RNA-Seq Quality Assessment

Sally Grindstaff

9/5/2021

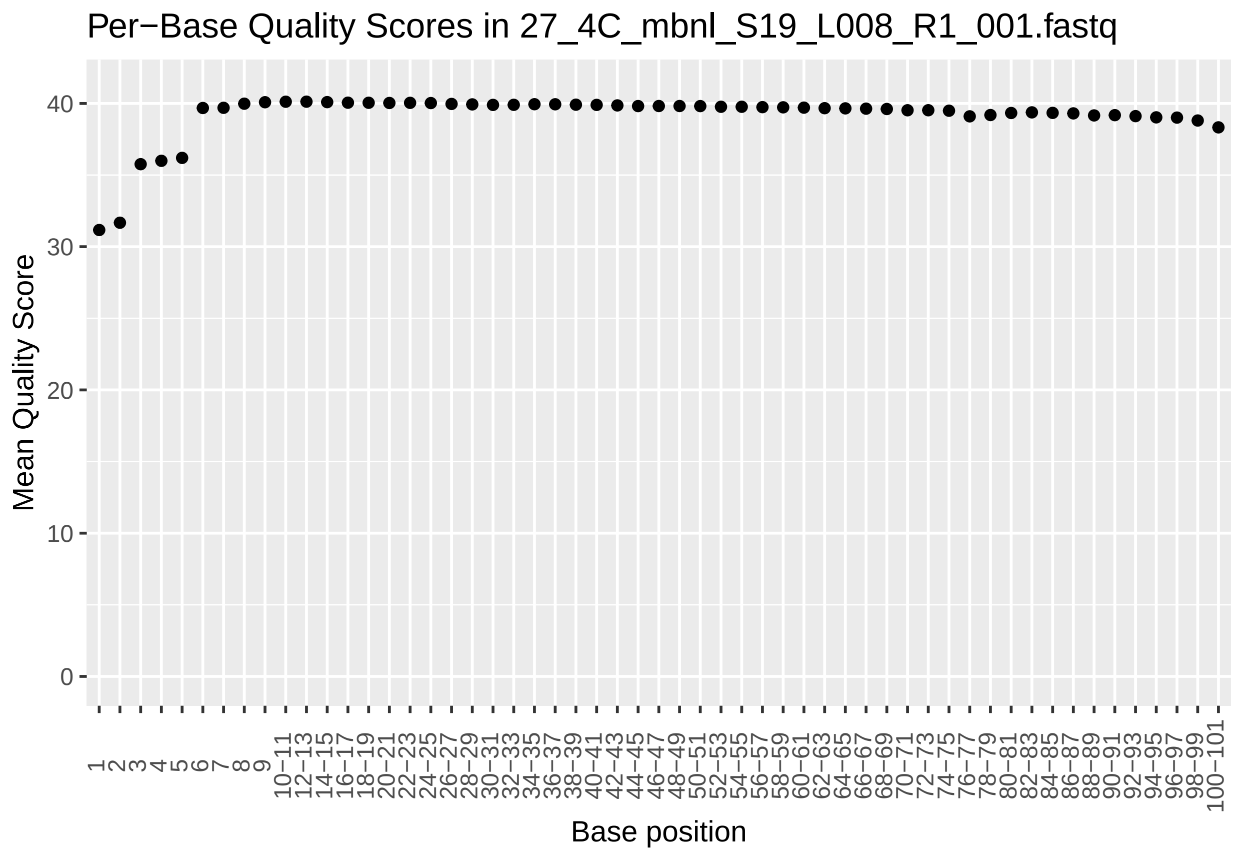
Part 1: Read quality score distributions

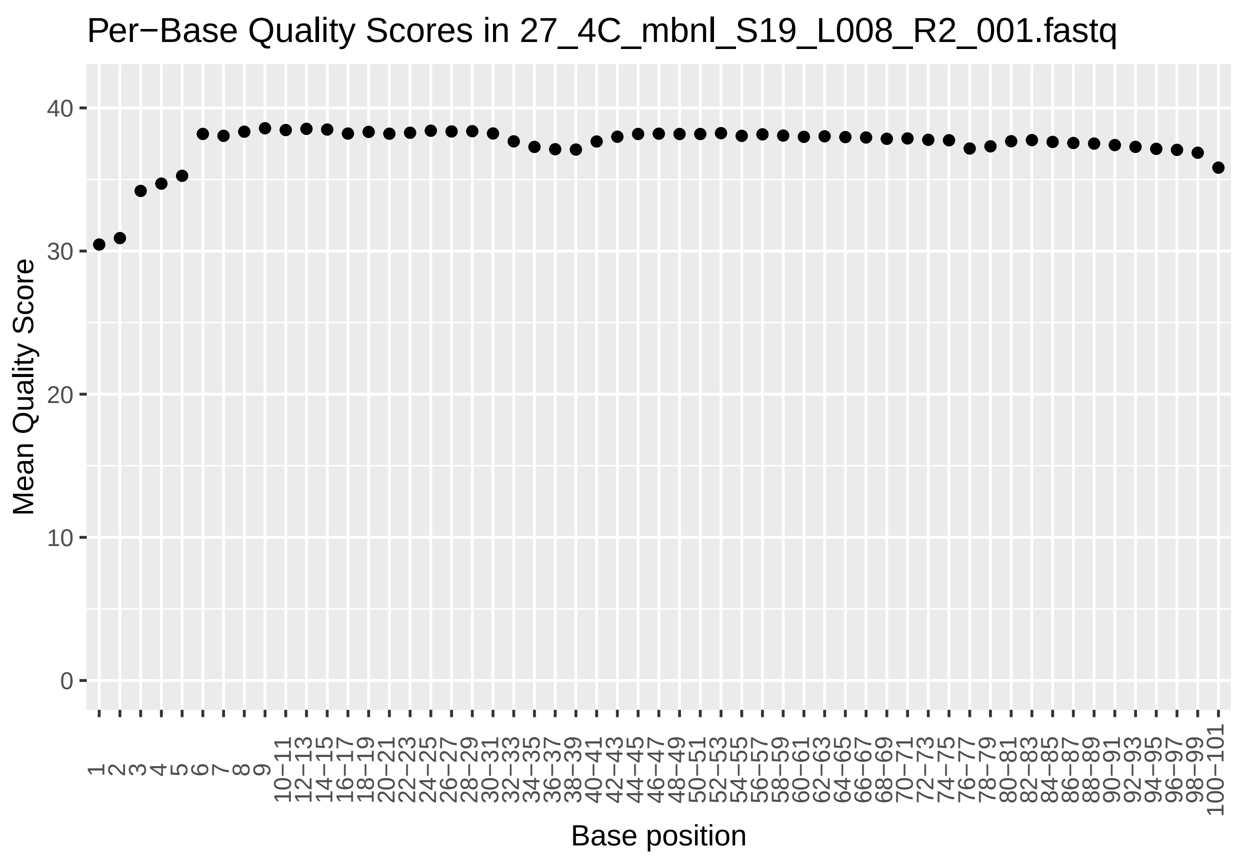
Plots of per-base quality scores and of per-base N content were generated using FASTQC version 0.11.5 as a module on Talapas, which was run by submitting the following script to Talapas' queuing system:

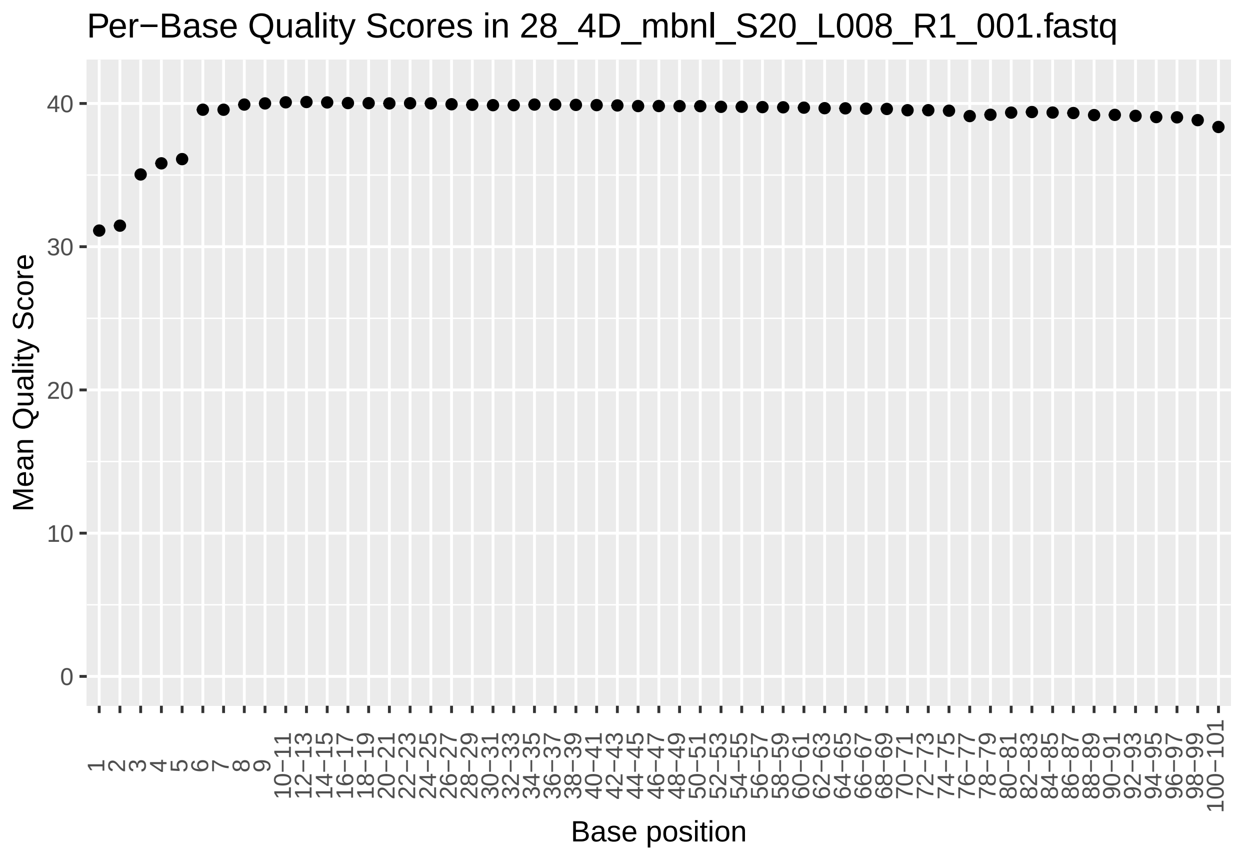
```
#!/bin/bash

#SBATCH --account=bgmp
#SBATCH --partition=bgmp
#SBATCH --cpus-per-task=1
#SBATCH --time=1-0:00:00

module load fastqc/0.11.5

input_dir=/projects/bgmp/shared/2017_sequencing/demultiplexed
output_dir=/projects/bgmp/sgrindst/bioinformatics/Bi623/QAA/fastqc_output/

/usr/bin/time -v fastqc $input_dir/27_4C_mbnl_S19_L008_R1_001.fastq.gz $input_dir/27_4C_mbnl_S19_L008_R

exit
```

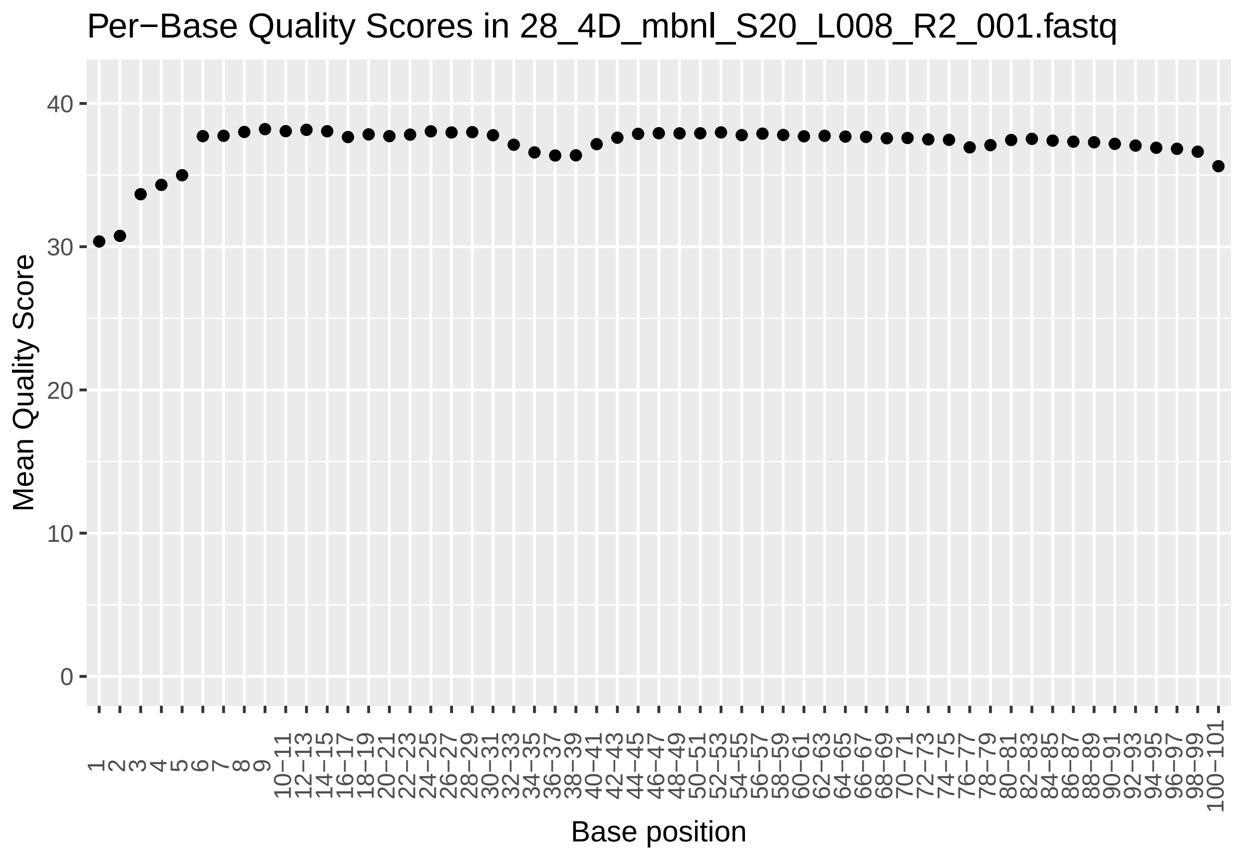
All plots from FASTQC have been re-plotted in R using the data points from FASTQC, for formatting purposes.

Per-base quality scores:

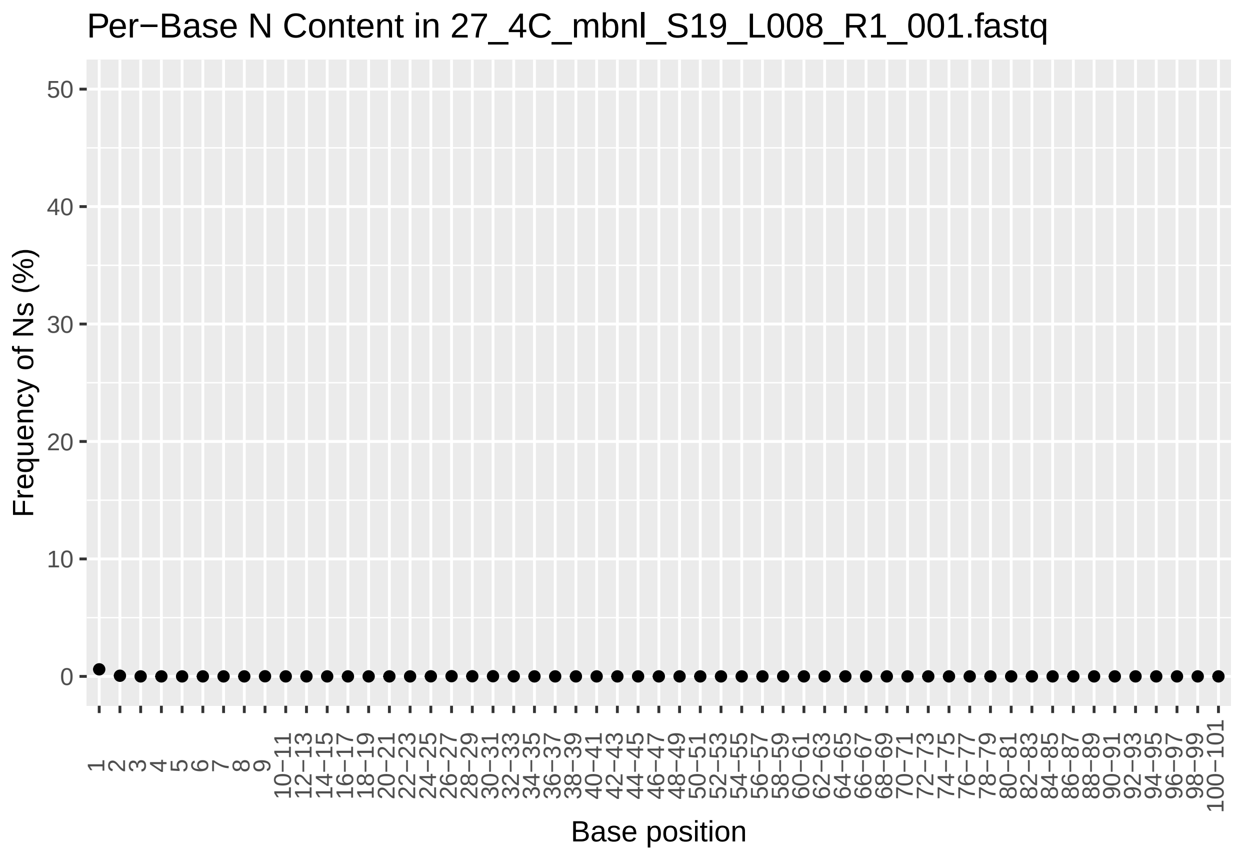

Per−Base Quality Scores in 27_4C_mbnl_S19_L008_R1_001.fastq
Mean Quality Score
0
10
20
30
40
Base position
1
2
3
4
5
6
7
8
9
10−11
12−13
14−15
16−17
18−19
20−21
22−23
24−25
26−27
28−29
30−31
32−33
34−35
36−37
38−39
40−41
42−43
44−45
46−47
48−49
50−51
52−53
54−55
56−57
58−59
60−61
62−63
64−65
66−67
68−69
70−71
72−73
74−75
76−77
78−79
80−81
82−83
84−85
86−87
88−89
90−91
92−93
94−95
96−97
98−99
100−101

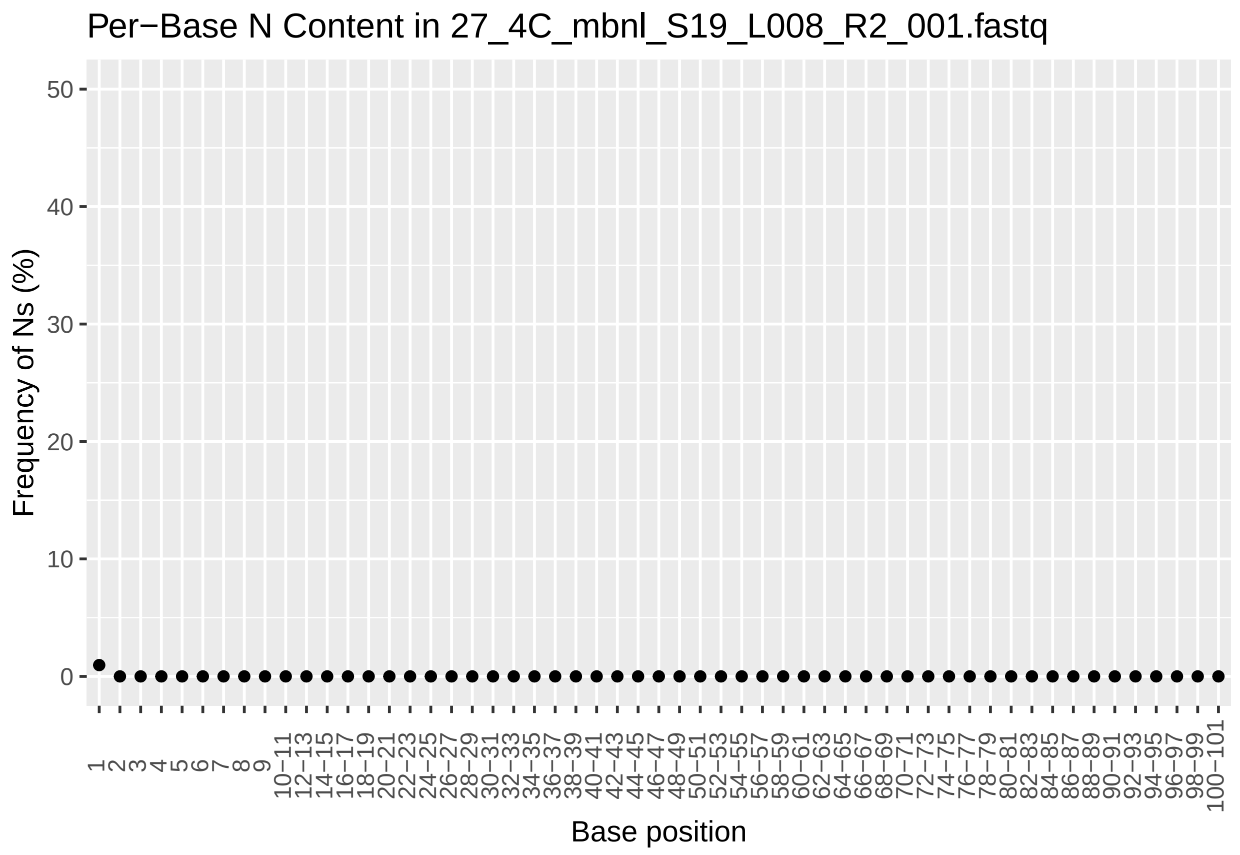

Per−Base Quality Scores in 27_4C_mbnl_S19_L008_R2_001.fastq
Mean Quality Score
Base position
0
10
20
30
40
1
2
3
4
5
6
7
8
9
10−11
12−13
14−15
16−17
18−19
20−21
22−23
24−25
26−27
28−29
30−31
32−33
34−35
36−37
38−39
40−41
42−43
44−45
46−47
48−49
50−51
52−53
54−55
56−57
58−59
60−61
62−63
64−65
66−67
68−69
70−71
72−73
74−75
76−77
78−79
80−81
82−83
84−85
86−87
88−89
90−91
92−93
94−95
96−97
98−99
100−101

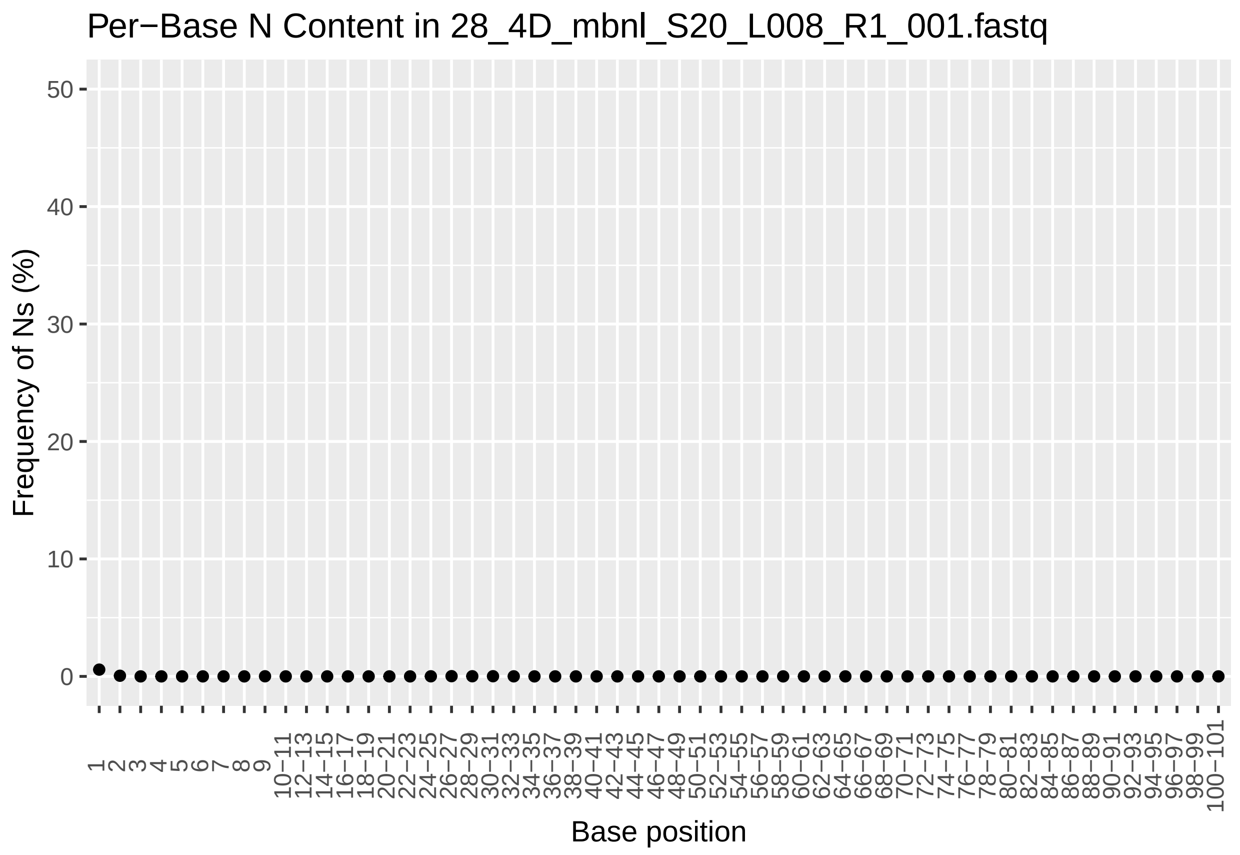

Per−Base Quality Scores in 28_4D_mbnl_S20_L008_R1_001.fastq
Mean Quality Score
0
10
20
30
40
Base position
1
2
3
4
5
6
7
8
9
10−11
12−13
14−15
16−17
18−19
20−21
22−23
24−25
26−27
28−29
30−31
32−33
34−35
36−37
38−39
40−41
42−43
44−45
46−47
48−49
50−51
52−53
54−55
56−57
58−59
60−61
62−63
64−65
66−67
68−69
70−71
72−73
74−75
76−77
78−79
80−81
82−83
84−85
86−87
88−89
90−91
92−93
94−95
96−97
98−99
100−101



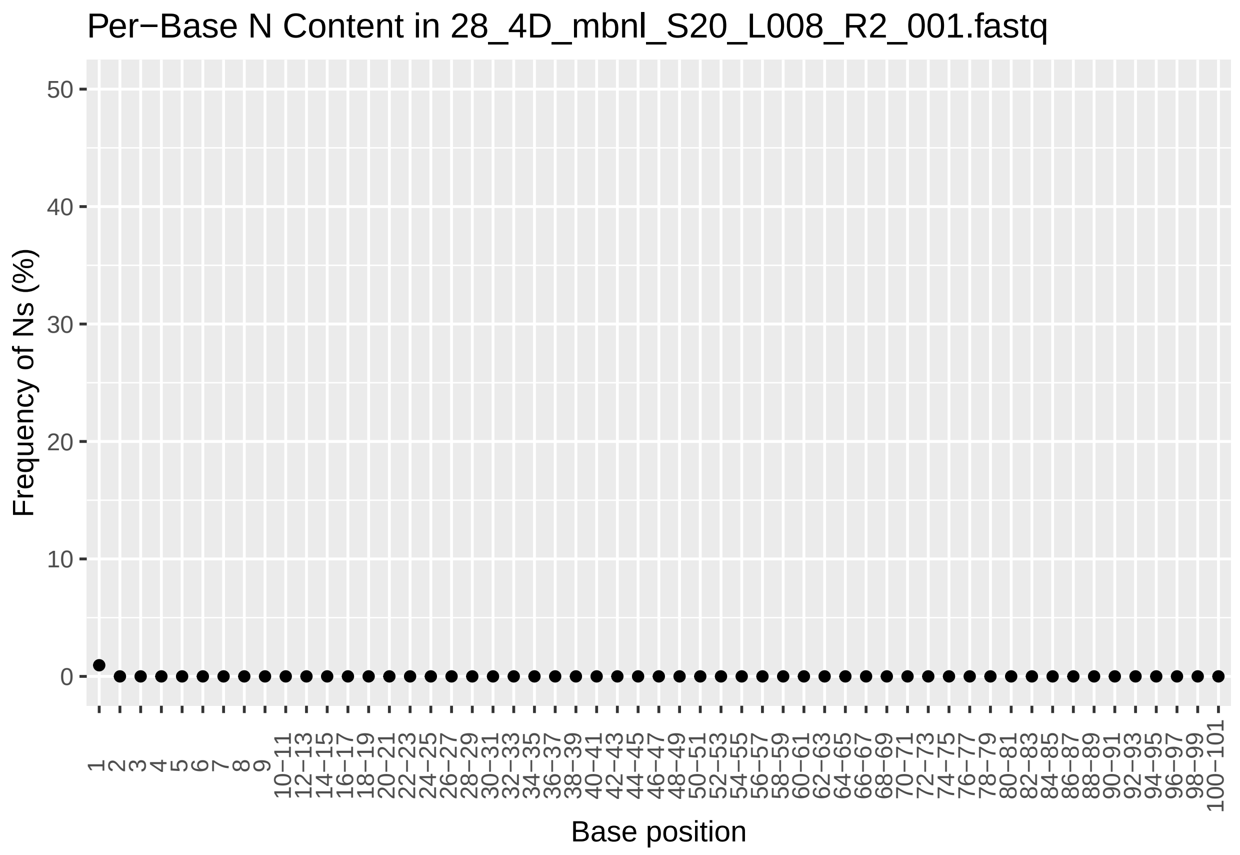
Per-base N content:


Per−Base N Content in 27_4C_mbnl_S19_L008_R1_001.fastq
Frequency of Ns (%)
0
10
20
30
40
50
Base position
1
2
3
4
5
6
7
8
9
10−11
12−13
14−15
16−17
18−19
20−21
22−23
24−25
26−27
28−29
30−31
32−33
34−35
36−37
38−39
40−41
42−43
44−45
46−47
48−49
50−51
52−53
54−55
56−57
58−59
60−61
62−63
64−65
66−67
68−69
70−71
72−73
74−75
76−77
78−79
80−81
82−83
84−85
86−87
88−89
90−91
92−93
94−95
96−97
98−99
100−101


Per−Base N Content in 27_4C_mbnl_S19_L008_R2_001.fastq
Frequency of Ns (%)
Base position
0
10
20
30
40
50
1
2
3
4
5
6
7
8
9
10−11
12−13
14−15
16−17
18−19
20−21
22−23
24−25
26−27
28−29
30−31
32−33
34−35
36−37
38−39
40−41
42−43
44−45
46−47
48−49
50−51
52−53
54−55
56−57
58−59
60−61
62−63
64−65
66−67
68−69
70−71
72−73
74−75
76−77
78−79
80−81
82−83
84−85
86−87
88−89
90−91
92−93
94−95
96−97
98−99
100−101



Per−Base N Content in 28_4D_mbnl_S20_L008_R1_001.fastq
Frequency of Ns (%)
50
40
30
20
10
0
1
2
3
4
5
6
7
8
9
10−11
12−13
14−15
16−17
18−19
20−21
22−23
24−25
26−27
28−29
30−31
32−33
34−35
36−37
38−39
40−41
42−43
44−45
46−47
48−49
50−51
52−53
54−55
56−57
58−59
60−61
62−63
64−65
66−67
68−69
70−71
72−73
74−75
76−77
78−79
80−81
82−83
84−85
86−87
88−89
90−91
92−93
94−95
96−97
98−99
100−101
Base position



For all 4 reads, the per-base N content is consistent with the per-base quality scores. Each quality score plot shows that the quality score starts lower and rises over the first 5 bases or so, after which it stabilizes. Each N content plots shows that the N content starts a little higher and then quickly drops down to around 0. So, both plots show that the N content and quality scores are slightly worse at the very beginning of the read. Also, the N content is very low for all 4 reads, and all 4 reads have decent quality scores (well above 35, and closer to 40 for the read 1 reads).

I also produced per-base quality score plots directly from the FASTQs using my own Python script. The Python script, followed by its dependent module Bioinfo.py and the bash scripts used to run the program, are shown here:

Bioinfo.py module:

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

def validate_base_seq(seq: str,RNAflag: bool =False) -> bool:
    '''This function takes a string. Returns True if string is composed
    of only As, Ts (or Us if RNAflag), Gs, Cs. False otherwise. Case insensitive.'''
    DNAbases = set('ATGCatcg')
    RNAbases = set('AUGCaucg')
    return set(seq)<=(RNAbases if RNAflag else DNAbases)

if __name__ == '__main__':
    assert validate_base_seq("AATAGAT") == True, "Validate base seq does not work on DNA"
    assert validate_base_seq("AAUAGAU", True) == True, "Validate base seq does not work on RNA"
    assert validate_base_seq("Hi there!") == False, "Validate base seq fails to recognize nonDNA"
    assert validate_base_seq("Hi there!", True) == False, "Validate base seq fails to recognize nonDNA"
    print("Passed DNA and RNA tests")

def convert_phred(letter: str) -> int:
    """Converts a single character into a phred score for quality scores encoded with Phred+33"""
    dec: int = ord(letter)
    phred: int = dec - 33
    return phred

if __name__ == '__main__':
    assert convert_phred('I') == 40, 'Wrong phred score'
    print('Converted character into phred score correctly')

def gc_content(DNA: str) -> int:
    '''Takes DNA (or RNA) sequence and returns GC content of the sequence in decimal format as a fracti
    GC_count = 0
    if validate_base_seq(DNA):
        DNA = DNA.upper()
        for letter in DNA:
            if letter == 'G' or letter == 'C':
                GC_count += 1
    else:
        print('Not a valid sequence')
    return GC_count/len(DNA)

if __name__ == '__main__':
    assert gc_content('ATGCGA') == 0.5, 'gc_content does not find correct GC content'
    print('Calculated gc content correctly')

def qual_score(phred_score: str) -> int:
    '''Calculates average quality score of a read, given a quality score string in Phred+33 notation'''
    sum: int = 0
    for letter in phred_score:
        sum += convert_phred(letter)
    return sum/len(phred_score)

if __name__ == '__main__':
    qual_score('JJII') == 40.5, 'Incorrect average phred quality score'
    print('Correct average phred quality score')

def fasta_fixer(fasta: str):
    '''Takes FASTA filename as input and outputs a FASTA file that has exactly 2 lines per record (one 
    output = fasta[0:-2] + 'fixed.fa'
    seq = ''
    with open(fasta, 'r') as fh, open(output, 'x') as out:
        for line in fh:
            line = line.strip()
            if line[0] == '>' and seq != '':
                out.write(header+'\n')
                out.write(seq+'\n')
                seq = ''
            if line[0] == '>':
                header = line
            else:
                seq += line
        out.write(header+'\n')
        out.write(seq+'\n')


DNA_bases = 'ATCGNatcgn'
RNA_bases = 'AUCGNaucgn'
```

Python script:

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

import argparse
import Bioinfo
from matplotlib import pyplot as plt
import gzip

def get_args():
    parser = argparse.ArgumentParser(description='Takes a zipped fastq file and outputs a png plot of t
    parser.add_argument('-i', '--input',help='Path to input zipped fastq file',required=True)
    parser.add_argument('-o','--output',help='Desired name (not including .png and not including any di
    parser.add_argument('-l','--length',help='Read length', required=True, type=int)
    return parser.parse_args()

args = get_args()

def init_list(lst: list, read_length: int, value: float=0.0) -> list:
    '''This function takes an empty list and will populate it with
    the value passed in "value". If no value is passed, initializes list
    with 101 values of 0.0.'''
    for i in range(read_length):
        lst.append(value)
    return lst

def populate_list(file, read_length):
    """Takes a FASTQ file and returns a list in which each item is the total sum of Q scores across all
    qscore_list = []
    qscore_list = init_list(qscore_list, read_length)
    with gzip.open(file, 'rt') as fh:
        num_lines = 0
        for line in fh:
            line = line.strip()
            num_lines += 1
            if num_lines % 4 == 0:
                i = 0
                for score in line:
                    qscore_list[i] += Bioinfo.convert_phred(score)
                    i += 1
        i = 0
        for score in qscore_list:
            qscore_list[i] = score/(num_lines/4)
            i +=1
    return qscore_list
```

```
read_list = populate_list(args.input, args.length)

def make_hist(some_list:list, name:str):
    plt.bar(range(len(some_list)), some_list)
    plt.xlabel('# Base Pair')
    plt.ylabel('Mean quality score')
    plt.title(f'Quality Score Distribution in {name}')
    plt.savefig(f'{name}_qscore_distribution.png')
    plt.close()

make_hist(read_list,args.output)
```

Scripts submitted to Talapas queue:

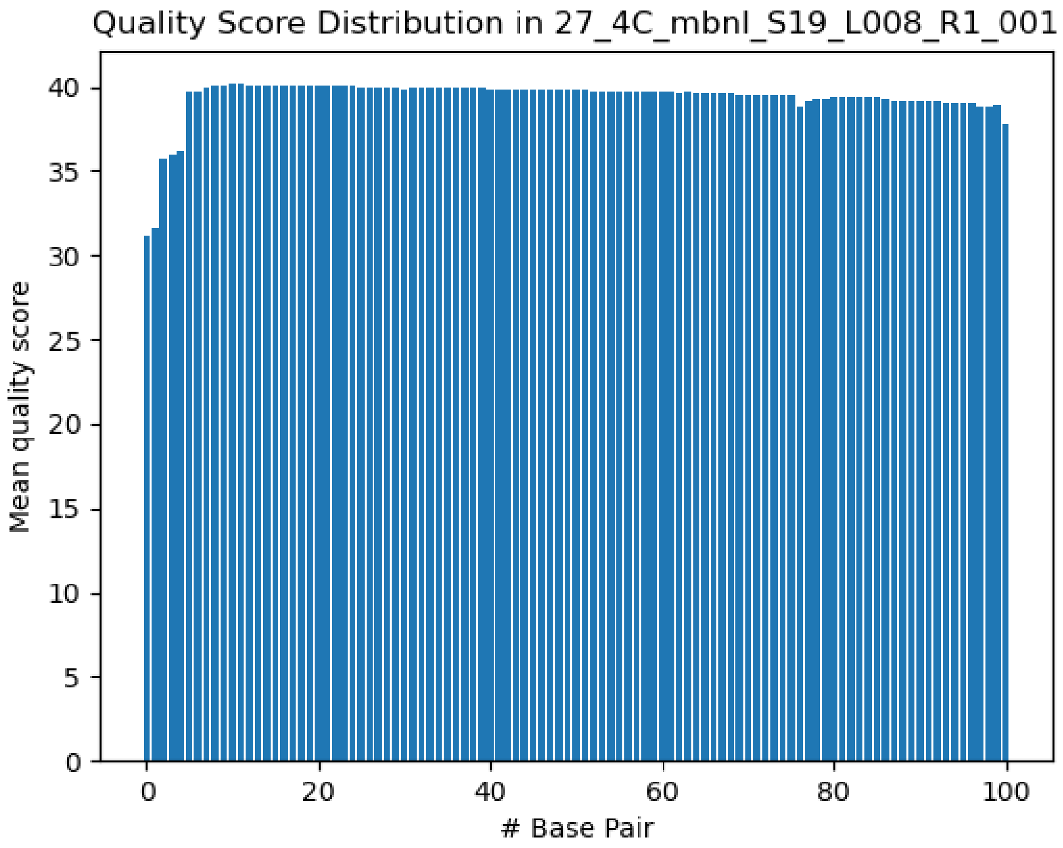
```
#!/bin/bash

#SBATCH --account=bgmp
#SBATCH --partition=bgmp
#SBATCH --cpus-per-task=1
#SBATCH --time=1-0:00:00

conda activate bgmp_py39

/usr/bin/time -v ./qual_scores.py -i /projects/bgmp/shared/2017_sequencing/demultiplexed/27_4C_mbnl_S19

exit
```

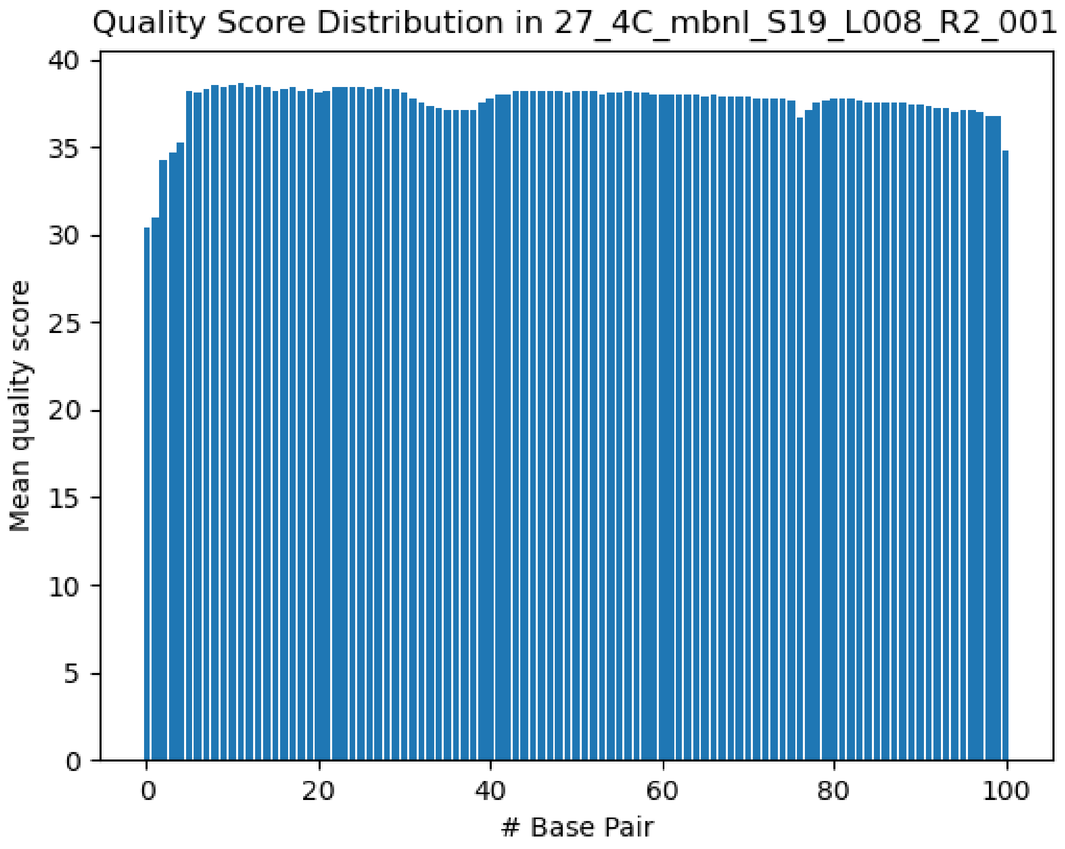
```
#!/bin/bash

#SBATCH --account=bgmp
#SBATCH --partition=bgmp
#SBATCH --cpus-per-task=1
#SBATCH --time=1-0:00:00

conda activate bgmp_py39

/usr/bin/time -v ./qual_scores.py -i /projects/bgmp/shared/2017_sequencing/demultiplexed/27_4C_mbnl_S19

exit
```

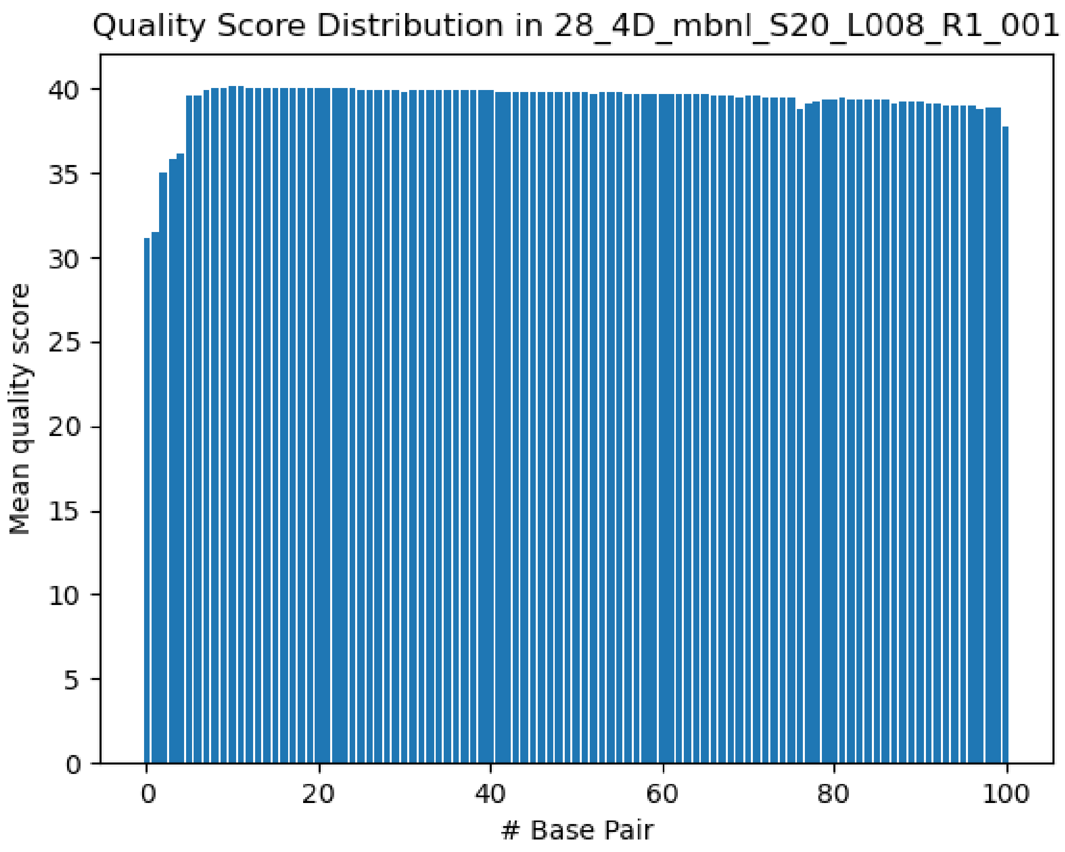
```
#!/bin/bash

#SBATCH --account=bgmp
#SBATCH --partition=bgmp
#SBATCH --cpus-per-task=1
#SBATCH --time=1-0:00:00

conda activate bgmp_py39

/usr/bin/time -v ./qual_scores.py -i /projects/bgmp/shared/2017_sequencing/demultiplexed/28_4D_mbnl_S20

exit
```

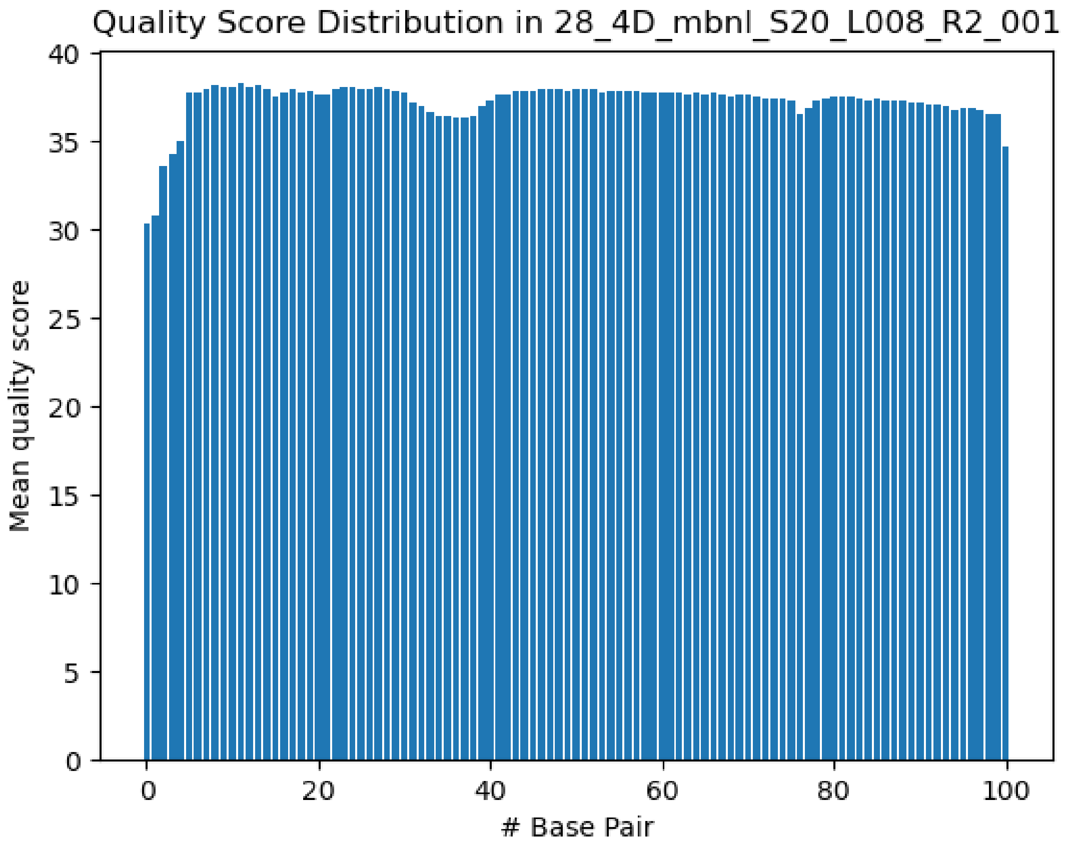
```
#!/bin/bash

#SBATCH --account=bgmp
#SBATCH --partition=bgmp
#SBATCH --cpus-per-task=1
#SBATCH --time=1-0:00:00

conda activate bgmp_py39

/usr/bin/time -v ./qual_scores.py -i /projects/bgmp/shared/2017_sequencing/demultiplexed/28_4D_mbnl_S20_

exit
```

My per-base quality score plots:




Quality Score Distribution in 27_4C_mbnl_S19_L008_R2_001
Mean quality score
# Base Pair
0
5
10
15
20
25
30
35
40
0
20
40
60
80
100


Quality Score Distribution in 28_4D_mbnl_S20_L008_R1_001
Mean quality score
# Base Pair
0
5
10
15
20
25
30
35
40
0
20
40
60
80
100



My quality score plots appear to match the quality score plots made with data produced by FASTQC. The only difference I can see is that my plots have more data points, because I calculated the mean quality score at each base position, while FASTQC calculated the mean across two bases at a time. This does not affect the overall shape of the quality score distributions.

Overall, the data quality of both libraries looks sufficient for downstream analysis. N-content is low across both reads in both libraries, with only slightly raised values at the beginning of each read. Read 1 quality scores are near 40 across the bulk of the read for both libraries. Read 2 quality scores are slightly lower, closer to 38, but this is expected for libraries sequenced with Illumina, because the sequencing molecule degrades as it sits on the flow cell and is repeatedly washed with harsh chemicals during read 1. FASTQC flags k-mer content and duplication levels as being potential problems for these libraries, but I know that these are RNA-seq libraries and are therefore expected to have differing levels of expression across the genome and to have exact copies of the same transcript due to the overexpression of some transcripts.

Part 2: Adapter trimming comparison

Adapter sequences were trimmed using cutadapt version 3.4. The adapter sequences were identified by searching for Illumina TruSeq adapter sequences on Illumina's website, and adapter presence and orientation was confirmed using the following bash commands in an interactive session on Talapas:

The following bash script was submitted to Talapas to run cutadapt:

```
#!/bin/bash

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

```
#SBATCH --cpus-per-task=1
#SBATCH --time=1-0:00:00

conda activate QAA

in_dir=/projects/bgmp/shared/2017_sequencing/demultiplexed
out_dir=/projects/bgmp/sgrindst/bioinformatics/Bi623/QAA/cutadapt

/usr/bin/time -v cutadapt -a AGATCGGAAGAGCACACGTCTGAACTCCAGTCA -A AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT \
-o $out_dir/27_4C_mbnl_S19_L008_R1_001.adaptertrimmed.fastq.gz \
-p $out_dir/27_4C_mbnl_S19_L008_R2_001.adaptertrimmed.fastq.gz \
$in_dir/27_4C_mbnl_S19_L008_R1_001.fastq.gz $in_dir/27_4C_mbnl_S19_L008_R2_001.fastq.gz

/usr/bin/time -v cutadapt -a AGATCGGAAGAGCACACGTCTGAACTCCAGTCA -A AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT \
-o $out_dir/28_4D_mbnl_S20_L008_R1_001.adaptertrimmed.fastq.gz \
-p $out_dir/28_4D_mbnl_S20_L008_R2_001.adaptertrimmed.fastq.gz \
$in_dir/28_4D_mbnl_S20_L008_R1_001.fastq.gz $in_dir/28_4D_mbnl_S20_L008_R2_001.fastq.gz

exit
```

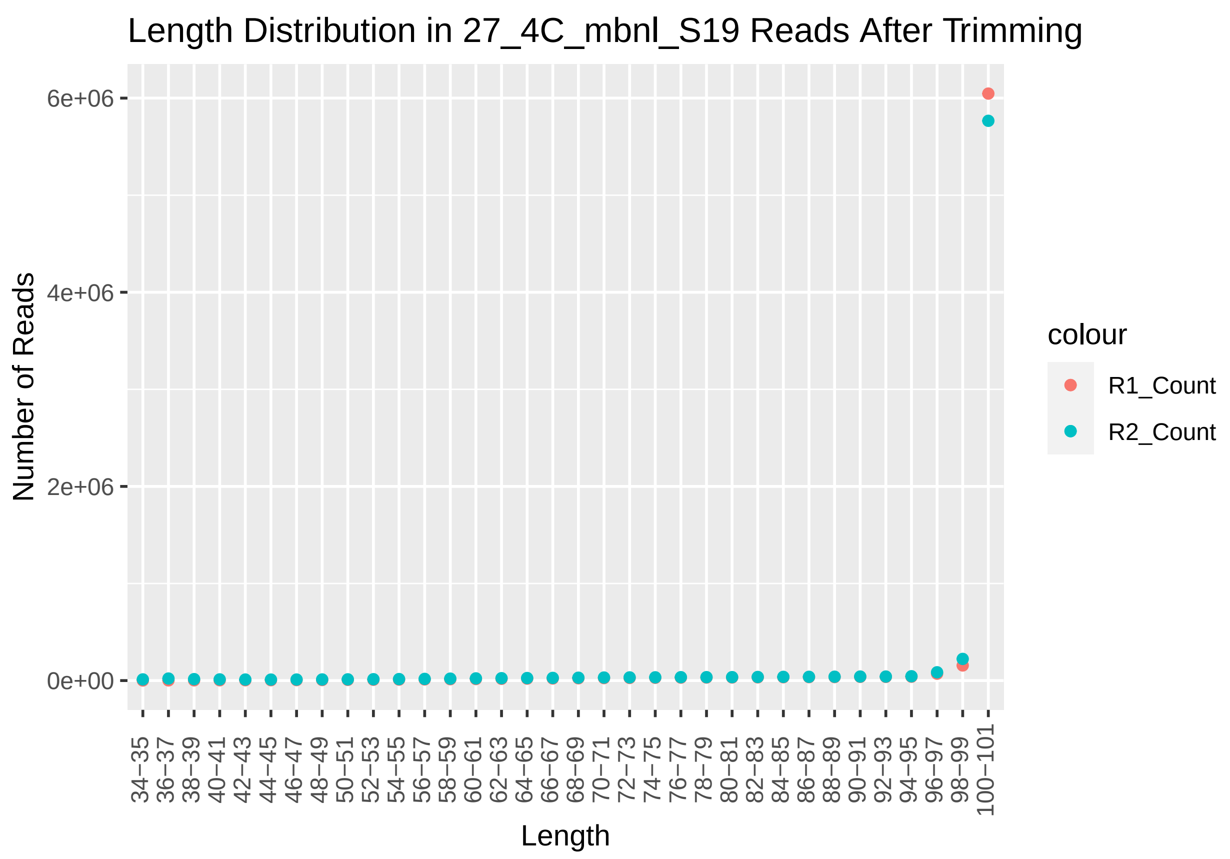
Cutadapt reports that for library 27_4C_mbnl_S19, 10.4% of read 1 reads were adapter-trimmed and 11.1% of read 2 reads were adapter-trimmed. For library 28_4D_mbnl_S20, 6.0% of read 1 reads were adapter-trimmed and 6.8% of read 2 reads were adapter-trimmed. This suggests that perhaps library 28_4D_mbnl_S20 had a fragment distribution shifted toward longer reads before sequencing.

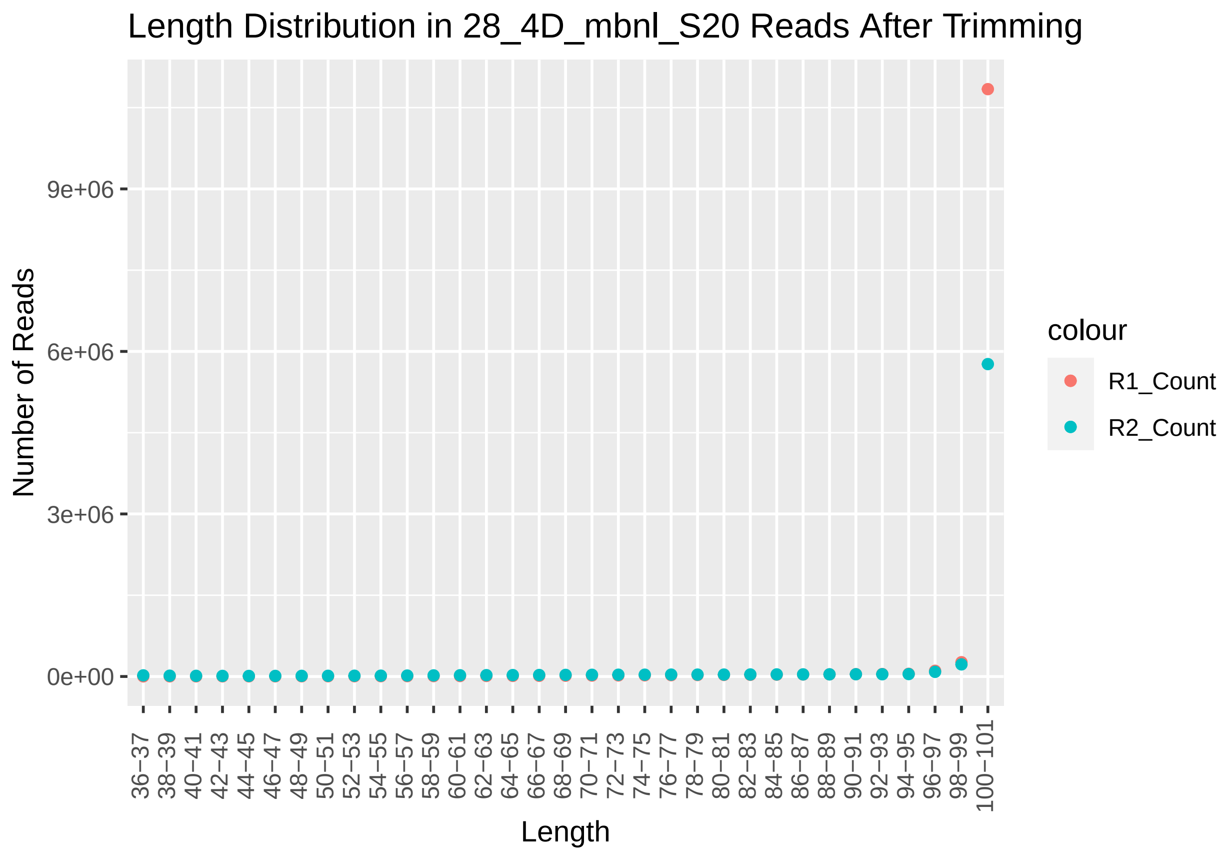
Trimmomatic version 0.39 was used to quality-trim the reads and was run with the following script on Talapas:

```
#!/bin/bash

#SBATCH --account=bgmp
#SBATCH --partition=bgmp
#SBATCH --cpus-per-task=1
#SBATCH --time=1-0:00:00

conda activate QAA

in_dir=/projects/bgmp/sgrindst/bioinformatics/Bi623/QAA/cutadapt
out_dir=/projects/bgmp/sgrindst/bioinformatics/Bi623/QAA/trimmed

/usr/bin/time -v trimmomatic PE -phred33 \
$in_dir/27_4C_mbnl_S19_L008_R1_001.adaptertrimmed.fastq.gz \
$in_dir/27_4C_mbnl_S19_L008_R2_001.adaptertrimmed.fastq.gz \
$out_dir/27_4C_mbnl_S19_L008_R1_001.trimmed.paired.fastq.gz \
$out_dir/27_4C_mbnl_S19_L008_R1_001.trimmed.unpaired.fastq.gz \
$out_dir/27_4C_mbnl_S19_L008_R2_001.trimmed.paired.fastq.gz \
$out_dir/27_4C_mbnl_S19_L008_R2_001.trimmed.unpaired.fastq.gz \
LEADING:3 TRAILING:3 SLIDINGWINDOW:5:15 MINLEN:35

/usr/bin/time -v trimmomatic PE -phred33 \
$in_dir/28_4D_mbnl_S20_L008_R1_001.adaptertrimmed.fastq.gz \
$in_dir/28_4D_mbnl_S20_L008_R2_001.adaptertrimmed.fastq.gz \
$out_dir/28_4D_mbnl_S20_L008_R1_001.trimmed.paired.fastq.gz \
$out_dir/28_4D_mbnl_S20_L008_R1_001.trimmed.unpaired.fastq.gz \
$out_dir/28_4D_mbnl_S20_L008_R2_001.trimmed.paired.fastq.gz \
$out_dir/28_4D_mbnl_S20_L008_R2_001.trimmed.unpaired.fastq.gz \
LEADING:3 TRAILING:3 SLIDINGWINDOW:5:15 MINLEN:35

exit
```

Trimmed read length distribution plots were generated by running FASTQC on the trimmed FASTQs and plotting the data in R. Only paired reads were used for this and all subsequent steps, meaning that the unpaired files output by Trimmomatic were not used.
Script submitted to Talapas to run FASTQC:

```
#!/bin/bash

#SBATCH --account=bgmp
#SBATCH --partition=bgmp
#SBATCH --cpus-per-task=1
#SBATCH --time=1-0:00:00

module load fastqc/0.11.5

input_dir=/projects/bgmp/sgrindst/bioinformatics/Bi623/QAA/trimmed
output_dir=/projects/bgmp/sgrindst/bioinformatics/Bi623/QAA/trimmed_fastqc_output/

/usr/bin/time -v fastqc \
$input_dir/27_4C_mbnl_S19_L008_R1_001.trimmed.paired.fastq.gz \
$input_dir/27_4C_mbnl_S19_L008_R2_001.trimmed.paired.fastq.gz \
$input_dir/28_4D_mbnl_S20_L008_R1_001.trimmed.paired.fastq.gz \
$input_dir/28_4D_mbnl_S20_L008_R2_001.trimmed.paired.fastq.gz \
-o $output_dir

exit
```


Length Distribution in 27_4C_mbnl_S19 Reads After Trimming
Number of Reads
6e+06
4e+06
2e+06
0e+00
Length
34−35
36−37
38−39
40−41
42−43
44−45
46−47
48−49
50−51
52−53
54−55
56−57
58−59
60−61
62−63
64−65
66−67
68−69
70−71
72−73
74−75
76−77
78−79
80−81
82−83
84−85
86−87
88−89
90−91
92−93
94−95
96−97
98−99
100−101
colour
R1_Count
R2_Count



From these plots, we can see that in both libraries read 2 is generally trimmed more often than read 1. We expect the adapter trimming rates to be about the same, since it is dependent on insert length and we expect read 1 and read 2 of the same read pair to have the same insert length. This is shown in the adapter-trimming rates reported by cutadapt, shown above. Therefore, the discrepancy between read 1 and read 2 trimming rates is due to a difference in quality trimming rates. This makes sense because read 2 tends to be of lower quality than read 1, due to the extra time sitting on the flow cell and enduring harsh washes during read 1. This is also reflected in the per-base quality scores, which we already saw to be lower in read 2. The discrepancy between read 1 and read 2 trimming rates is especially pronounced in the second library, 28_4D_mbnl_S20.

Part 3: Alignment and strand-specificity

A *Mus musculus* reference genome was generated from a FASTA file and a GTF file, both downloaded from Ensembl release 104. Both reference genome generation and alignment of the RNA-seq data to the reference database were performed using STAR version 2.7.9a. STAR was run by submitting scripts to Talapas' queuing system.
Genome generation script:

```
#!/bin/bash

#SBATCH --account=bgmp
#SBATCH --partition=bgmp
#SBATCH --cpus-per-task=8
#SBATCH --nodes=1
#SBATCH --time=1-0:00:00

conda activate QAA
```

```
/usr/bin/time -v STAR \
--runThreadN 8 \
--runMode genomeGenerate \
--genomeDir Mus_musculus.GRCm39.dna.ens104.STAR_2.7.9a \
--genomeFastaFiles /projects/bgmp/sgrindst/bioinformatics/Bi623/QAA/mus/Mus_musculus.GRCm39.dna.primary_
--sjdbGTFfile /projects/bgmp/sgrindst/bioinformatics/Bi623/QAA/mus/Mus_musculus.GRCm39.104.gtf

exit
```

Alignment script:

```
#!/bin/bash

#SBATCH --account=bgmp
#SBATCH --partition=bgmp
#SBATCH --cpus-per-task=8
#SBATCH --nodes=1
#SBATCH --time=1-0:00:00

conda activate QAA

in_dir=/projects/bgmp/sgrindst/bioinformatics/Bi623/QAA/trimmed

/usr/bin/time -v STAR --runThreadN 8 --runMode alignReads \
--outFilterMultimapNmax 3 \
--outSAMunmapped Within KeepPairs \
--alignIntronMax 1000000 --alignMatesGapMax 1000000 \
--readFilesCommand zcat \
--readFilesIn $in_dir/27_4C_mbnl_S19_L008_R1_001.trimmed.paired.fastq.gz $in_dir/27_4C_mbnl_S19_L008_R2_
--genomeDir Mus_musculus.GRCm39.dna.ens104.STAR_2.7.9a \
--outFileNamePrefix 27_4C_mbnl_S19_L008_

/usr/bin/time -v STAR --runThreadN 8 --runMode alignReads \
--outFilterMultimapNmax 3 \
--outSAMunmapped Within KeepPairs \
--alignIntronMax 1000000 --alignMatesGapMax 1000000 \
--readFilesCommand zcat \
--readFilesIn $in_dir/28_4D_mbnl_S20_L008_R1_001.trimmed.paired.fastq.gz $in_dir/28_4D_mbnl_S20_L008_R2_
--genomeDir Mus_musculus.GRCm39.dna.ens104.STAR_2.7.9a \
--outFileNamePrefix 28_4D_mbnl_S20_L008_

exit
```

From STAR's output SAM files, counts of mapped and unmapped reads were obtained by running the following Python script:

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

import argparse
import re

def get_args():
    parser = argparse.ArgumentParser(description='Takes SAM as input and returns count of reads mapping
    parser.add_argument('-f', '--file', help='input SAM file')
    return parser.parse_args()

args = get_args()

mapped_sum = 0 #counts number of unique aligned reads
unmapped_sum = 0 #counts number of unique unaligned reads

with open(args.file, 'r') as fh:
    for line in fh:
        line=line.strip()
        if line[0] == '@': #ignore header section
            continue
        flag = int(re.sub(r'[^\t]+\t([^\t]+).*',r'\1',line)) #extract bitwise flag
        if((flag & 4) != 4 and (flag & 256 != 256)):
            mapped_sum += 1
        if((flag & 4) == 4 and (flag & 256 != 256)):
            unmapped_sum += 1

print('Number of mapped reads: ', mapped_sum)
print('Number of unmapped reads: ', unmapped_sum)
```

The above Python script was run on Talapas with this wrapper script:

```
#!/bin/bash

#SBATCH --account=bgmp
#SBATCH --partition=bgmp
#SBATCH --cpus-per-task=1
#SBATCH --time=1-0:00:00

#determine the number of reads mapping and not mapping to reference

/usr/bin/time -v /projects/bgmp/sgrindst/bioinformatics/Bi621/PS/ps8-sally-grindstaff/sam_mapping_count
-f aligned/27_4C_mbnl_S19_L008_Aligned.out.sam > aligned/27_4C_mbnl_S19_L008_mapped_counts.txt

/usr/bin/time -v /projects/bgmp/sgrindst/bioinformatics/Bi621/PS/ps8-sally-grindstaff/sam_mapping_count
-f aligned/28_4D_mbnl_S20_L008_Aligned.out.sam > aligned/28_4D_mbnl_S20_L008_mapped_counts.txt
```

For library 27_4C_mbnl_S19,
Number of mapped reads: 13320042
Number of unmapped reads: 433870

For library 28_4D_mbnl_S20,
Number of mapped reads: 22657659
Number of unmapped reads: 793147

Before running HTSeq-Count, SAMTools version 1.13 was used to convert SAMs to BAMs, sort by name, and convert back to SAMs (as recommended by HTSeq-Count) with the following script:

```
#!/bin/bash

#SBATCH --account=bgmp
```

```
#SBATCH --partition=bgmp
#SBATCH --cpus-per-task=8
#SBATCH --nodes=1
#SBATCH --time=1-0:00:00

#sort sam files by name in order to run HTSeq-count later

conda activate QAA

/usr/bin/time -v samtools view -u aligned/27_4C_mbnl_S19_L008_Aligned.out.sam | \
samtools sort -n | \
samtools view -h -o aligned/27_4C_mbnl_S19_L008_Aligned.sorted.sam

/usr/bin/time -v samtools view -u aligned/28_4D_mbnl_S20_L008_Aligned.out.sam | \
samtools sort -n | \
samtools view -h -o aligned/28_4D_mbnl_S20_L008_Aligned.sorted.sam

exit
```

Then HTSeq-Count version 0.13.5 was run twice per library (once with the stranded parameter set to no and once with the stranded parameter set to yes) with this script:

```
#!/bin/bash

#SBATCH --account=bgmp
#SBATCH --partition=bgmp
#SBATCH --cpus-per-task=8
#SBATCH --nodes=1
#SBATCH --time=1-0:00:00

conda activate QAA

/usr/bin/time -v htseq-count -s yes \
aligned/27_4C_mbnl_S19_L008_Aligned.sorted.sam mus/Mus_musculus.GRCm39.104.gtf \
> aligned/27_4C_mbnl_S19_L008.stranded.genecount

/usr/bin/time -v htseq-count -s no \
aligned/27_4C_mbnl_S19_L008_Aligned.sorted.sam mus/Mus_musculus.GRCm39.104.gtf \
> aligned/27_4C_mbnl_S19_L008.unstranded.genecount

/usr/bin/time -v htseq-count -s yes \
aligned/28_4D_mbnl_S20_L008_Aligned.sorted.sam mus/Mus_musculus.GRCm39.104.gtf \
> aligned/28_4D_mbnl_S20_L008.stranded.genecount

/usr/bin/time -v htseq-count -s no \
aligned/28_4D_mbnl_S20_L008_Aligned.sorted.sam mus/Mus_musculus.GRCm39.104.gtf \
> aligned/28_4D_mbnl_S20_L008.unstranded.genecount

exit
```

HTSeq-Count results:

```
## # A tibble: 55,420 x 2
##    ENSMUSG00000000001   ‘4‘
##    <chr>              <dbl>
##  1 ENSMUSG00000000003     0
##  2 ENSMUSG00000000028     6
##  3 ENSMUSG00000000031     0
##  4 ENSMUSG00000000037     0
##  5 ENSMUSG00000000049    16
##  6 ENSMUSG00000000056     3
##  7 ENSMUSG00000000058     2
##  8 ENSMUSG00000000078     3
##  9 ENSMUSG00000000085     7
## 10 ENSMUSG00000000088     4
## # ... with 55,410 more rows

## # A tibble: 55,420 x 2
##    ENSMUSG00000000001 ‘1372‘
##    <chr>               <dbl>
##  1 ENSMUSG00000000003      0
##  2 ENSMUSG00000000028    652
##  3 ENSMUSG00000000031      0
##  4 ENSMUSG00000000037      0
##  5 ENSMUSG00000000049      0
##  6 ENSMUSG00000000056    164
##  7 ENSMUSG00000000058    547
##  8 ENSMUSG00000000078   1065
##  9 ENSMUSG00000000085    341
## 10 ENSMUSG00000000088    314
## # ... with 55,410 more rows

## # A tibble: 55,420 x 2
##    ENSMUSG00000000001   ‘5‘
##    <chr>              <dbl>
##  1 ENSMUSG00000000003     0
##  2 ENSMUSG00000000028     4
##  3 ENSMUSG00000000031     0
##  4 ENSMUSG00000000037     0
##  5 ENSMUSG00000000049    21
##  6 ENSMUSG00000000056     4
##  7 ENSMUSG00000000058     1
##  8 ENSMUSG00000000078     2
##  9 ENSMUSG00000000085     4
## 10 ENSMUSG00000000088     4
## # ... with 55,410 more rows

## # A tibble: 55,420 x 2
##    ENSMUSG00000000001 ‘2901‘
##    <chr>               <dbl>
##  1 ENSMUSG00000000003      0
##  2 ENSMUSG00000000028    988
##  3 ENSMUSG00000000031      0
##  4 ENSMUSG00000000037      0
##  5 ENSMUSG00000000049      0
##  6 ENSMUSG00000000056    305
##  7 ENSMUSG00000000058    698
##  8 ENSMUSG00000000078   1387
##  9 ENSMUSG00000000085    408
## 10 ENSMUSG00000000088    400
## # ... with 55,410 more rows
```

For each HTSeq-Count run, the number of reads mapping to features and the total number of reads were analyzed with the following bash script:

```
#!/bin/bash

#SBATCH --account=bgmp
#SBATCH --partition=bgmp
#SBATCH --cpus-per-task=1
#SBATCH --time=1-0:00:00

#count the number of reads mapping to a feature and the total reads (using htseq-count output) in order
#determine whether the data is stranded or unstranded

#commands are taken from my ICA4 GitHub markdown file from Bi621

echo '27_4C_mbnl_S19_L008.stranded.genecount:' > aligned/htseq_count_stats.txt
echo 'reads mapping to feature:' >> aligned/htseq_count_stats.txt
grep -v '^_' aligned/27_4C_mbnl_S19_L008.stranded.genecount | awk '{sum+=$2}END{print sum}' >> aligned/
echo 'total reads:' >> aligned/htseq_count_stats.txt
awk '{sum+=$2}END{print sum}' aligned/27_4C_mbnl_S19_L008.stranded.genecount >> aligned/htseq_count_sta
echo '' >> aligned/htseq_count_stats.txt

echo '27_4C_mbnl_S19_L008.unstranded.genecount:' >> aligned/htseq_count_stats.txt
echo 'reads mapping to feature:' >> aligned/htseq_count_stats.txt
grep -v '^_' aligned/27_4C_mbnl_S19_L008.unstranded.genecount | awk '{sum+=$2}END{print sum}' >> aligne
echo 'total reads:' >> aligned/htseq_count_stats.txt
awk '{sum+=$2}END{print sum}' aligned/27_4C_mbnl_S19_L008.unstranded.genecount >> aligned/htseq_count_s
echo '' >> aligned/htseq_count_stats.txt

echo '28_4D_mbnl_S20_L008.stranded.genecount:' >> aligned/htseq_count_stats.txt
echo 'reads mapping to feature:' >> aligned/htseq_count_stats.txt
grep -v '^_' aligned/28_4D_mbnl_S20_L008.stranded.genecount | awk '{sum+=$2}END{print sum}' >> aligned/
echo 'total reads:' >> aligned/htseq_count_stats.txt
awk '{sum+=$2}END{print sum}' aligned/28_4D_mbnl_S20_L008.stranded.genecount >> aligned/htseq_count_sta
echo '' >> aligned/htseq_count_stats.txt

echo '28_4D_mbnl_S20_L008.unstranded.genecount:' >> aligned/htseq_count_stats.txt
echo 'reads mapping to feature:' >> aligned/htseq_count_stats.txt
grep -v '^_' aligned/28_4D_mbnl_S20_L008.unstranded.genecount | awk '{sum+=$2}END{print sum}' >> aligne
echo 'total reads:' >> aligned/htseq_count_stats.txt
awk '{sum+=$2}END{print sum}' aligned/28_4D_mbnl_S20_L008.unstranded.genecount >> aligned/htseq_count_s

exit
```

27_4C_mbnl_S19_L008.stranded.genecount:
reads mapping to feature:
271747

total reads:
6876956

27_4C_mbnl_S19_L008.unstranded.genecount:
reads mapping to feature:
5545724
total reads:
6876956

28_4D_mbnl_S20_L008.stranded.genecount:
reads mapping to feature:
424716
total reads:
11725403

28_4D_mbnl_S20_L008.unstranded.genecount:
reads mapping to feature:
9414809
total reads:
11725403

Converting these numbers to percentages, we get: 27_4C_mbnl_S19_L008, stranded – 3.9515594% of reads mapping to features
27_4C_mbnl_S19_L008, unstranded – 80.6421329% of reads mapping to features
28_4D_mbnl_S20_L008, stranded – 3.6221868% of reads mapping to features
28_4D_mbnl_S20_L008, unstranded – 80.2941187% of reads mapping to features

From these data, I propose that the libraries were prepared using a stranded method. I believe this to be true because if the libraries were unstranded, then I would expect about half as many reads to map to features with the stranded parameter set to 'yes' than with the stranded parameter set to 'no'. This is because in an unstranded library reads 1 and 2 can both correspond to either the template or coding strand of the genome, so I would not expect a bias. However, in this case I see an order of magnitude difference (3.9515594% vs. 80.6421329% for the first library and 3.6221868% vs. 80.2941187% for the second library) when I set the stranded parameter to yes vs. no in HTSeq-Count. This indicates that most of the mapping features are coming from one strand over the other, which leads me to believe that the libraries were prepared in a strand-specific manner.