 33 for each base position across all sequences.

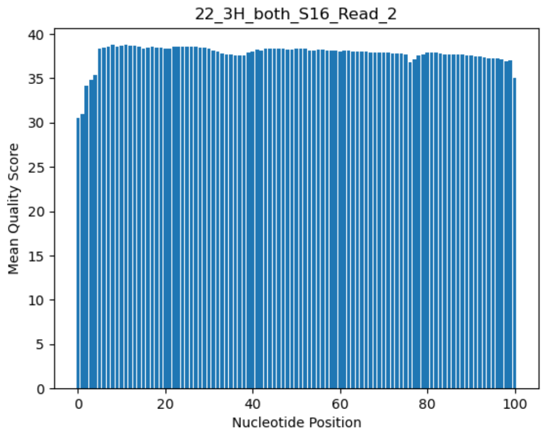
Plots for 22__3H__both:

Graph order: Top two represent Read 1 (forward) and bottom two represent Read 2 (reverse).



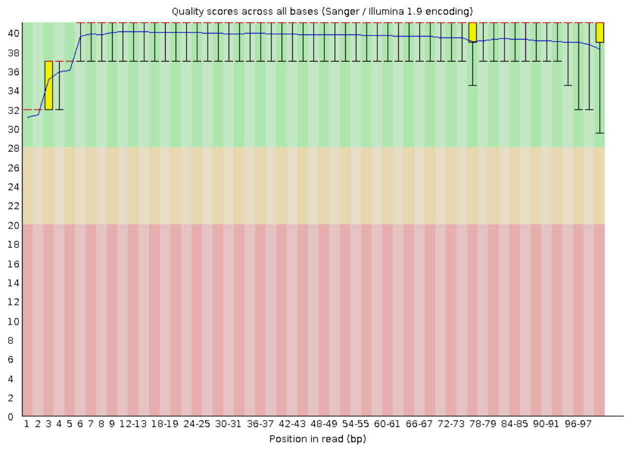


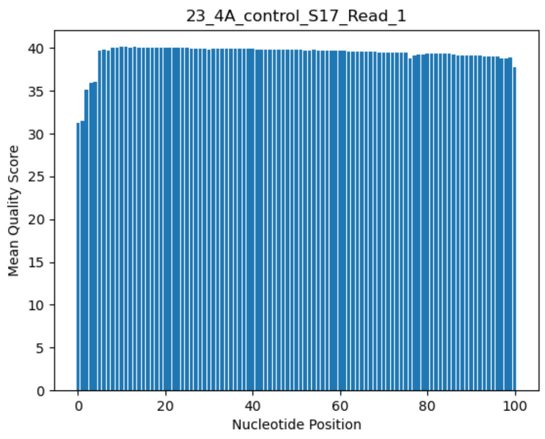


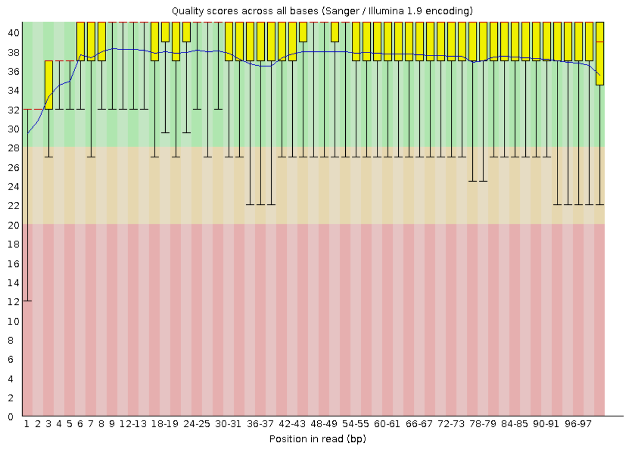


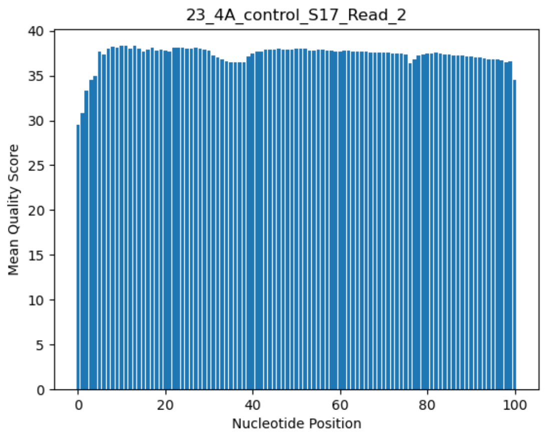
Plots for 23_4A_control:

Graph order: Top two represent Read 1 (forward) and bottom two represent Read 2 (reverse).









In comparing the distribution plots generated by fastQC compared to those I generated, it appears the distribution plots all show an identical pattern to the FastQC distribution plots, without the standard deviation bars. In comparing the run time of my script compared to FastQC, my script ran much slower, with a total completion time for all four graphs of about 40 minutes. The FastQC software generated all the plots in 10 minutes. When considering the cause of this difference, I would think that the fastQC software is optimized for speed and efficieny, whereas my script is not. Additionally, FastQC is written in java, which is a compiled language compared to python, which is an interpreted language, and so it runs slower.

In considering the overall data quality of these two libraries, I would say it is excellent. There are very few 'N's and almost all base-calls have Qscores above 38, which represents an error probability of 0.00016. The only point of concern pertains to 22__3H__both__S16 in that the per-base-quality displays a seemingly high degree of variability in base calls towards the end of the sequence read. While more variable, the variability still seems to lie within the range of high Qscores so it is not much of a concern.

Adaptor trimming comparison

Cutadapt (https://cutadapt.readthedocs.io/en/stable/) was used with the following parameters to trim adapter sequences:

```
cat cutadapt.wrapper.sh
```

```
## #!/usr/bin/bash
## #SBATCH --account=bgmp
## #SBATCH --partition=bgmp
## #SBATCH --nodes=1
## #SBATCH --ntasks-per-node=1
## #SBATCH --cpus-per-task=1
## #SBATCH --time=0-16:00:00
## #SBATCH --output=Cutadapt.out.%j
##
## file1read1='/projects/bgmp/shared/2017_sequencing/demultiplexed/23_4A_control_S17_L008_R1_001.fastq.g
## file1read2='/projects/bgmp/shared/2017_sequencing/demultiplexed/23_4A_control_S17_L008_R2_001.fastq.g
## file2read1='/projects/bgmp/shared/2017_sequencing/demultiplexed/22_3H_both_S16_L008_R1_001.fastq.gz'
## file2read2='/projects/bgmp/shared/2017_sequencing/demultiplexed/22_3H_both_S16_L008_R2_001.fastq.gz'
##
## conda activate QAA
##
## cutadapt -a AGATCGGAAGAGCACACGTCTGAACTCCAGTCA -A AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT \
## -o '/home/zsisson2/bgmp/bioinformatics/Bi623/QAA/Cutadapt_output/23_4A_control_S17_L008_R1.fastq.gz'
## -p '/home/zsisson2/bgmp/bioinformatics/Bi623/QAA/Cutadapt_output/23_4A_control_S17_L008_R2.fastq.gz'
## $file1read1 $file1read2
##
## cutadapt -a AGATCGGAAGAGCACACGTCTGAACTCCAGTCA -A AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT \
## -o '/home/zsisson2/bgmp/bioinformatics/Bi623/QAA/Cutadapt_output/22_3H_both_S16_L008_R1.fastq.gz' \
## -p '/home/zsisson2/bgmp/bioinformatics/Bi623/QAA/Cutadapt_output/22_3H_both_S16_L008_R2.fastq.gz' \
## $file2read1 $file2read2
```

##Note that the adapter sequences were taken from Illumina's support page for the TRUseq kit (https://support.illumina.com/bulletins/2016/12/what-sequences-do-i-use-for-adapter-trimming.html) ## The following table summarizes the proportion of reads that were trimmed:

File	Total Proportion trimmed	Proportion of forward reads trimmed	Proportion of reverse reads trimmed
22_3H_both	8.4%	3.8%	4.6%
23_4A_control	6.8%	3.1%	3.7%

To confirm the adapter sequence orientation, the following command was used to identify sequence lines that contain the adapter:

for Read 1 check (performed on both files, but only one file example shown):

```
zcat /projects/bgmp/shared/2017_sequencing/demultiplexed/22_3H_both_S16_L008_R1_00fastq.gz | \
grep 'AGATCGGAAGAGCACACGTCTGAACTCCAGTCA' | wc -l
```

```
output: 7563
```

for Read 2 check (performed on both files, but only one file example shown):

```
zcat /projects/bgmp/shared/2017_sequencing/demultiplexed/22_3H_both_S16_L008_R2_001.fastq.gz | \
grep 'AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT' | wc -l
```

```
output: 7848
```

In short, the commands are used to read in all lines of the FASTQ file, and grab only those lines which contain the appropriate adapter-sequence string. Finally the code outputs a number representing how many lines contain that sequence. Additionally, by printing out a the grepped lines, we can see that the adapter sequence consistently occurs at the end of the read, confirming its presence. Furthermore, reading into Illumina's adapter kits, I can see that this library was built using the TruSeq single index, which reports the same forward and reverse indexes. https://support.illumina.com/bulletins/2016/12/what-sequences-do-i-use-for-adapter-trimming.html

Next, Trimmomatic (http://www.usadellab.org/cms/?page=trimmomatic) was used with the following parameters to trim the sequence reads.

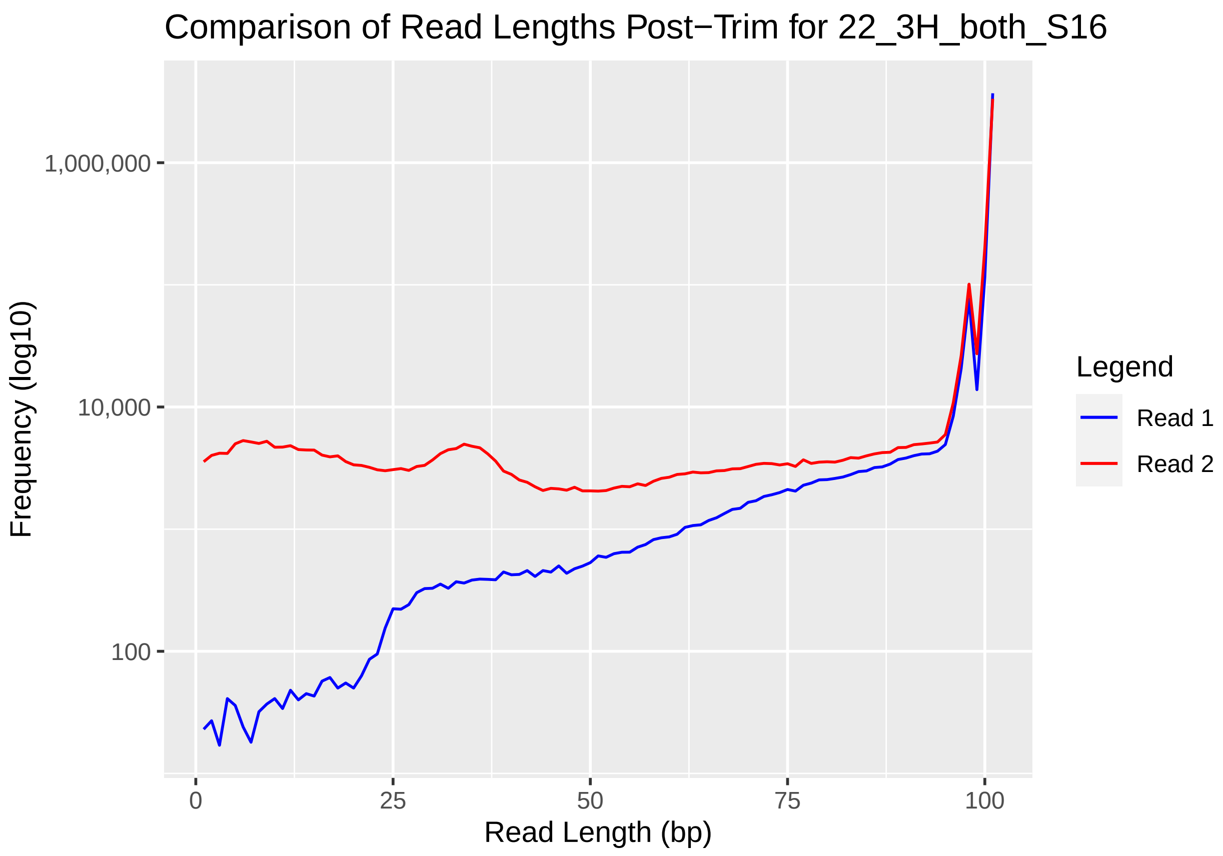
```
cat trimmomatic.wrapper.sh
```

```
## #!/usr/bin/bash
## #SBATCH --account=bgmp
## #SBATCH --partition=bgmp
## #SBATCH --nodes=1
## #SBATCH --ntasks-per-node=1
## #SBATCH --cpus-per-task=8
## #SBATCH --time=0-16:00:00
## #SBATCH --output=trimmomatic.out.%j
##
## #files are taken from 'Cutadapt_output' directory within /QAA
## file1read1='/home/zsisson2/bgmp/bioinformatics/Bi623/QAA/Cutadapt_output/22_3H_both_S16_L008_R1.fast
## file1read2='/home/zsisson2/bgmp/bioinformatics/Bi623/QAA/Cutadapt_output/22_3H_both_S16_L008_R2.fast
## file2read1='/home/zsisson2/bgmp/bioinformatics/Bi623/QAA/Cutadapt_output/23_4A_control_S17_L008_R1.f
## file2read2='/home/zsisson2/bgmp/bioinformatics/Bi623/QAA/Cutadapt_output/23_4A_control_S17_L008_R2.f
##
## conda activate QAA
##
## /usr/bin/time -v  trimmomatic PE -threads 8 $file1read1 $file1read2 \
## 'Trimmomatic_output/22_3H_both_S16_L008_R1.trimmed.fastq.gz' \
## 'Trimmomatic_output/22_3H_both_S16_L008_R1un.trimmed.fastq.gz' \
## 'Trimmomatic_output/22_3H_both_S16_L008_R2.trimmed.fastq.gz' \
## 'Trimmomatic_output/22_3H_both_S16_L008_R2un.trimmed.fastq.gz' \
## LEADING:3 \
## TRAILING:3 \
## SLIDINGWINDOW:5:15 \
##
## /usr/bin/time -v  trimmomatic PE -threads 8 $file2read1 $file2read2 \
## 'Trimmomatic_output/23_4A_control_S17_L008_R1.trimmed.fastq.gz' \
## 'Trimmomatic_output/23_4A_control_S17_L008_R1un.trimmed.fastq.gz' \
## 'Trimmomatic_output/23_4A_control_S17_L008_R2.trimmed.fastq.gz' \
## 'Trimmomatic_output/23_4A_control_S17_L008_R2un.trimmed.fastq.gz' \
## LEADING:3 \
## TRAILING:3 \
## SLIDINGWINDOW:5:15 \
```

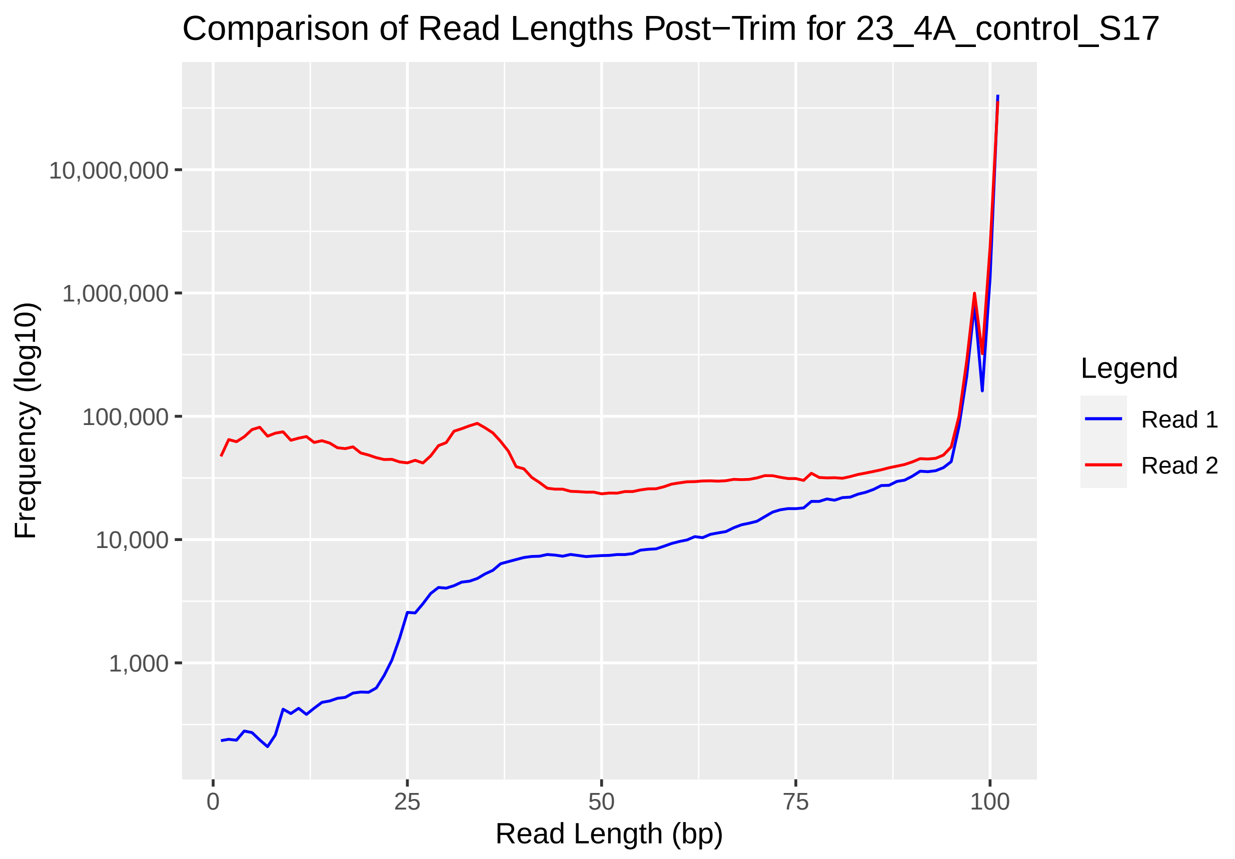
To compare the amount of trimming each read underwent, the following code was used to generate a plot to visualize the difference.

```r
#Reading in the Sequence Data as Vectors
w = c(readLines('22_3H_both_S16_L008_R1.trimmed.seqonly.txt'))

x = c(readLines('22_3H_both_S16_L008_R2.trimmed.seqonly.txt'))

y = c(readLines('23_4A_control_S17_L008_R1.trimmed.seqonly.txt'))

z = c(readLines('23_4A_control_S17_L008_R2.trimmed.seqonly.txt'))

##Create Data Frames for Read1/Read2 of the sequence Data
df_22_3H <- data.frame(w,x)
df_23_4A <- data.frame(y,z)
names(df_22_3H) <- c("Read_1_(forward)","Read_2_(reverse)")
names(df_23_4A) <- c("Read_1_(forward)","Read_2_(reverse)")

##Create a new column containing the counts of the sequences
df_22_3H <- mutate(df_22_3H,Read_1_count=str_length(`Read_1_(forward)`))
df_22_3H <- mutate(df_22_3H,Read_2_count=str_length(`Read_2_(reverse)`))

df_23_4A <- mutate(df_23_4A,Read_1_count=str_length(`Read_1_(forward)`))
df_23_4A <- mutate(df_23_4A,Read_2_count=str_length(`Read_2_(reverse)`))

##Group the sequences by read length
Seq_counts_22_3H <- count(df_22_3H,`Read_1_count`)
Seq_counts_22_3H <- cbind(Seq_counts_22_3H, count(df_22_3H,`Read_2_count`))
colnames(Seq_counts_22_3H) <- c("Post_Trim_Sequence_Length", "count_R1","len","count_R2")
```

```
Seq_counts_23_4A <- count(df_23_4A,`Read_1_count`)
Seq_counts_23_4A <- cbind(Seq_counts_23_4A, count(df_23_4A,`Read_2_count`))
colnames(Seq_counts_23_4A) <- c("Post_Trim_Sequence_Length", "count_R1","len","count_R2")

#Plotting the distributions
colors <- c("Read 1" = "blue", "Read 2" = "red")

distrib_22_3H = ggplot(data = Seq_counts_22_3H, aes(x = Post_Trim_Sequence_Length)) +
  geom_line(aes(y = count_R1, color = "Read 1")) +
  geom_line(aes(y = count_R2, color = "Read 2")) +
  labs(x="Read Length (bp)", y="Frequency (log10)",
  title = "Comparison of Read Lengths Post-Trim for 22_3H_both_S16",
  color = "Legend") +
  scale_color_manual(values = colors) +
  scale_y_continuous(trans = 'log10', labels = scales::comma)

distrib_23_4A = ggplot(data = Seq_counts_23_4A, aes(x = Post_Trim_Sequence_Length)) +
  geom_line(aes(y = count_R1, color = "Read 1")) +
  geom_line(aes(y = count_R2, color = "Read 2")) +
  labs(x="Read Length (bp)", y="Frequency (log10)",
  title = "Comparison of Read Lengths Post-Trim for 23_4A_control_S17",
  color = "Legend") +
  scale_color_manual(values = colors) +
  scale_y_continuous(trans = 'log10', labels = scales::comma)

print(distrib_22_3H)
```



```
print(distrib_23_4A)
```



The Read 2 sequences appear to have undergone more trimming. This falls in line with what I would expect to see, considering that during Illumina sequencing, Read 1 begins at the primer binding and extends towards the 3' end. If the polymerase extends past the index, then the adapter will also get sequenced, resulting in more sequences in read 2 that need to be trimmed than in Read 1.

Alignment and strand-specificity

To align the reads from the RNA-seq, STAR, a splice aware aligner https://github.com/alexdobin/STAR was used. First, GTF and Fasta files for the Mouse genome were acquired from Ensembl via http://ftp.ensembl.org/pub/release-104/gtf/mus_musculus/Mus_musculus.GRCm39.104.gtf.gz and http://ftp.ensembl.org/pub/release-104/fasta/mus_musculus/dna/Mus_musculus.GRCm39.dna.primary_assembly.fa.gz, respectively. A reference database was generated using the following command with STAR installed.

```
cat STAR_generate.sh
```

```
## #!/usr/bin/bash
## #SBATCH --account=bgmp
## #SBATCH --partition=bgmp
## #SBATCH --nodes=1
## #SBATCH --ntasks-per-node=1
## #SBATCH --cpus-per-task=8
## #SBATCH --time=0-20:00:00
## #SBATCH --output=STAR_generate.out.%j
##
## GTF='Ensembl_data/Mus_musculus.GRCm39.104.gtf'
## Fasta='Ensembl_data/Mus_musculus.GRCm39.dna.primary_assembly.fa'
##
## conda activate QAA
## /usr/bin/time -v  STAR --runThreadN 8 \
## --runMode genomeGenerate \
## --genomeDir STAR_mouse_database \
## --genomeFastaFiles $Fasta \
## --sjdbGTFfile $GTF \
```

Sequences were then aligned with the following run parameters.

```
cat STAR_Align_RNAseq.sh
```

```
## #!/usr/bin/bash
## #SBATCH --account=bgmp
## #SBATCH --partition=bgmp
## #SBATCH --nodes=1
## #SBATCH --ntasks-per-node=1
## #SBATCH --cpus-per-task=8
## #SBATCH --time=0-20:00:00
## #SBATCH --output=STARalign.out.%j
##
## Read1_22_3H='Trimmomatic_output/22_3H_both_S16_L008_R1.trimmed.fastq.gz'
## Read2_22_3H='Trimmomatic_output/22_3H_both_S16_L008_R2.trimmed.fastq.gz'
##
## Read1_23_4A='Trimmomatic_output/23_4A_control_S17_L008_R1.trimmed.fastq.gz'
## Read2_23_4A='Trimmomatic_output/23_4A_control_S17_L008_R2.trimmed.fastq.gz'
##
## /usr/bin/time -v  STAR --runThreadN 8 \
## --runMode alignReads \
## --outFilterMultimapNmax 3 \
## --outSAMunmapped Within KeepPairs \
## --alignIntronMax 1000000 \
## --alignMatesGapMax 1000000 \
## --readFilesCommand zcat \
## --readFilesIn $Read1_22_3H $Read2_22_3H \
## --genomeDir STAR_mouse_database \
## --outFileNamePrefix 'RNA_seq_aligned/Mus_22_3H_both_16_L008.'
##
##
## /usr/bin/time -v  STAR --runThreadN 8 \
## --runMode alignReads \
## --outFilterMultimapNmax 3 \
## --outSAMunmapped Within KeepPairs \
## --alignIntronMax 1000000 \
## --alignMatesGapMax 1000000 \
## --readFilesCommand zcat \
## --readFilesIn $Read1_23_4A $Read2_23_4A \
## --genomeDir STAR_mouse_database \
## --outFileNamePrefix 'RNA_seq_aligned/Mus_23_4A_control_S17_L008.'
```

From the SAM file produced during this alignment, the following read mapping statistics were produced.

For 22_3H_both_S16_L008:

```
Sequences Mapped:  7869231
Sequences Unmapped:  215473
```

And for 23_4A_control_S17_L008:

```
Sequences Mapped:  82156195
Sequences Unmapped:  6154403
```

Next, a count of reads that mapped to features was produced using HTseq (https://htseq.readthedocs.io/en/master/) using the following command. The count was performed two times for each file, with the parameter '–stranded' being set to both 'yes' and 'no' in order to compare results.

```
cat HTseq_count.sh
```

```
## #!/usr/bin/bash
## #SBATCH --account=bgmp
## #SBATCH --partition=bgmp
## #SBATCH --nodes=1
## #SBATCH --ntasks-per-node=1
## #SBATCH --cpus-per-task=1
## #SBATCH --time=0-20:00:00
## #SBATCH --output=HTseq_count.out.%j
##
##
## file_22_3H_both_16='RNA_seq_aligned/Mus_22_3H_both_16_L008.Aligned.out.sam'
## file_23_4A_control_17='RNA_seq_aligned/Mus_23_4A_control_S17_L008.Aligned.out.sam'
## GTF='Ensembl_data/Mus_musculus.GRCm39.104.gtf'
##
## conda activate QAA
##
## /usr/bin/time -v htseq-count --stranded=yes $file_22_3H_both_16 $GTF > 'HTseq_output/22_3H_both_stra
## /usr/bin/time -v htseq-count --stranded=no $file_22_3H_both_16 $GTF > 'HTseq_output/22_3H_both_unstr
##
## /usr/bin/time -v htseq-count --stranded=yes $file_23_4A_control_17 $GTF > 'HTseq_output/23_4A_contro
## /usr/bin/time -v htseq-count --stranded=no $file_23_4A_control_17 $GTF > 'HTseq_output/23_4A_control
```

The resulting files were then processed using the following code to sum the number of total feature hits.

```
#22_3H_both_stranded
cat 22_3H_both_stranded | grep '^EN' | awk '{sum += $2}END{print sum }'

#22_3H_both_unstranded
cat 22_3H_both_unstranded | grep '^EN' | awk '{sum += $2}END{print sum }'

#23_4A_control_stranded
cat 23_4A_control_stranded | grep '^EN' | awk '{sum += $2}END{print sum }'

#23_4A_control_unstranded
cat 23_4A_control_unstranded | grep '^EN' | awk '{sum += $2}END{print sum }'
```

```
## 144370
## 3334038
## 1546457
## 32327308
```

The results are summarized in the following tables:

For 22_3H_both:

Statistic	Stranded = yes	Stranded = no
No_feature	3,573,473	190,786
ambiguous	2,333	195,352
too_low_aQual	80,996	80,996
not_aligned	64,525	64,525
alignment_not_unique	176,655	176,655
Total Sequences	4,042,352	4,042,352
Total Mapped	144,370	3,334,038
Percent Hit	3.57%	82.47%

For 23_4A_control:

Statistic	Stranded = yes	Stranded = no
No_feature	36,845,000	4,087,870
ambiguous	35,387	2,011,666
too_low_aQual	1,167,001	1,167,001
not_aligned	2,438,035	2,438,035
alignment_not_unique	2,123,419	2,123,419
Total Sequences	44155299	44155299
Total Mapped	1546457	32327308
Percent Hit	3.50%	73.21%

From these data, I would conclude that the the RNA-seq libraries were made with a strand-specific kit. In stranded kits there should be 50% less reads that mapped when compared to the unstranded kits. This is because in stranded kits, only one of the two RNA strands gets used in the final library and so only one strand with a specific directionality gets mapped to the reference genome, halving the number of reads that mapped. Looking at the tables, it appears that when the stranded flag = no, the percentage of gene features with hits is 82% and 73% for the two libraries. Looking at the percentages for the stranded = yes column, both libraries show values of about 3.5%, much lower than the expected value of ~40% and ~35% if the kit was truly unstranded, and would display half the number of hits as the in the stranded = no run. For this reason I would conclude that the kits used were stranded.