Bi623: RNA-Seq Quality Assessment Assignment

Thai Nguyen

Due 9/7/2021

Objectives

The objectives of this assignment are to use existing tools for quality assessment and adapter trimming, compare the quality assessments to those from your own software, and to demonstrate your ability to summarize other important information about this RNA-Seq data set.

Data:

Each of you will be working with 2 of the demultiplexed file pairs. For all steps below, process the two libraries separately. Library assignments are here: `/projects/bgmp/shared/Bi623/QAA_data_assignments.txt`

The demultiplexed, gzipped .fastq files are here: `/projects/bgmp/shared/2017_sequencing/demultiplexed/`

Do not move, copy, or unzip these data!

Part 1 – Read quality score distributions

1. Using `FastQC` via the command line on Talapas, produce plots of quality score distributions for R1 and R2 reads. Also, produce plots of the per-base N content, and comment on whether or not they are consistent with the quality score plots. As shown in the plots below, the per-base N content plots are consistent with the per-base quality score distribution plots. The per-base N content plots have higher N content for the first few base pair positions, which corresponds with the lower q-scores for the first few base pair positions in the q-score distribution plots.

FastQC bash script:

```
#!/bin/bash

#SBATCH --account=bgmp
#SBATCH --partition=bgmp
#SBATCH --cpus-per-task=8
#SBATCH --time=10:00:00
#SBATCH --output=fastqc_pt1_%j.out
#SBATCH --error=fastqc_pt1_%j.err

module load fastqc/0.11.5
```

```
input_dir="/projects/bgmp/shared/2017_sequencing/demultiplexed/"
dataset1_r1="21_3G_both_S15_L008_R1_001.fastq.gz"
dataset1_r2="21_3G_both_S15_L008_R2_001.fastq.gz"
dataset2_r1="4_2C_mbnl_S4_L008_R1_001.fastq.gz"
dataset2_r2="4_2C_mbnl_S4_L008_R2_001.fastq.gz"
output_dir="/projects/bgmp/tnguye14/bioinfo/Bi623/QAA/pt1/fastqc_pt1_output/"

/usr/bin/time -v fastqc -t 2 ${input_dir}$dataset1_r1 ${input_dir}$dataset1_r2 -o $output_dir
/usr/bin/time -v fastqc -t 2 ${input_dir}$dataset2_r1 ${input_dir}$dataset2_r2 -o $output_dir

exit
```

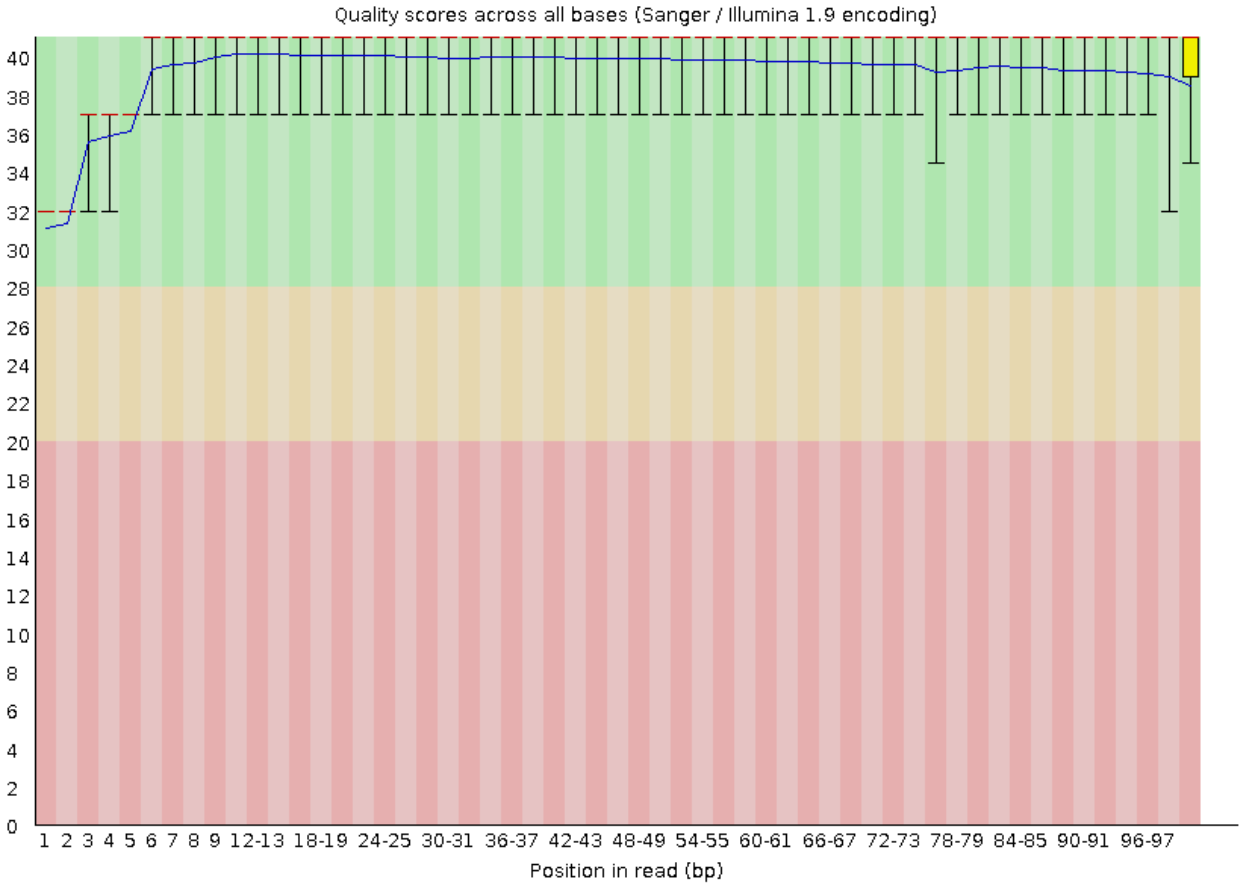

Figure 1: Quality score distribution for 4_2C_mbnl_S4_L008_R1.

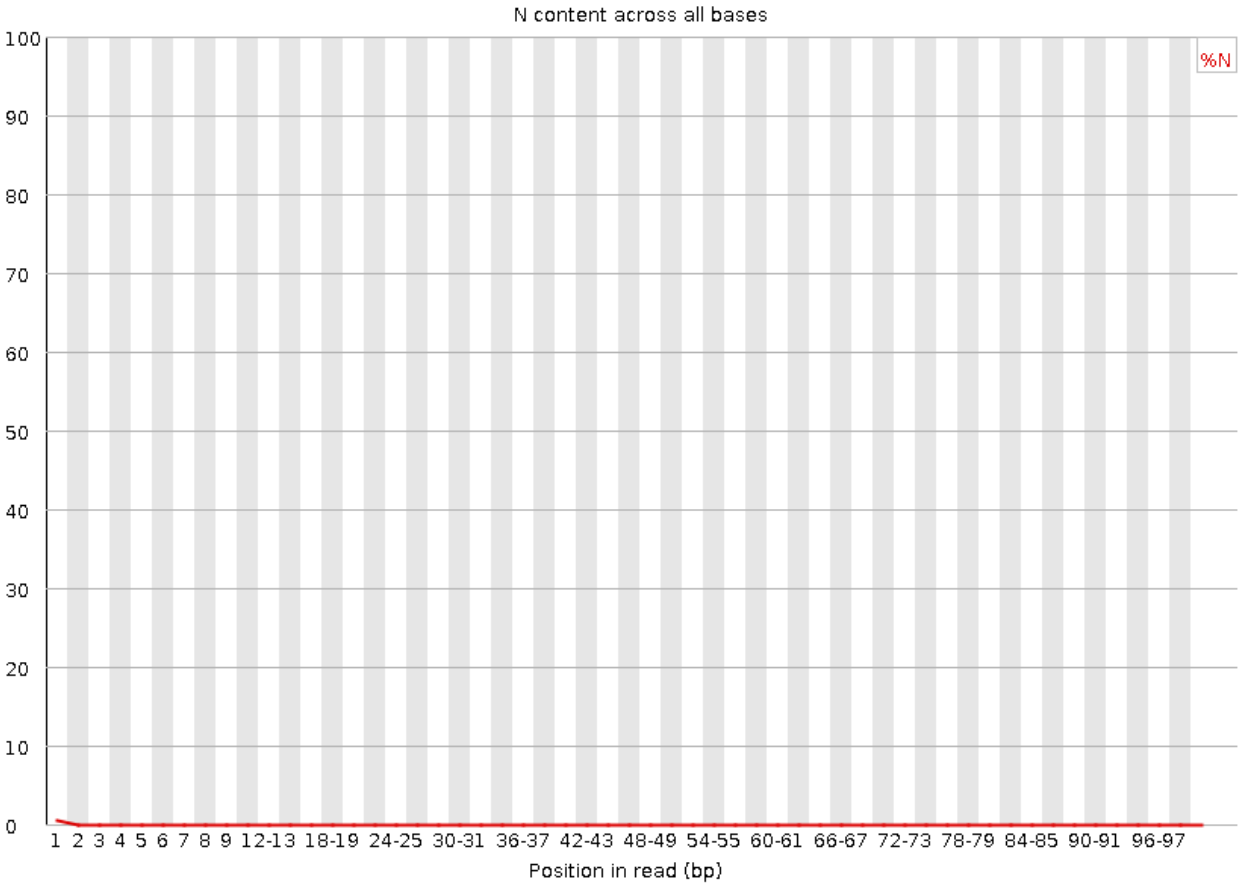

Figure 2: N content for 4_2C_mbnl_S4_L008_R1.

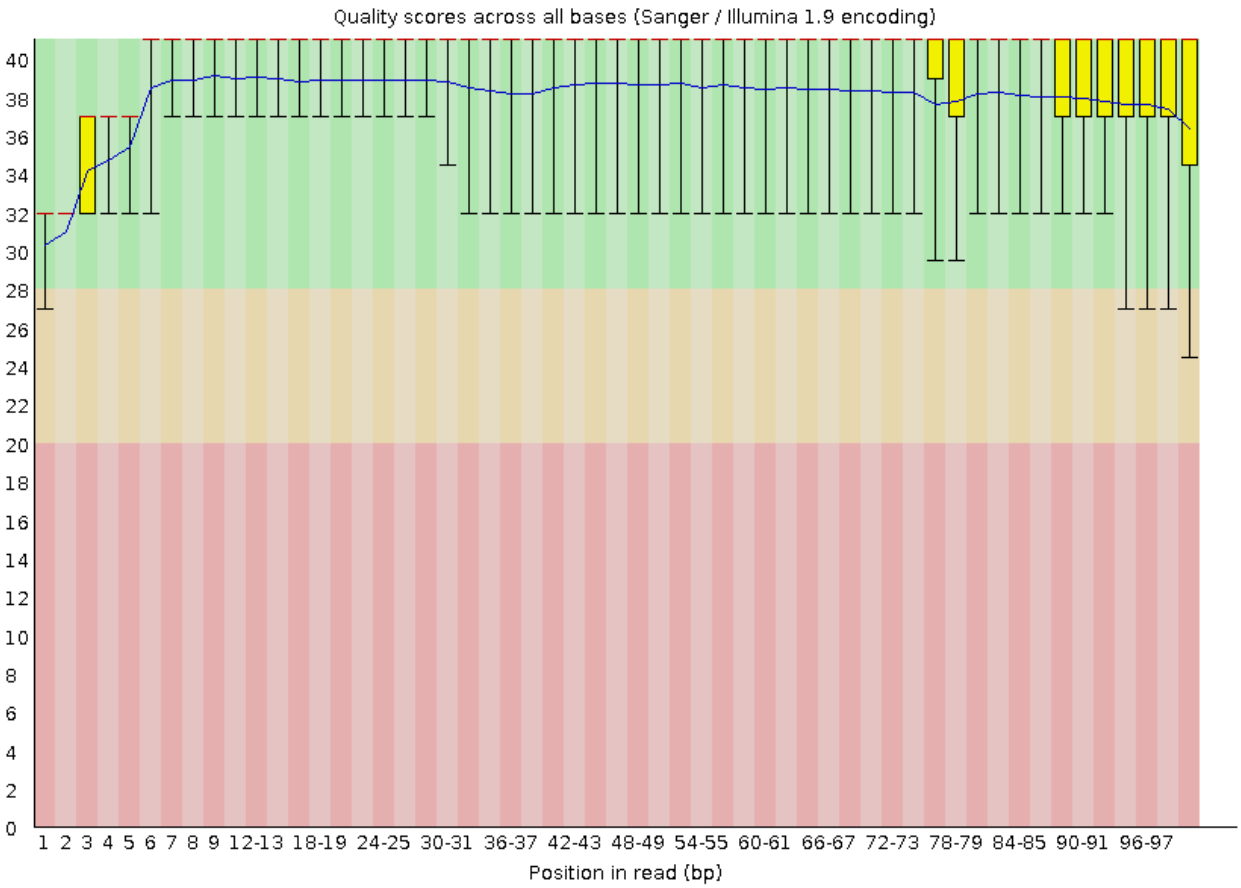

Figure 3: Quality score distribution for 4_2C_mbnl_S4_L008_R2.

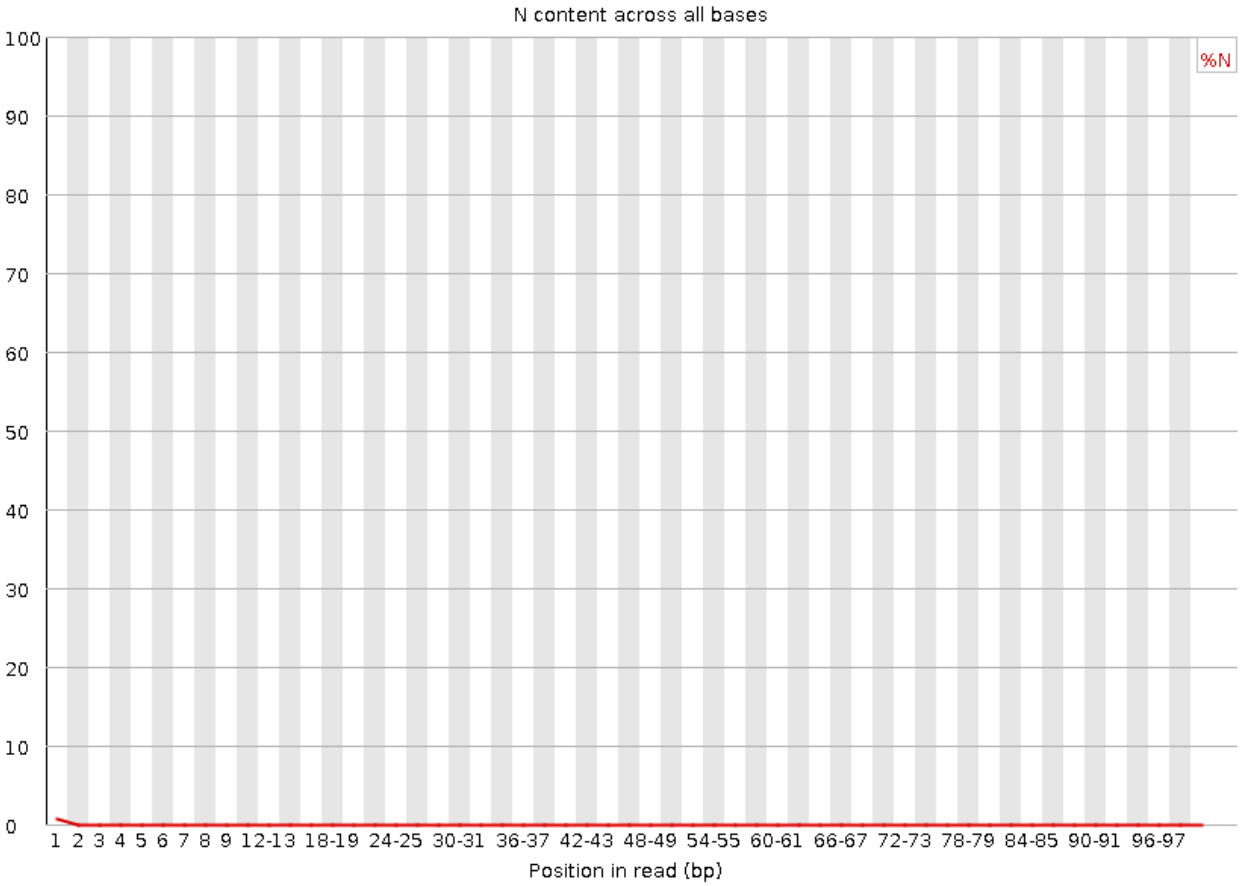

Figure 4: N content for 4_2C_mbnl_S4_L008_R2.

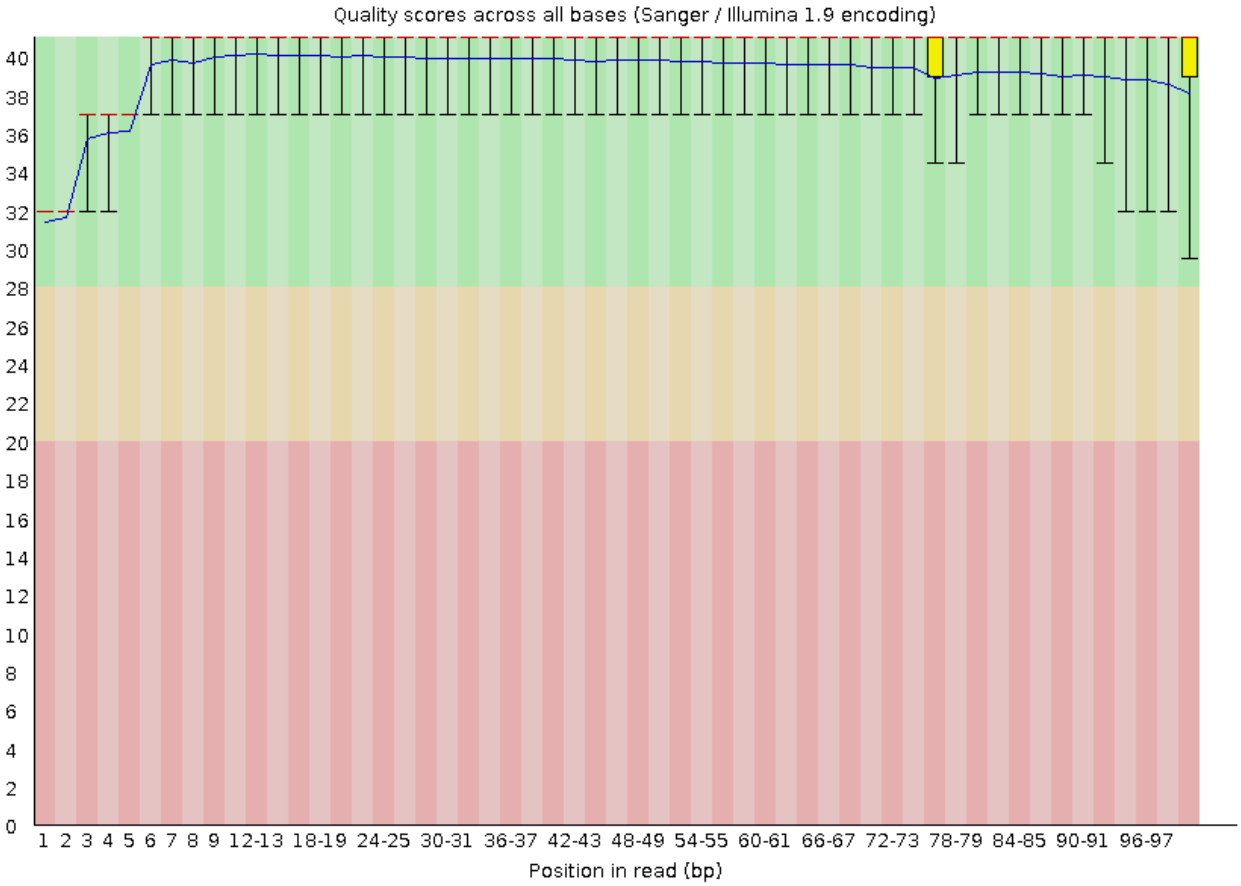

Figure 5: Quality score distribution for 21_3G_both_S15_L008_R1.

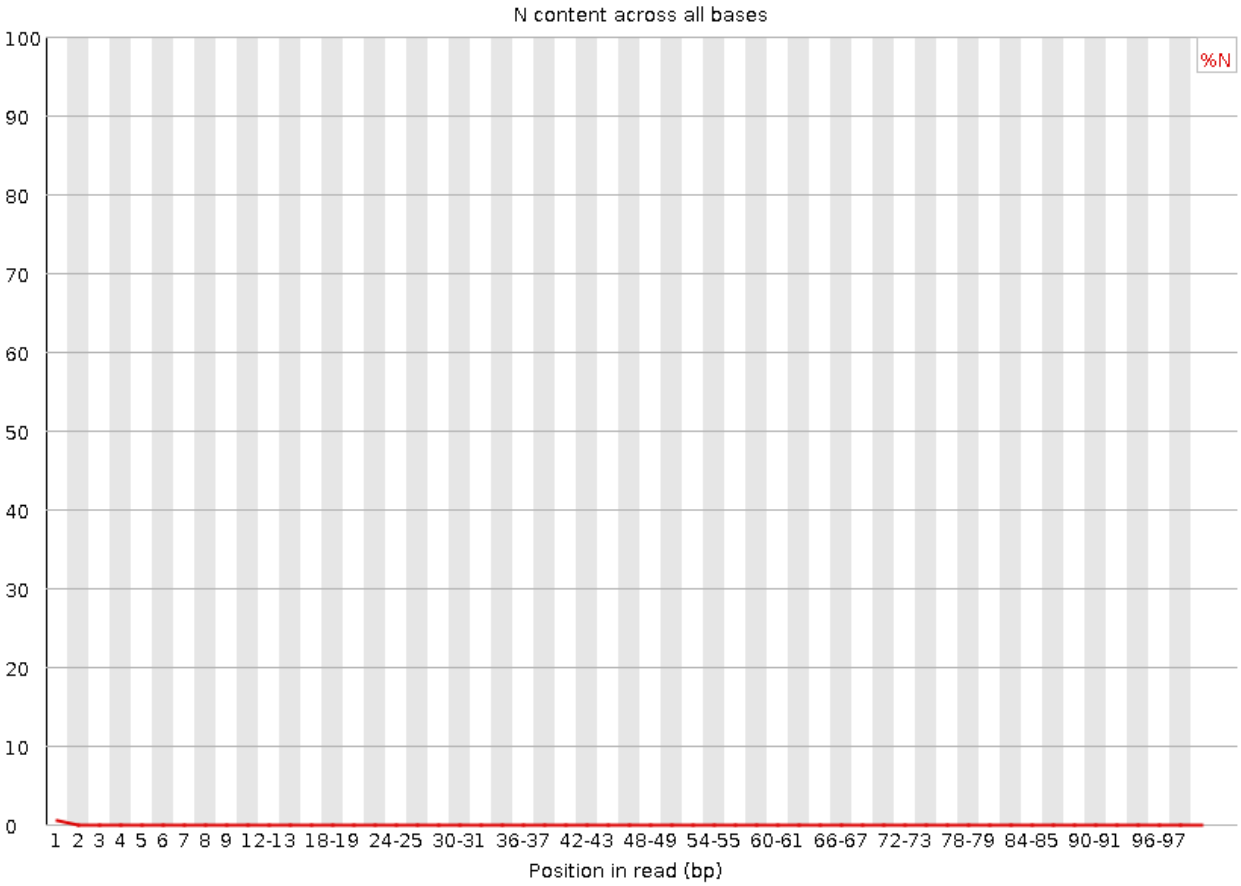

Figure 6: N content for 21_3G_both_S15_L008_R1.

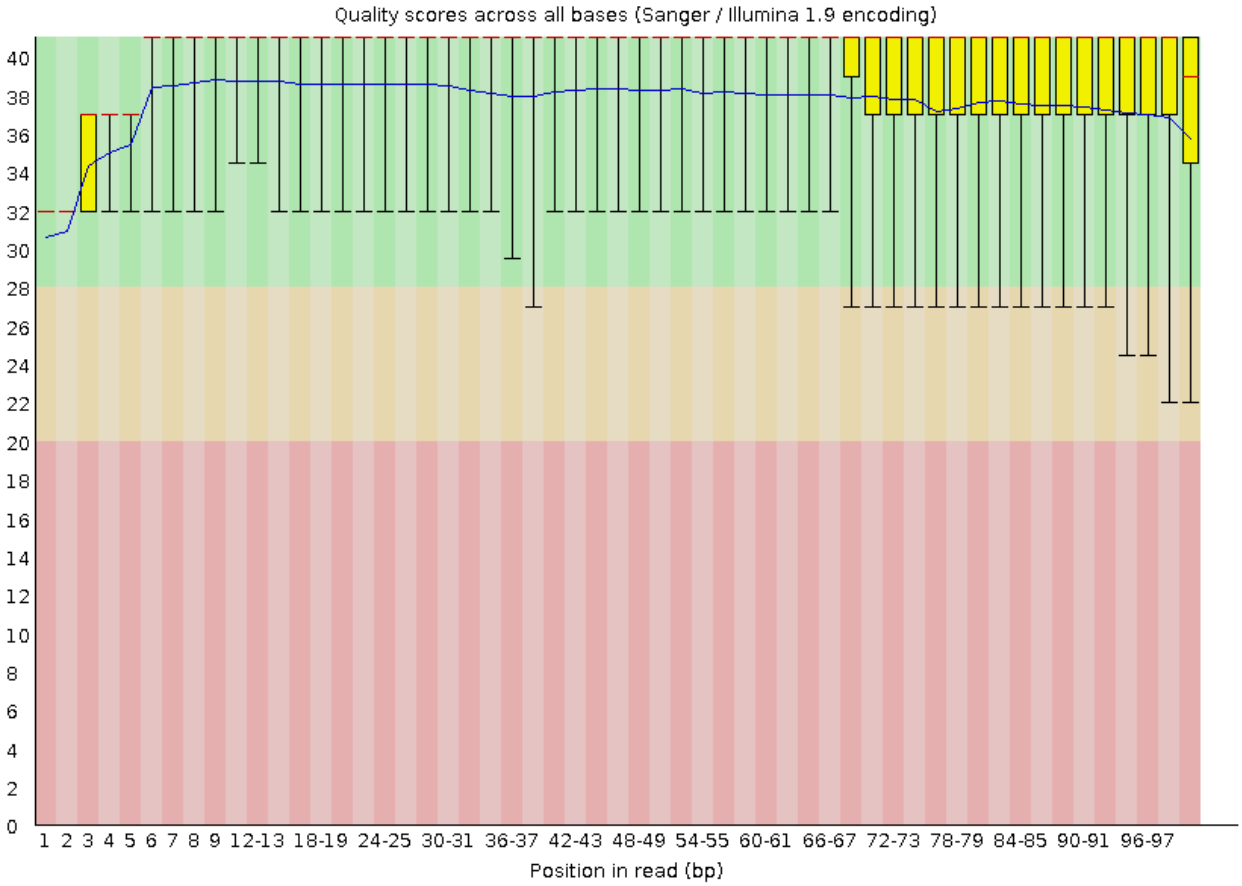

Figure 7: Quality score distribution for 21_3G_both_S15_L008_R2.

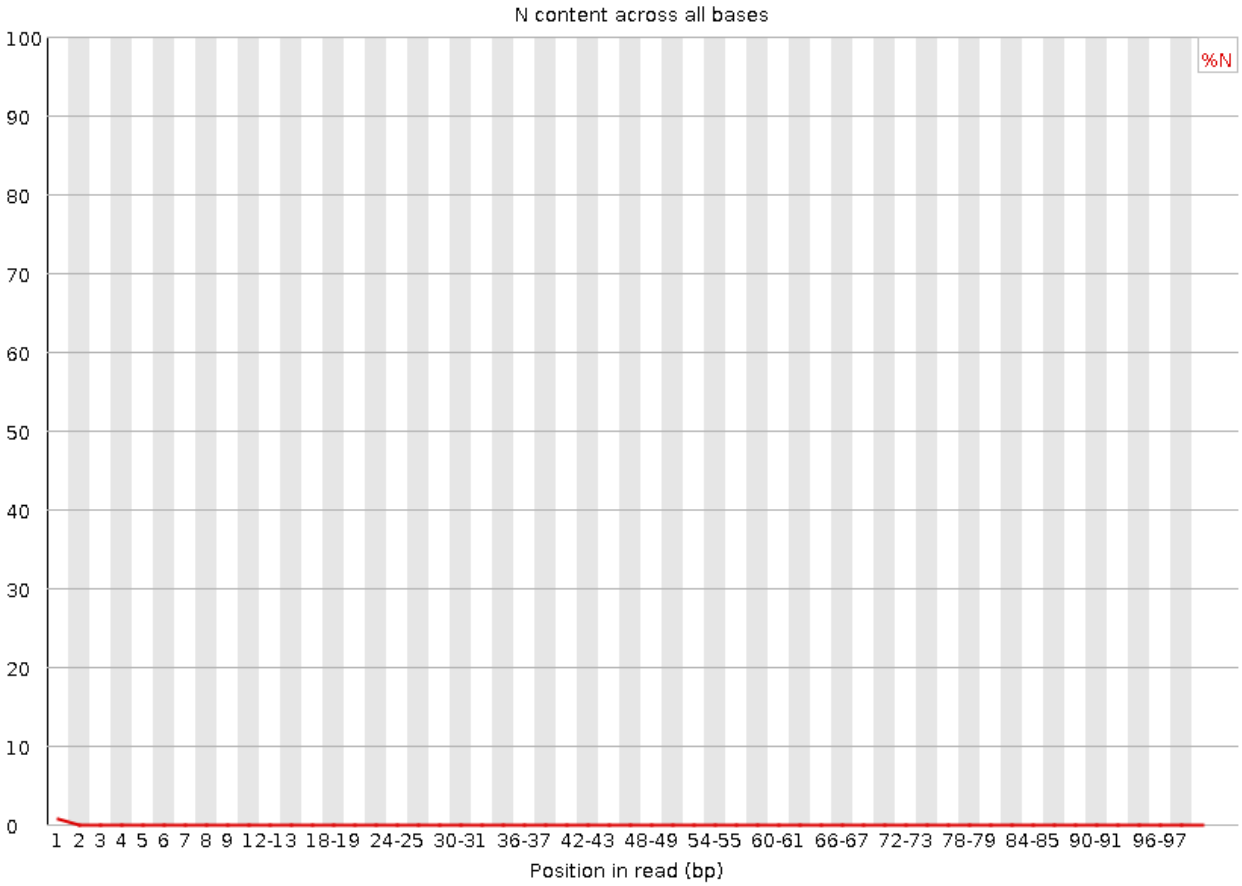

Figure 8: N content for 21_3G_both_S15_L008_R2.

2. Run your quality score plotting script from your Demultiplexing assignment from Bi622. 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? See below plots for Python and bash wrapper scripts.

The FastQC qscore distribution plots show a similar trend to the plots generated by my script: lower qscores for the first few base pair positions, then mostly consistently high qscores for the rest of the base pair positions.

The runtime of the FastQC commands was also much faster than the runtime for my qscore plotting scripts. It took FastQC a total of about 2 mins 8 secs to process all 4 input files, while my script took a total of about 18 mins to process all 4 input files.

One reason for this difference in runtime could be that FastQC is written in Java by professionals, while my script is written in Python, which may not handle memory as efficiently.

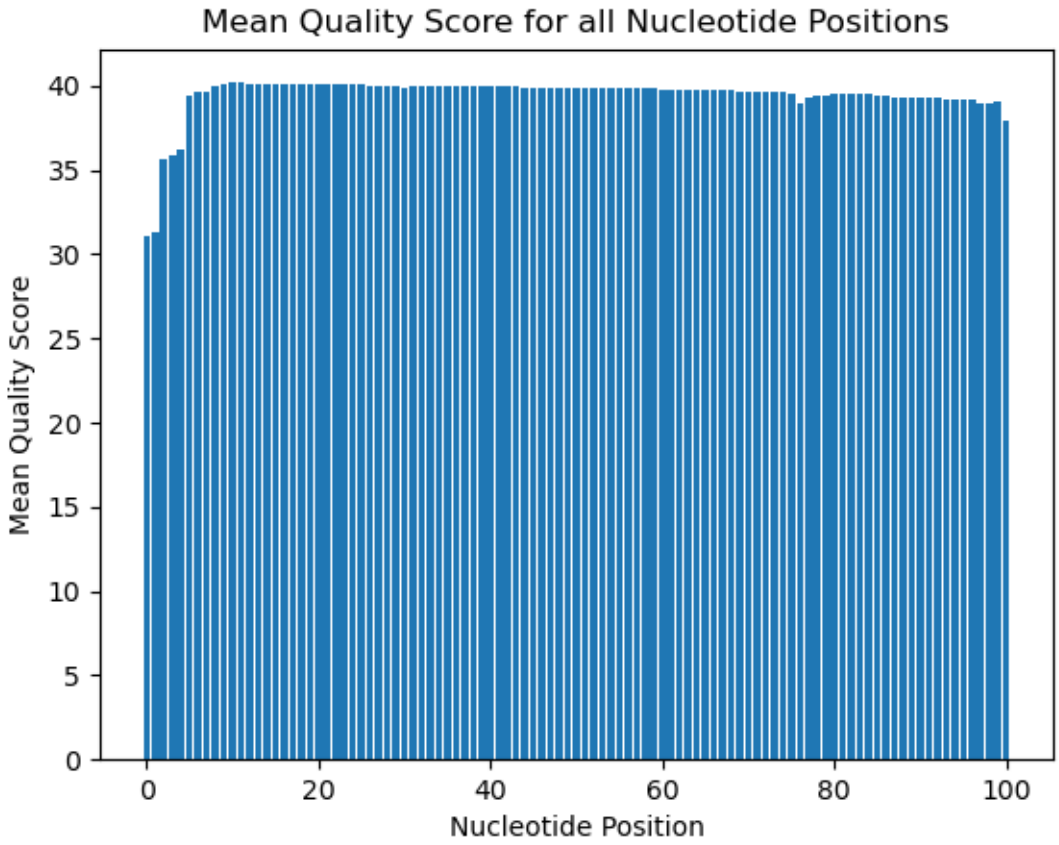

Figure 9: Mean quality score distribution from Python script for 4_2C_mbnl_S4_L008_R1.

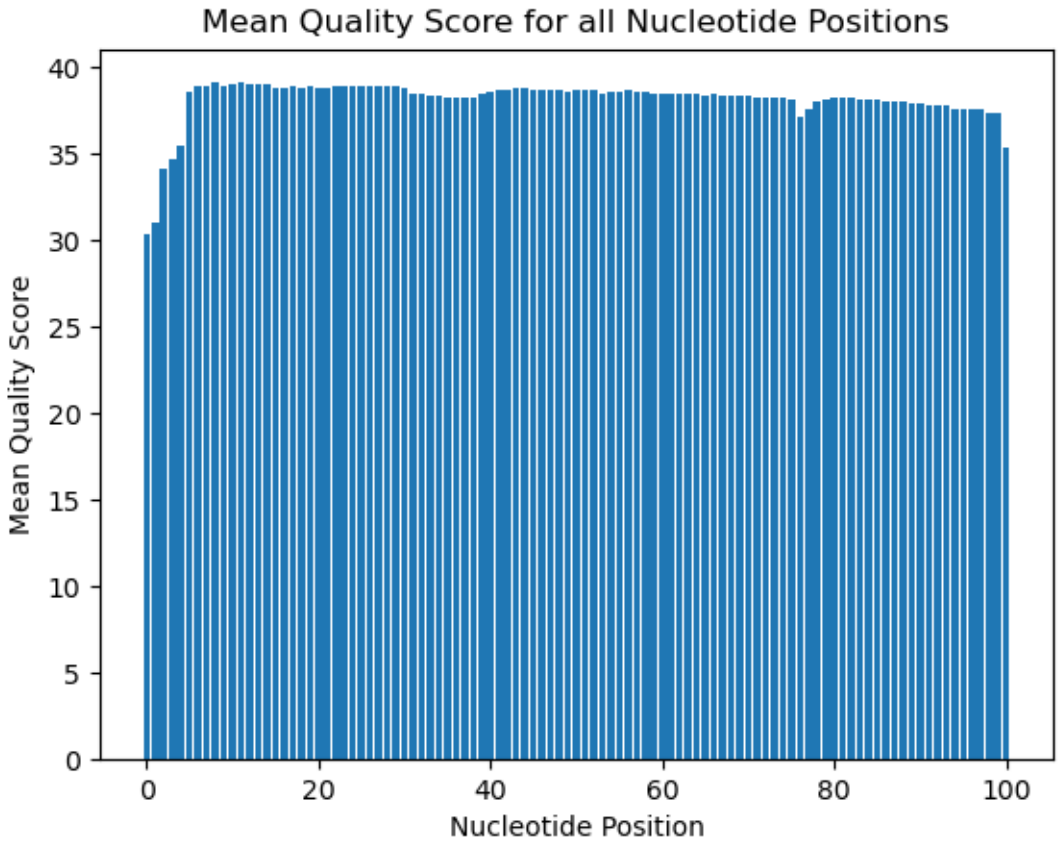

Figure 10: Mean quality score distribution from Python script for 4_2C_mbnl_S4_L008_R2.

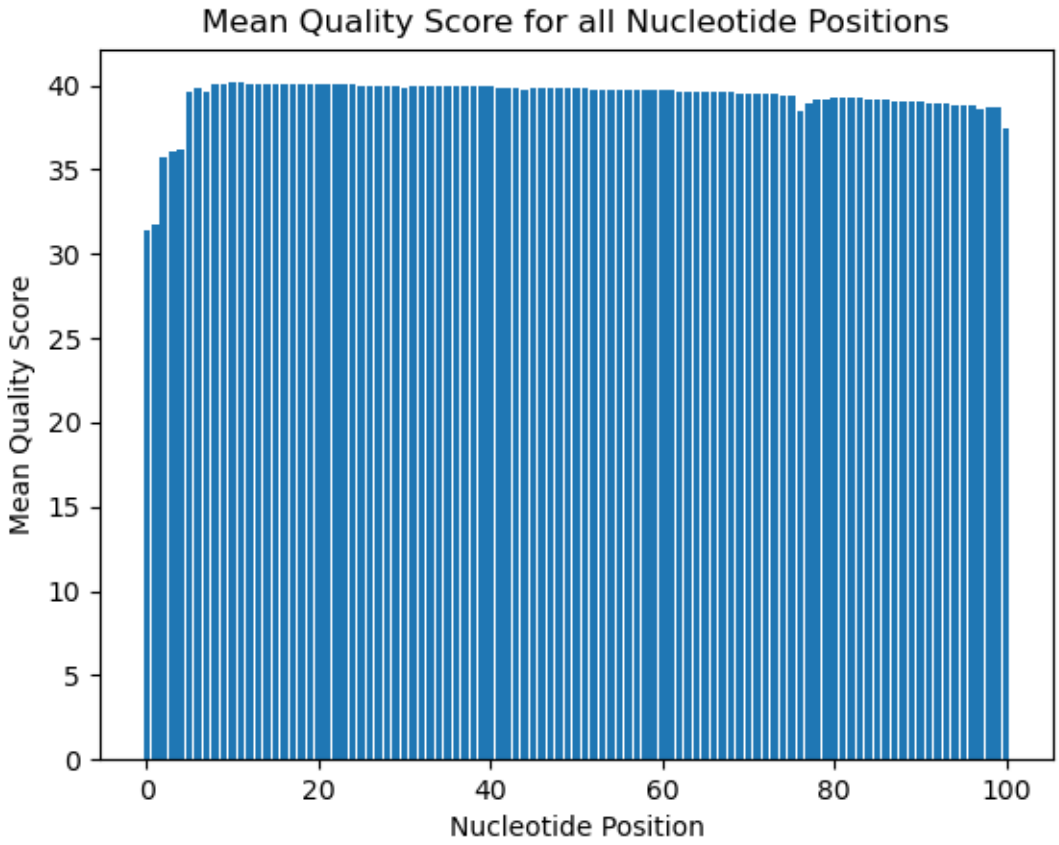

Figure 11: Mean quality score distribution from Python script for 21_3G_both_S15_L008_R1.

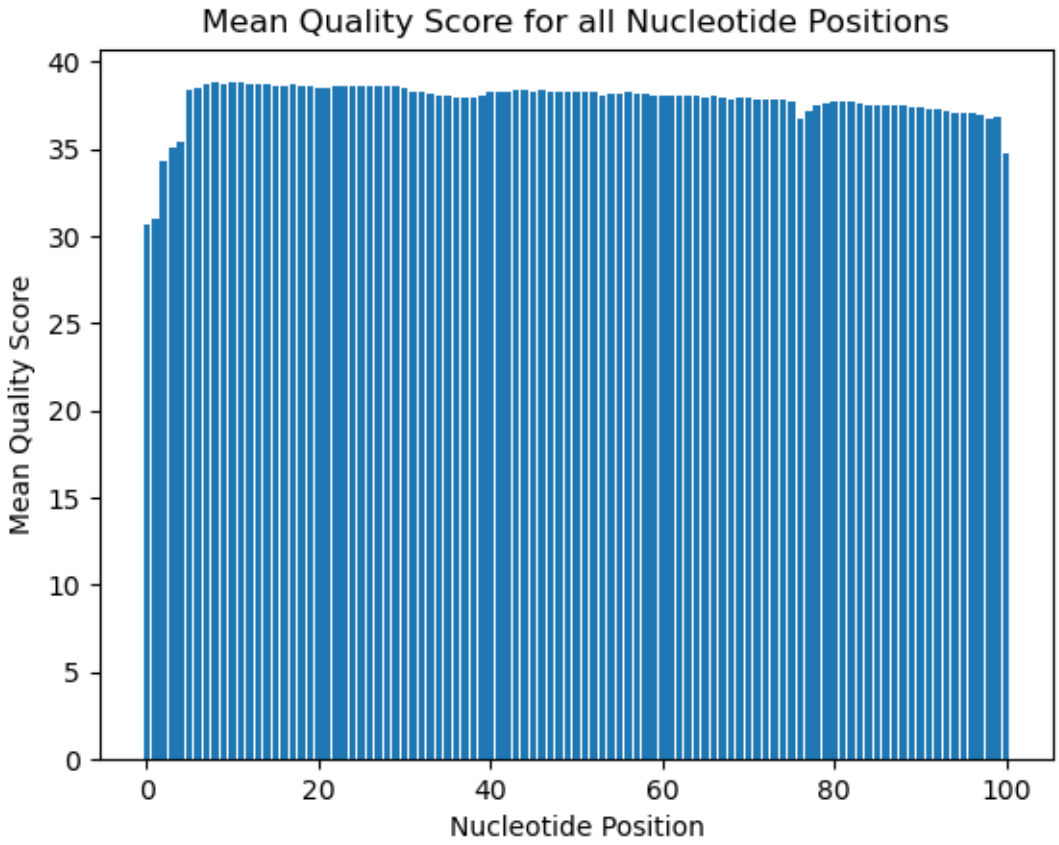

Figure 12: Mean quality score distribution from Python script for 21_3G_both_S15_L008_R2.

Quality score plotting python script:

```
#!/bin/python
import argparse
import gzip     #needed to read g-zipped files
import matplotlib.pyplot as plt
import Bioinfo

def get_args():
    '''Defines/sets possible command line arguments for script'''
    parser = argparse.ArgumentParser("A program to process FASTQ data, get average quality score for ea
    parser.add_argument("-f", nargs="+", help="specifies input FASTQ filename(s)", type=str, required=T
    parser.add_argument("-o", help="specifies output file prefix", type=str, required=True)
    return parser.parse_args()

def parse_file(in_filename: str, out_file_prefix: str):
    '''Open gzipped FASTQ file, put all qscores in file into arrays to track qscores for each nucleotid
    # local variables
    line_num: int = 0   #keep track of what line number of input file you're on
    read_num: int = 0   #keep track of number of records/reads in input file
    nuc_pos: int = 0    #keep track of nucleotide position
    converted_phred: int = 0
```

```
    sum_qscores_list: list = []     #keep track of the running total of qscores for each nucleotide pos
    mean_qscores_list: list = []    #holds the mean qscore for each nucleotide position
    output_fname: str = "means_" + out_file_prefix + ".tsv"
    plot_name: str = "plot_" + out_file_prefix + ".png"

    with gzip.open(in_filename, 'rt') as input_file, open(output_fname, 'w') as output_file:
        for line in input_file:
            line_num += 1
            line = line.strip()
            if line_num % 4 == 0:   #grab the qscore line in filename
                nuc_pos = 0
                for char in line:
                    converted_phred = Bioinfo.convert_phred(char)
                    #if the sum_qscores_list doesn't yet have an element for each read position in the
                    if len(sum_qscores_list) != len(line):
                        sum_qscores_list.append(converted_phred)
                    else:
                        sum_qscores_list[nuc_pos] += converted_phred
                    nuc_pos += 1
                read_num += 1   #once whole qscore line is read, increment the read number
        mean_qscores_list = [sum_qscore / read_num for sum_qscore in sum_qscores_list]    #divide each

        #write the mean qscore for each nucleotide position into an output file
        output_file.write("Position\tMean\n")
        for i in range(len(mean_qscores_list)):
            output_file.write(str(i) + "\t" + str(mean_qscores_list[i]) + "\n")

    #plot mean qscore distribution
    plt.bar(range(len(mean_qscores_list)), mean_qscores_list)
    plt.title("Mean Quality Score for all Nucleotide Positions")
    plt.xlabel("Nucleotide Position")
    plt.ylabel("Mean Quality Score")
    plt.savefig(plot_name)

def main():
    '''Main function, drives the order of execution for script'''
    # local variables
    args = get_args()
    output_file_prefix = args.o

    for input_filename in args.f:
        parse_file(input_filename, output_file_prefix)

if __name__ == "__main__":
    main()
```

Wrapper scripts for generating quality score distribution plots:

read_qscores_f1_r1_wrapper.sh

```
#!/bin/bash

#SBATCH --account=bgmp
#SBATCH --partition=bgmp
```

```
#SBATCH --cpus-per-task=8
#SBATCH --time=16:00:00
#SBATCH --output=f1_r1_read_qscores_wrapper_%j.out
#SBATCH --error=f1_r1_read_qscores_wrapper_%j.err

conda activate bgmp_py39
script_file="/projects/bgmp/tnguye14/bioinfo/Bi623/QAA/pt1/read_qscores.py"
input_dir="/projects/bgmp/shared/2017_sequencing/demultiplexed/"
input_file="21_3G_both_S15_L008_R1_001.fastq.gz"
output_file_prefix=$input_file

/usr/bin/time -v python $script_file -f ${input_dir}$input_file -o $output_file_prefix

exit
```

read_qscores_f1_r2_wrapper.sh

```
#!/bin/bash

#SBATCH --account=bgmp
#SBATCH --partition=bgmp
#SBATCH --cpus-per-task=8
#SBATCH --time=16:00:00
#SBATCH --output=f1_r2_read_qscores_wrapper_%j.out
#SBATCH --error=f1_r2_read_qscores_wrapper_%j.err

conda activate bgmp_py39
script_file="/projects/bgmp/tnguye14/bioinfo/Bi623/QAA/pt1/read_qscores.py"
input_dir="/projects/bgmp/shared/2017_sequencing/demultiplexed/"
input_file="21_3G_both_S15_L008_R2_001.fastq.gz"
output_file_prefix=$input_file

/usr/bin/time -v python $script_file -f ${input_dir}$input_file -o $output_file_prefix

exit
```

read_qscores_f2_r1_wrapper.sh

```
#!/bin/bash

#SBATCH --account=bgmp
#SBATCH --partition=bgmp
#SBATCH --cpus-per-task=8
#SBATCH --time=16:00:00
#SBATCH --output=f2_r1_read_qscores_wrapper_%j.out
#SBATCH --error=f2_r1_read_qscores_wrapper_%j.err

conda activate bgmp_py39
script_file="/projects/bgmp/tnguye14/bioinfo/Bi623/QAA/pt1/read_qscores.py"
input_dir="/projects/bgmp/shared/2017_sequencing/demultiplexed/"
input_file="4_2C_mbnl_S4_L008_R1_001.fastq.gz"
output_file_prefix=$input_file
```

```
/usr/bin/time -v python $script_file -f ${input_dir}$input_file -o $output_file_prefix

exit
```

read_qscores_f2_r2_wrapper.sh

```
#!/bin/bash

#SBATCH --account=bgmp
#SBATCH --partition=bgmp
#SBATCH --cpus-per-task=8
#SBATCH --time=16:00:00
#SBATCH --output=f2_r2_read_qscores_wrapper_%j.out
#SBATCH --error=f2_r2_read_qscores_wrapper_%j.err

conda activate bgmp_py39
script_file="/projects/bgmp/tnguye14/bioinfo/Bi623/QAA/pt1/read_qscores.py"
input_dir="/projects/bgmp/shared/2017_sequencing/demultiplexed/"
input_file="4_2C_mbnl_S4_L008_R2_001.fastq.gz"
output_file_prefix=$input_file

/usr/bin/time -v python $script_file -f ${input_dir}$input_file -o $output_file_prefix

exit
```

3. Comment on the overall data quality of your two libraries. The overall data quality of the two libraries is high (average qscores > 38 except for the first few base pair positions), with the read 2 (R2) file for each library having slightly lower qscores on average.

Part 2 – Adapter trimming comparison

4. Create a new conda environment called `QAA` and install `cutadapt` and `Trimmomatic`. Google around if you need a refresher on how to create conda environments. Make sure you check your installations with:

- `cutadapt --version` (should be 3.4)
- `trimmomatic -version` (should be 0.39)

Commands on Talapas:

```
conda create --name QAA python=3.9
conda activate QAA
conda install cutadapt
conda install Trimmomatic
```

5. Using `cutadapt`, properly trim adapter sequences from your assigned files. Be sure to read how to use `cutadapt`. Use default settings. What proportion of reads (both forward and reverse) were trimmed? For the 21_3G_both_S15_L008 dataset:

- Adapters trimmed from Read 1: 613,874 (6.6% of read pairs)
- Adapters trimeed from Read 2: 679,275 (7.4% of read pairs)

For the 4_2C_mbnl_S4_L008 dataset:

- Adapters trimmed from Read 1: 570,274 (6.2% of read pairs)
- Adapters trimeed from Read 2: 637,307 (6.9% of read pairs)

Cutadapt script:

```
#!/bin/bash

#SBATCH --account=bgmp
#SBATCH --partition=bgmp
#SBATCH --cpus-per-task=8
#SBATCH --time=10:00:00
#SBATCH --output=cutadapt_%j.out
#SBATCH --error=cutadapt_%j.err

conda activate QAA

input_dir="/projects/bgmp/shared/2017_sequencing/demultiplexed/"
dataset1_r1="21_3G_both_S15_L008_R1_001.fastq.gz"
dataset1_r2="21_3G_both_S15_L008_R2_001.fastq.gz"
dataset2_r1="4_2C_mbnl_S4_L008_R1_001.fastq.gz"
dataset2_r2="4_2C_mbnl_S4_L008_R2_001.fastq.gz"
r1_adapter="AGATCGGAAGAGCACACGTCTGAACTCCAGTCA"
r2_adapter="AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT"
output_dir="/projects/bgmp/tnguye14/bioinfo/Bi623/QAA/pt2/cutadapt_output/"
output_file1="${output_dir}adapter_trimmed_${dataset1_r1}"
output_file2="${output_dir}adapter_trimmed_${dataset1_r2}"
output_file3="${output_dir}adapter_trimmed_${dataset2_r1}"
output_file4="${output_dir}adapter_trimmed_${dataset2_r2}"

# cut read 1 and read 2 adapter sequences from the dataset 1 paired-end files
/usr/bin/time -v cutadapt -a $r1_adapter -A $r2_adapter \
-o $output_file1 -p $output_file2 \
${input_dir}$dataset1_r1 ${input_dir}$dataset1_r2

# cut read 1 and read 2 adapter sequences from the dataset 2 paired-end files
/usr/bin/time -v cutadapt -a $r1_adapter -A $r2_adapter \
-o $output_file3 -p $output_file4 \
${input_dir}$dataset2_r1 ${input_dir}$dataset2_r2

exit
```

Sanity check: Use your Unix skills to search for the adapter sequences in your datasets and confirm the expected sequence orientations. Report the commands you used, the reasoning behind them, and how you confirmed the adapter sequences.

Running a `zcat $fastq_gz_file | grep --color='auto' $adapter_seq` command for the R1 adapter in the R1 read files shows that the R1 adapter tends to be present closer to the 3' end of the read sequence. The same trend occurs when running a similar command to find R2 adapters in the R2 read files.

In addition, as shown below, the R1 adapter sequence is present in forward orientation (5'- AGATCGGAAGAGCACACGTCTGAACTCCAGTCA -3') for 66732 of the 21_3G_both_S15_L008_R1 reads and 51173 of the 4_2C_mbnl_S4_L008_R1 reads. The R2 adapter sequence is present in forward orientation (5'- AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT -3') for 67707 of the 21_3G_both_S15_L008_R1 reads and 51593 of the 4_2C_mbnl_S4_L008_R1 reads.

However, the reverse orientation (5'- ACTGACCTCAAGTCTGCACACGAGAAGGCTAGA -3') of the R1 adapter sequence was not present in any of the R1 reads in either dataset. In addition, the reverse orientation (5'- TGTGAGAAAGGGATGTGCTGCGAGAAGGCTAGA -3') of the R2 adapter sequence was not present in any of the R2 reads in either dataset. Thus, the R1 adapter sequence is 5'- AGATCGGAAGAGCACACGTCTGAACTCCAGTCA -3' and the R2 adapter sequence is 5'- AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT -3'.

Checking for the presence of the R2 adapter sequence in the R1 reads (and vice versa) also yielded counts of zero, indicating that the R1 adapters were correctly applied to the R1 reads and the R2 adapters were correctly applied to the R2 reads.

Bash script to count how many times each potential orientation (forward or reverse) of the adapter sequences occurs in the read sequences:

```
#!/bin/bash

#SBATCH --account=bgmp
#SBATCH --partition=bgmp
#SBATCH --cpus-per-task=8
#SBATCH --nodes=1
#SBATCH --time=10:00:00
#SBATCH --output=confirm_%j.out
#SBATCH --error=confirm_%j.err

set1_r1="/projects/bgmp/shared/2017_sequencing/demultiplexed/21_3G_both_S15_L008_R1_001.fastq.gz"
set1_r2="/projects/bgmp/shared/2017_sequencing/demultiplexed/21_3G_both_S15_L008_R2_001.fastq.gz"
set2_r1="/projects/bgmp/shared/2017_sequencing/demultiplexed/4_2C_mbnl_S4_L008_R1_001.fastq.gz"
set2_r2="/projects/bgmp/shared/2017_sequencing/demultiplexed/4_2C_mbnl_S4_L008_R2_001.fastq.gz"
adapter_r1="AGATCGGAAGAGCACACGTCTGAACTCCAGTCA"
adapter_r1_rev="ACTGACCTCAAGTCTGCACACGAGAAGGCTAGA"
adapter_r2="AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT"
adapter_r2_rev="TGTGAGAAAGGGATGTGCTGCGAGAAGGCTAGA"

# search for R1 adapter forward sequence in R1 files
echo "21_3G_both_S15_L008_R1 forward R1 adapters"
zcat $set1_r1 | grep -c $adapter_r1
echo "4_2C_mbnl_S4_L008_R1 forward R1 adapters"
zcat $set2_r1 | grep -c $adapter_r1

# search for R2 adapter forward sequence in R2 files
echo "21_3G_both_S15_L008_R2 forward R2 adapters"
zcat $set1_r2 | grep -c $adapter_r2
echo "4_2C_mbnl_S4_L008_R2 forward R2 adapters"
zcat $set2_r2 | grep -c $adapter_r2

# search for reverse of R1 adapter sequence in R1 files
```

```
echo "21_3G_both_S15_L008_R1 rev R1 adapters"
zcat $set1_r1 | grep -c $adapter_r1_rev
echo "4_2C_mbnl_S4_L008_R1 rev R1 adapters"
zcat $set2_r1 | grep -c $adapter_r1_rev

# search for reverse of R2 adapter sequence in R2 files
echo "21_3G_both_S15_L008_R2 rev R2 adapters"
zcat $set1_r2 | grep -c $adapter_r2_rev
echo "4_2C_mbnl_S4_L008_R2 rev R2 adapters"
zcat $set2_r2 | grep -c $adapter_r2_rev

# search for R2 adapter forward sequence in R1 files
echo "21_3G_both_S15_L008_R1, count of R2 forward adapters"
zcat $set1_r1 | grep -c $adapter_r2
echo "4_2C_mbnl_S4_L008_R1 count of R2 forward adapters"
zcat $set2_r1 | grep -c $adapter_r2

# search for R1 adapter forward sequence in R2 files
echo "21_3G_both_S15_L008_R2 count of R1 forward adapters"
zcat $set1_r2 | grep -c $adapter_r1
echo "4_2C_mbnl_S4_L008_R2 count of R1 forward adapters"
zcat $set2_r2 | grep -c $adapter_r1
```

Output of running the above script:

```
21_3G_both_S15_L008_R1 forward R1 adapters
66732
4_2C_mbnl_S4_L008_R1 forward R1 adapters
51173
21_3G_both_S15_L008_R2 forward R2 adapters
67707
4_2C_mbnl_S4_L008_R2 forward R2 adapters
51593
21_3G_both_S15_L008_R1 rev R1 adapters
0
4_2C_mbnl_S4_L008_R1 rev R1 adapters
0
21_3G_both_S15_L008_R2 rev R2 adapters
0
4_2C_mbnl_S4_L008_R2 rev R2 adapters
0
21_3G_both_S15_L008_R1, count of R2 forward adapters
0
4_2C_mbnl_S4_L008_R1 count of R2 forward adapters
0
21_3G_both_S15_L008_R2 count of R1 forward adapters
0
4_2C_mbnl_S4_L008_R2 count of R1 forward adapters
0
```

6. Use `Trimmomatic` to quality trim your reads. Specify the following, in this order:

- **LEADING: quality of 3**

- **TRAILING: quality of 3**
- **SLIDING WINDOW: window size of 5 and required quality of 15**
- **MINLENGTH: 35 bases**

Be sure to output compressed files and clear out any intermediate files.

Trimmomatic script:

```
#!/bin/bash

#SBATCH --account=bgmp
#SBATCH --partition=bgmp
#SBATCH --cpus-per-task=8
#SBATCH --time=10:00:00
#SBATCH --output=trimmomatic_%j.out
#SBATCH --error=trimmomatic_%j.err

conda activate QAA

# if trimmomatic.jar path is unknown, run:
# find /projects/bgmp/tnguye14/miniconda3/envs/QAA/ -name "trimmomatic.jar" -print

trimmomatic_jar_path="/projects/bgmp/tnguye14/miniconda3/envs/QAA/share/trimmomatic-0.39-2/trimmomatic.
input_dir="/projects/bgmp/tnguye14/bioinfo/Bi623/QAA/pt2/cutadapt_output/"
output_dir="/projects/bgmp/tnguye14/bioinfo/Bi623/QAA/pt2/trimmomatic_output/"
adapter_trimmed_dataset1_R1="${input_dir}adapter_trimmed_21_3G_both_S15_L008_R1_001.fastq.gz"
adapter_trimmed_dataset1_R2="${input_dir}adapter_trimmed_21_3G_both_S15_L008_R2_001.fastq.gz"
adapter_trimmed_dataset2_R1="${input_dir}adapter_trimmed_4_2C_mbnl_S4_L008_R1_001.fastq.gz"
adapter_trimmed_dataset2_R2="${input_dir}adapter_trimmed_4_2C_mbnl_S4_L008_R2_001.fastq.gz"
output_dataset1_template="${output_dir}q_and_adapter_trimmed_21_3G_both_S15_L008.fastq.gz"
output_dataset2_template="${output_dir}q_and_adapter_trimmed_4_2C_mbnl_S4_L008.fastq.gz"
dataset1_trimlog="${output_dir}trimlog_21_3G_both_S15_L008.txt"
dataset2_trimlog="${output_dir}trimlog_4_2C_mbnl_S4_L008.txt"

#run trimmomatic in paired-end (PE) mode on each dataset
/usr/bin/time -v java -jar $trimmomatic_jar_path \
PE \
$adapter_trimmed_dataset1_R1 $adapter_trimmed_dataset1_R2 \
-baseout $output_dataset1_template \
-trimlog $dataset1_trimlog \
-threads 4 \
LEADING:3 \
TRAILING:3 \
SLIDINGWINDOW:5:15 \
MINLEN:35

/usr/bin/time -v java -jar $trimmomatic_jar_path \
PE \
$adapter_trimmed_dataset2_R1 $adapter_trimmed_dataset2_R2 \
-baseout $output_dataset2_template \
-trimlog $dataset2_trimlog \
-threads 4 \
LEADING:3 \
TRAILING:3 \
```

```
SLIDINGWINDOW:5:15 \
MINLEN:35

exit
```

7. Plot the trimmed read length distributions for both R1 and R2 reads (on the same plot). You can produce 2 different plots for your 2 different RNA-seq samples. There are a number of ways you could possibly do this. One useful thing your plot should show, for example, is whether R1s are trimmed more extensively than R2s, or vice versa. Comment on whether you expect R1s and R2s to be adapter-trimmed at different rates. R2 reads are expected to be trimmed at higher rates than R1 reads. This may be due to degradation of R2 libraries and/or sequencing reagents because they've been on the sequencer flowcell for a longer period of time.

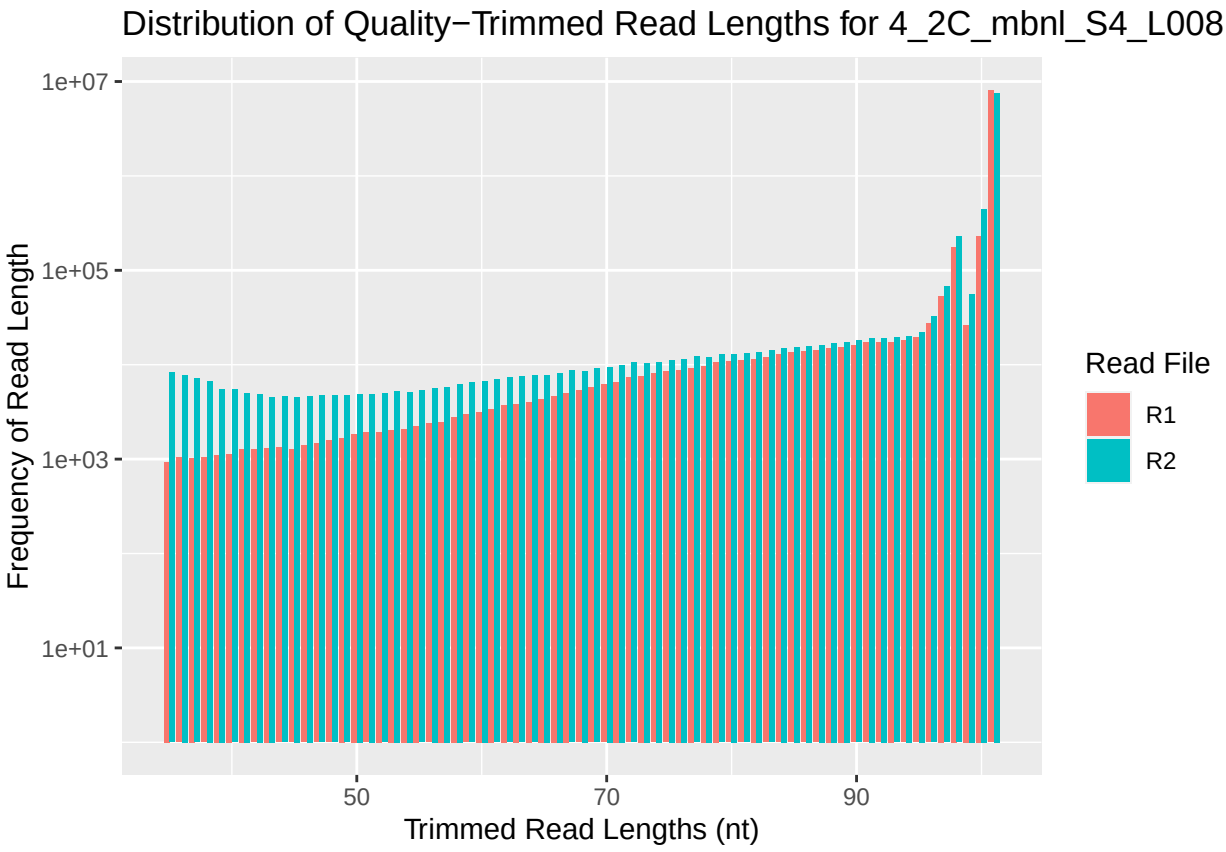

Table 1: Trimmed read length distribution for R1 and R2 reads in 21_3G_both_S15_L008

Trimmed_read_length	Frequency_R1	Frequency_R2
35	873	9070
36	981	8763
37	1091	8359
38	1130	7727
39	1196	6791
40	1327	6679
41	1353	6350
42	1329	6188
43	1392	5725
44	1485	5693
45	1532	5765
46	1579	5600
47	1679	5764
48	1846	5904
49	2035	6141
50	2174	6005
51	2302	6204
52	2464	6422
53	2639	6458
54	2709	6597
55	2948	7010

Trimmed_read_length	Frequency_R1	Frequency_R2
56	3279	7360
57	3415	7563
58	3672	8164
59	4111	8471
60	4388	8820
61	4656	9301
62	4845	9501
63	5331	10044
64	5641	10074
65	5731	10375
66	5958	10447
67	6632	11030
68	6868	11004
69	7161	11418
70	7616	11761
71	7944	12258
72	8412	12475
73	8867	12443
74	9205	12796
75	9437	12974
76	9833	13171
77	10374	14088
78	10698	13618
79	10838	13727
80	11194	14244
81	11752	14423
82	11785	14581
83	12171	14941
84	12745	15505
85	13231	15968
86	13594	16311
87	13933	16794
88	14427	17150
89	14561	17340
90	15069	17978
91	15936	18747
92	15595	18448
93	16197	19033
94	16707	19491
95	17962	21433
96	25849	31139
97	52602	66513
98	173175	225278
99	29406	64584
100	247703	468720
101	7896745	7322596

Table 2: Trimmed read length distribution for R1 and R2 reads in 4_2C_mbnl_S4_L008

Trimmed_read_length	Frequency_R1	Frequency_R2
35	937	8291
36	1042	7741
37	1021	7174
38	1038	6672
39	1099	5537
40	1137	5468
41	1273	5035
42	1265	4882
43	1307	4530
44	1331	4611
45	1265	4546
46	1395	4646
47	1457	4719
48	1571	4707
49	1677	4718
50	1850	4893
51	1910	4917
52	1934	4950
53	2034	5241
54	2081	5156
55	2239	5342
56	2393	5644
57	2469	5789
58	2794	6280
59	2970	6499
60	3144	6754
61	3394	6967
62	3717	7347
63	3838	7504
64	4041	7716
65	4303	7732
66	4684	8116
67	5025	8683
68	5308	8607
69	5706	9084
70	6143	9456
71	6573	9951
72	7299	10583
73	7595	10441
74	8052	10696
75	8458	11167
76	8840	11486
77	9287	12169
78	9684	12016
79	10614	12861
80	10862	12932
81	11128	13135
82	11483	13635
83	11968	14173

Trimmed_read_length	Frequency_R1	Frequency_R2
84	12787	14947
85	13410	15435
86	13786	15829
87	14170	16266
88	15019	17021
89	15329	17450
90	15988	18086
91	17187	19208
92	17246	19094
93	17458	19337
94	17978	20069
95	19442	22270
96	27383	32525
97	53452	67662
98	175379	227454
99	26223	56433
100	231667	444599
101	8063842	7555497

Part 3 – Alignment and strand-specificity

8. Install sofware. In your QAA environment, use conda to install:

```
- star
- numpy
- pysam
- matplotlib

Then ‘‘‘pip install HTSeq‘‘‘
```

Commands:

```
conda activate QAA
conda install star -c bioconda
conda install numpy pysam matplotlib
pip install HTSeq
```

9. Find publicly available mouse genome fasta files (Ensemble release 104) and generate an alignment database from them. Align the reads to your mouse genomic database using a splice-aware aligner. Use the settings specified in PS8 from Bi621.

- **Hint - you will need to use gene models to perform splice-aware alignment, see PS8 from Bi621.**

Mouse genome used: **Mus musculus**

Mouse genome and GTF files used to generate alignment reference database to which the 4_2C_mbnl_S4_L008 and 21_3G_both_S15_L008 datasets will be aligned:

- Mus_musculus.GRCm39.dna.primary_assembly.fa.gz
- Mus_musculus.GRCm39.104.gtf.gz

Script for building alignment reference database with STAR:

```
#!/bin/bash

#SBATCH --account=bgmp
#SBATCH --partition=bgmp
#SBATCH --cpus-per-task=8
#SBATCH --time=10:00:00
#SBATCH --output=build_star_db%j.out
#SBATCH --error=build_star_db%j.err

conda activate QAA

my_dir="/projects/bgmp/tnguye14/bioinfo/Bi623/QAA/pt3/"
genome_dir="${my_dir}Mus_musculus.GRCm39.dna.ens104.STAR_2.7.9a/"
genome_fasta_file_gz="${my_dir}Mus_musculus.GRCm39.dna.primary_assembly.fa.gz"
GTF_file_gz="${my_dir}Mus_musculus.GRCm39.104.gtf.gz"
genome_fasta_file_unzipped="${my_dir}Mus_musculus.GRCm39.dna.primary_assembly.fa"
GTF_file_unzipped="${my_dir}Mus_musculus.GRCm39.104.gtf"

#unzip the files to build STAR database
/usr/bin/time -v gunzip $genome_fasta_file_gz $GTF_file_gz

/usr/bin/time -v STAR --runThreadN 8 \
--runMode genomeGenerate \
--genomeDir $genome_dir \
--genomeFastaFiles $genome_fasta_file_unzipped \
--sjdbGTFfile $GTF_file_unzipped

#zip the unzipped files back up to save storage space
/usr/bin/time -v gzip $genome_fasta_file_unzipped $GTF_file_unzipped

exit
```

Script for performing STAR alignments of adapter- and quality-trimmed R1 and R2 reads from 4_2C_mbnl_S4_L008:

```
#!/bin/bash

#SBATCH --account=bgmp
#SBATCH --partition=bgmp
#SBATCH --cpus-per-task=8
#SBATCH --nodes=1
#SBATCH --time=10:00:00
#SBATCH --output=star_align1_%j.out
#SBATCH --error=star_align1_%j.err

conda activate QAA

my_dir="/projects/bgmp/tnguye14/bioinfo/Bi623/QAA/pt3/"
```

```
input_read_file1="/projects/bgmp/tnguye14/bioinfo/Bi623/QAA/pt2/trimmomatic_output/q_and_adapter_trimme
input_read_file2="/projects/bgmp/tnguye14/bioinfo/Bi623/QAA/pt2/trimmomatic_output/q_and_adapter_trimme
genome_dir="${my_dir}Mus_musculus.GRCm39.dna.ens104.STAR_2.7.9a"
output_filename_prefix="${my_dir}star_align_output/star_4_2C_mbnl_S4_L008_"

/usr/bin/time -v STAR --runThreadN 8 \
--runMode alignReads \
--outFilterMultimapNmax 3 \
--outSAMunmapped Within KeepPairs \
--alignIntronMax 1000000 --alignMatesGapMax 1000000 \
--readFilesCommand zcat \
--readFilesIn $input_read_file1 $input_read_file2 \
--genomeDir $genome_dir \
--outFileNamePrefix $output_filename_prefix

exit
```

Script for performing STAR alignments of adapter- and quality-trimmed R1 and R2 reads from 21_3G_both_S15_L008:

```
#!/bin/bash

#SBATCH --account=bgmp
#SBATCH --partition=bgmp
#SBATCH --cpus-per-task=8
#SBATCH --nodes=1
#SBATCH --time=10:00:00
#SBATCH --output=star_align2_%j.out
#SBATCH --error=star_align2_%j.err

conda activate QAA

my_dir="/projects/bgmp/tnguye14/bioinfo/Bi623/QAA/pt3/"
input_read_file1="/projects/bgmp/tnguye14/bioinfo/Bi623/QAA/pt2/trimmomatic_output/q_and_adapter_trimme
input_read_file2="/projects/bgmp/tnguye14/bioinfo/Bi623/QAA/pt2/trimmomatic_output/q_and_adapter_trimme
genome_dir="${my_dir}Mus_musculus.GRCm39.dna.ens104.STAR_2.7.9a"
output_filename_prefix="${my_dir}star_align_output/star_21_3G_both_S15_L008_"

/usr/bin/time -v STAR --runThreadN 8 \
--runMode alignReads \
--outFilterMultimapNmax 3 \
--outSAMunmapped Within KeepPairs \
--alignIntronMax 1000000 --alignMatesGapMax 1000000 \
--readFilesCommand zcat \
--readFilesIn $input_read_file1 $input_read_file2 \
--genomeDir $genome_dir \
--outFileNamePrefix $output_filename_prefix

exit
```

10. Using your script from PS8 in Bi621, report the number of mapped and unmapped reads from each of your 2 sam files. Make sure that your script is looking at the bitwise flag to

determine if reads are primary or secondary mapping (update your script if necessary). For `star_4_2C_mbnl_S4_L008_Aligned.out.sam` file:

- Mapped count: 17172669
- Unmapped count: 788093

For `star_21_3G_both_S15_L008_Aligned.out.sam` file:

- Mapped count: 17061136
- Unmapped count: 645494

Python script for counting number of mapped and unmapped reads from SAM files:

```
#!/bin/python
import argparse

def get_args():
    '''Defines/sets possible command line arguments for script'''
    parser = argparse.ArgumentParser("A program to parse SAM file and count number of mapped reads")
    parser.add_argument("-f", nargs="+", help="Specifies input filename(s). Must be SAM files.", type=s
    return parser.parse_args()

def get_mapped_unmapped_counts(filename: str):
    '''Get mapped and unmapped read counts from an input SAM file for reads that are not from a secondar
    #local variables
    mapped_count: int = 0
    unmapped_count: int = 0
    line_tokens: list = []
    bitwise_flag: int = 0
    mapped_count: int = 0
    unmapped_count: int = 0

    print(filename)

    with open(filename, 'r') as fh:
        for line in fh:
            if line.startswith("@"):
                continue    #go to next loop iteration and read next line in file
            else:
                line_tokens = line.split("\t")
                bitwise_flag = int(line_tokens[1])

                #if read is not unmapped (read is mapped) AND read is not from a secondary alignment (r
                if ((bitwise_flag & 4) != 4) and ((bitwise_flag & 256) != 256): #if read is mapped
                    #mapped = True
                    mapped_count += 1
                else:   #if read is unmapped
                    if ((bitwise_flag & 256) != 256): #if read is not from a secondary alignment (read
                        unmapped_count += 1

    print("Mapped count:", mapped_count)
    print("Unmapped count:", unmapped_count)
```

```
def main():
    '''Main function, drives the order of execution for script'''
    #local variables
    args = get_args()
    file_list: list = args.f

    for file in file_list:
        get_mapped_unmapped_counts(file)

if __name__ == "__main__":
    main()
```

Bash wrapper script for running the above SAM-parsing Python script in SLURM on Talapas:

```
#!/bin/bash

#SBATCH --account=bgmp
#SBATCH --partition=bgmp
#SBATCH --cpus-per-task=8
#SBATCH --time=10:00:00
#SBATCH --output=parse_SAM_%j.out
#SBATCH --error=parse_SAM_%j.err

conda activate QAA

script_file="/projects/bgmp/tnguye14/bioinfo/Bi623/QAA/pt3/parse_SAM_redux.py"
sam_file_1="/projects/bgmp/tnguye14/bioinfo/Bi623/QAA/pt3/star_align_output/star_4_2C_mbnl_S4_L008_Alig
sam_file_2="/projects/bgmp/tnguye14/bioinfo/Bi623/QAA/pt3/star_align_output/star_21_3G_both_S15_L008_Al

/usr/bin/time -v python $script_file -f $sam_file_1 $sam_file_2

exit
```

11. Count reads that map to features using htseq-count. You should run htseq-count twice: once with `--stranded=yes` and again with `--stranded=no`. Use default parameters otherwise. See below for first 20 lines and last 5 lines of htseq-count output for stranded and unstranded runs.

See pt3/ directory in QAA repository on Github for full htseq-count output files (55000+ lines each) from stranded and unstranded runs. Note that column 2 in each output file holds counts for the STAR-aligned 4_2C_mbnl_S4_L008 SAM file, and column 3 in each output file holds counts for the STAR-aligned 21_3G_both_S15_L008 SAM file.

Table 3: Htseq-count output (stranded setting), first 20 lines

Gene/Feature	Read count (4_2C_mbnl_S4_L008)	Read count (21_3G_both_S15_L008)
ENSMUSG00000000001	7	5
ENSMUSG00000000003	0	0
ENSMUSG00000000028	6	2
ENSMUSG00000000031	0	0
ENSMUSG00000000037	0	0
ENSMUSG00000000049	28	35
ENSMUSG00000000056	4	3

Gene/Feature	Read count (4_2C_mbnl_S4_L008)	Read count (21_3G_both_S15_L008)
ENSMUSG00000000058	1	1
ENSMUSG00000000078	4	2
ENSMUSG00000000085	9	12
ENSMUSG00000000088	5	1
ENSMUSG00000000093	2	0
ENSMUSG00000000094	0	0
ENSMUSG00000000103	0	0
ENSMUSG00000000120	3	2
ENSMUSG00000000125	0	1
ENSMUSG00000000126	0	1
ENSMUSG00000000127	1	5
ENSMUSG00000000131	4	2
ENSMUSG00000000134	5	7

Table 4: Htseq-count output (stranded setting), last 5 lines

Gene/Feature	Read count (4_2C_mbnl_S4_L008)	Read count (21_3G_both_S15_L008)
__no_feature	7807229	7806546
__ambiguous	5212	7215
__too_low_aQual	9632	12683
__not_aligned	388842	315936
__alignment_not_unique	420417	383458

Table 5: Htseq-count output (unstranded setting), first 20 lines

Gene/Feature	Read count (4_2C_mbnl_S4_L008)	Read count (21_3G_both_S15_L008)
ENSMUSG00000000001	1699	1644
ENSMUSG00000000003	0	0
ENSMUSG00000000028	841	794
ENSMUSG00000000031	0	0
ENSMUSG00000000037	0	0
ENSMUSG00000000049	0	1
ENSMUSG00000000056	217	201
ENSMUSG00000000058	532	570
ENSMUSG00000000078	1245	1327
ENSMUSG00000000085	485	484
ENSMUSG00000000088	293	271
ENSMUSG00000000093	35	35
ENSMUSG00000000094	0	0
ENSMUSG00000000103	0	0
ENSMUSG00000000120	3	3
ENSMUSG00000000125	0	2
ENSMUSG00000000126	107	80
ENSMUSG00000000127	430	427
ENSMUSG00000000131	718	793
ENSMUSG00000000134	655	681

Table 6: Htseq-count output (stranded setting), last 5 lines

Gene/Feature	Read count (4_2C_mbnl_S4_L008)	Read count (21_3G_both_S15_L008)
__no_feature	707441	759034
__ambiguous	409736	403046
__too_low_aQual	9632	12683
__not_aligned	388842	315936
__alignment_not_unique	420417	383458

Bash script for htseq-count (stranded setting):

```
#!/bin/bash

#SBATCH --account=bgmp
#SBATCH --partition=bgmp
#SBATCH --cpus-per-task=8
#SBATCH --time=10:00:00
#SBATCH --output=htseq_stranded_%j.out
#SBATCH --error=htseq_stranded_%j.err

conda activate QAA

#star_aligned_file1="/projects/bgmp/tnguye14/bioinfo/Bi623/QAA/pt3/star_align_output/star_4_2C_mbnl_S4_
#star_aligned_file2="/projects/bgmp/tnguye14/bioinfo/Bi623/QAA/pt3/star_align_output/star_21_3G_both_S1

star_aligned_file1="/projects/bgmp/tnguye14/bioinfo/Bi623/QAA/pt3/sorted_star_4_2C_mbnl_S4_L008_Aligned
star_aligned_file2="/projects/bgmp/tnguye14/bioinfo/Bi623/QAA/pt3/sorted_star_21_3G_both_S15_L008_Align
ref_genome_gtf_zipped="/projects/bgmp/tnguye14/bioinfo/Bi623/QAA/pt3/Mus_musculus.GRCm39.104.gtf.gz"
ref_genome_gtf_unzipped="/projects/bgmp/tnguye14/bioinfo/Bi623/QAA/pt3/Mus_musculus.GRCm39.104.gtf"

/usr/bin/time -v gunzip $ref_genome_gtf_zipped

/usr/bin/time -v htseq-count --stranded=yes \
$star_aligned_file1 $star_aligned_file2 \
$ref_genome_gtf_unzipped

/usr/bin/time -v gzip $ref_genome_gtf_unzipped

exit
```

Bash script for htseq-count (unstranded setting):

```
#!/bin/bash

#SBATCH --account=bgmp
#SBATCH --partition=bgmp
#SBATCH --cpus-per-task=8
#SBATCH --time=10:00:00
#SBATCH --output=htseq_unstranded_%j.out
#SBATCH --error=htseq_unstranded_%j.err
```

```
conda activate QAA

# star_aligned_file1="/projects/bgmp/tnguye14/bioinfo/Bi623/QAA/pt3/star_align_output/star_4_2C_mbnl_S4
# star_aligned_file2="/projects/bgmp/tnguye14/bioinfo/Bi623/QAA/pt3/star_align_output/star_21_3G_both_S

star_aligned_file1="/projects/bgmp/tnguye14/bioinfo/Bi623/QAA/pt3/sorted_star_4_2C_mbnl_S4_L008_Aligned
star_aligned_file2="/projects/bgmp/tnguye14/bioinfo/Bi623/QAA/pt3/sorted_star_21_3G_both_S15_L008_Align
ref_genome_gtf_zipped="/projects/bgmp/tnguye14/bioinfo/Bi623/QAA/pt3/Mus_musculus.GRCm39.104.gtf.gz"
ref_genome_gtf_unzipped="/projects/bgmp/tnguye14/bioinfo/Bi623/QAA/pt3/Mus_musculus.GRCm39.104.gtf"

/usr/bin/time -v gunzip $ref_genome_gtf_zipped

/usr/bin/time -v htseq-count --stranded=no \
$star_aligned_file1 $star_aligned_file2 \
$ref_genome_gtf_unzipped

/usr/bin/time -v gzip $ref_genome_gtf_unzipped

exit
```

12. Demonstrate convincingly whether or not the data are from "strand-specific" RNA-Seq libraries. Include any comands/scripts used. Briefly describe your evidence, using quantitative statements (e.g. "I propose that these data are/are not strand-specific, because X% of the reads are y, as opposed to z.").

- **Hint - recall ICA4 from Bi621.**

The data looks like it's from unstranded RNA-seq libraries because the mapped read percentages for unstranded htseq output (78.4411% for 4_2C_mbnl_S4_L008, 78.831% for 21_3G_both_S15_L008) are much higher than for stranded htseq output (3.8868% for 4_2C_mbnl_S4_L008, 3.69892% for 21_3G_both_S15_L008). See below for mapped and unmapped read percentages for each dataset, as well as the bash script used to generate this data.

Mapped and unmapped counts and percentages for each dataset:

```
***** 4_2C_mbnl_S4_L008 *****
* MAPPED COUNTS *
Stranded:  349049
Unstranded:  7044313
* UNMAPPED COUNTS *
Stranded:  8631332
Unstranded:  1936068
* PERCENT MAPPED *
Stranded:  3.8868
Unstranded:  78.4411

***** 21_3G_both_S15_L008 *****
* MAPPED COUNTS *
Stranded:  327477
Unstranded:  6979158
* UNMAPPED COUNTS *
Stranded:  8525838
Unstranded:  1874157
```

```
* PERCENT MAPPED *
Stranded:  3.69892
Unstranded:  78.831
```

Bash script used to generate mapped and unmapped count and percentage data from each dataset:

```
#!/bin/bash

#SBATCH --account=bgmp
#SBATCH --partition=bgmp
#SBATCH --cpus-per-task=8
#SBATCH --time=10:00:00
#SBATCH --output=get_mapped_stats_%j.out
#SBATCH --error=get_mapped_stats_%j.err

conda activate QAA

htseq_stranded_output="/projects/bgmp/tnguye14/bioinfo/Bi623/QAA/pt3/htseq_stranded_16213781.out"
htseq_unstranded_output="/projects/bgmp/tnguye14/bioinfo/Bi623/QAA/pt3/htseq_unstranded_16213805.out"

#get mapped counts for 4_2C_mbnl_S4_L008 from stranded and unstranded htseqcount outputs
echo "***** 4_2C_mbnl_S4_L008 *****"
echo "* MAPPED COUNTS *"
awk '$1~/^ENS/{sum+=$2} END {print "Stranded: ", sum}' $htseq_stranded_output
awk '$1~/^ENS/{sum+=$2} END {print "Unstranded: ", sum}' $htseq_unstranded_output

echo "* UNMAPPED COUNTS *"
awk '$1!~/^ENS/{sum+=$2} END {print "Stranded: ", sum}' $htseq_stranded_output
awk '$1!~/^ENS/{sum+=$2} END {print "Unstranded: ", sum}' $htseq_unstranded_output

echo "* PERCENT MAPPED *"
awk '$1~/^ENS/{mappedreads+=$2} {totalreads+=$2} END {print "Stranded: ", (mappedreads/totalreads)*100}
awk '$1~/^ENS/{mappedreads+=$2} {totalreads+=$2} END {print "Unstranded: ", (mappedreads/totalreads)*10

#get mapped counts for 21_3G_both_S15_L008 from stranded and unstranded htseqcount outputs
echo "***** 21_3G_both_S15_L008 *****"
echo "* MAPPED COUNTS *"
awk '$1~/^ENS/{sum+=$3} END {print "Stranded: ", sum}' $htseq_stranded_output
awk '$1~/^ENS/{sum+=$3} END {print "Unstranded: ", sum}' $htseq_unstranded_output

echo "* UNMAPPED COUNTS *"
awk '$1!~/^ENS/{sum+=$3} END {print "Stranded: ", sum}' $htseq_stranded_output
awk '$1!~/^ENS/{sum+=$3} END {print "Unstranded: ", sum}' $htseq_unstranded_output

echo "* PERCENT MAPPED *"
awk '$1~/^ENS/{mappedreads+=$3} {totalreads+=$3} END {print "Stranded: ", (mappedreads/totalreads)*100}
awk '$1~/^ENS/{mappedreads+=$3} {totalreads+=$3} END {print "Unstranded: ", (mappedreads/totalreads)*10

exit
```