QAA

Pranav Muthuraman

9/7/2022

Part 1

Per-base quality score plots were generated by running FASTQC version 11.5 on Talapas, using the following shell script:

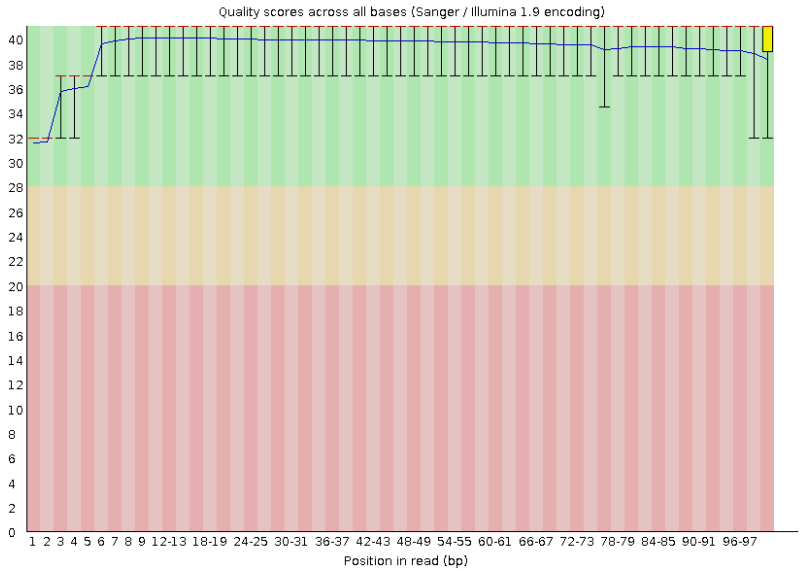
```
cat fastqc_run.sh
```

```
#!/bin/bash

#SBATCH --account=bgmp
#SBATCH --partition=bgmp
#SBATCH --cpus-per-task=1
#SBATCH --time=1-0:00:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --nodelist=n226         ### Run on node 226

module load fastqc/0.11.5

input_dir=/projects/bgmp/shared/2017_sequencing/demultiplexed
output_dir=/projects/bgmp/pmuthura/bioinfo/Bi623/QAA/fastqc_output

/usr/bin/time -v fastqc $input_dir/8_2F_fox_S7_L008_R1_001.fastq.gz $input_dir/8_2F_fox_S7_L008_R2_001.fastq.

exit
```

8_2F_fox_S7_L008_R1_001.fastq.gz

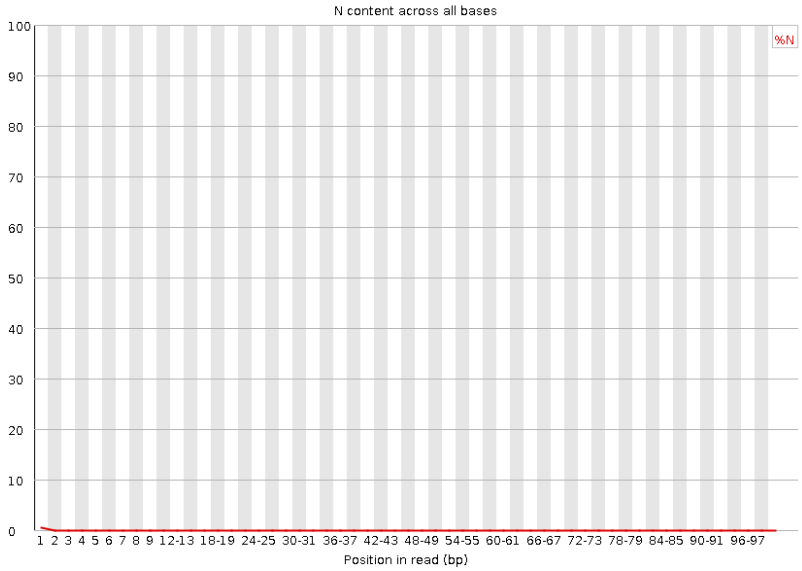
Quality Score Distribution for Fox Read 1

```
knitr::include_graphics("/Users/pranav_muthuraman/Desktop/QAA/fox_read1.png")
```



Per base N Distribution for Fox Read 1

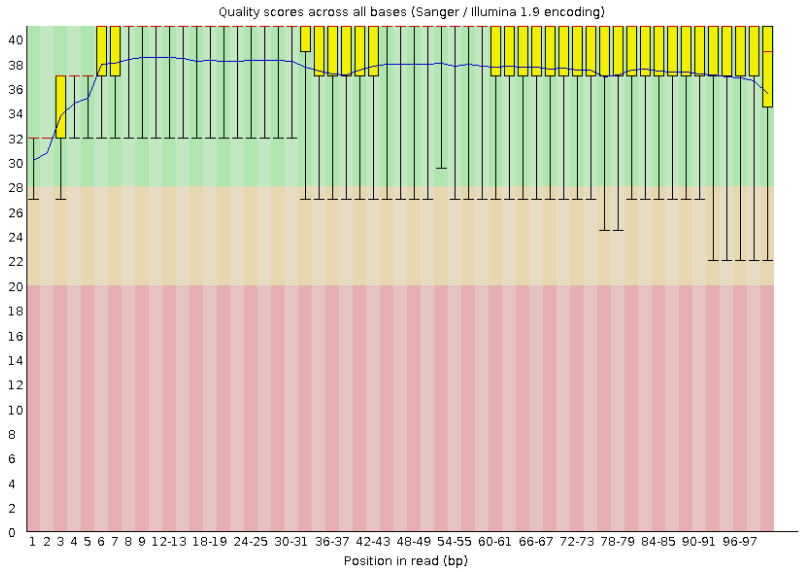
```
knitr::include_graphics("/Users/pranav_muthuraman/Desktop/QAA/fox_read1_N.png")
```



8_2F_fox_S7_L008_R2_001.fastq.gz

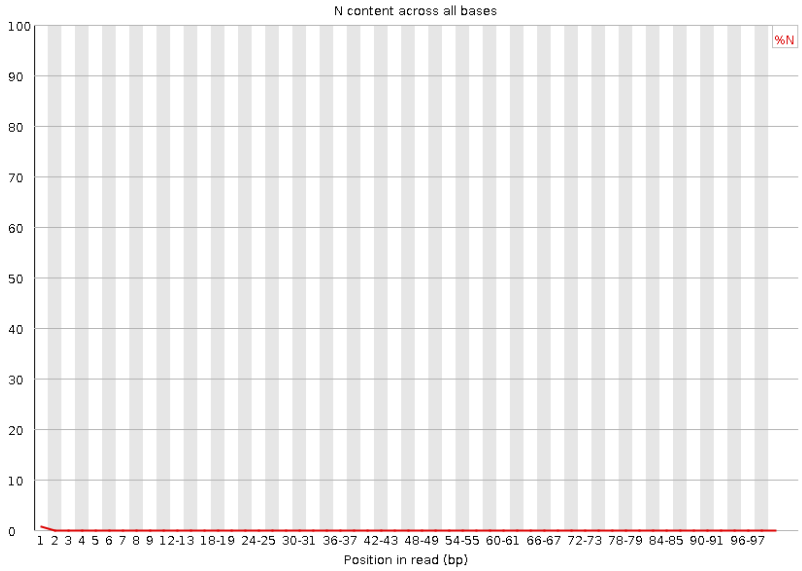
Quality Score Distribution for Fox Read 2

```
knitr::include_graphics("/Users/pranav_muthuraman/Desktop/QAA/fox_read2.png")
```



Per base N Distribution for Fox Read 2

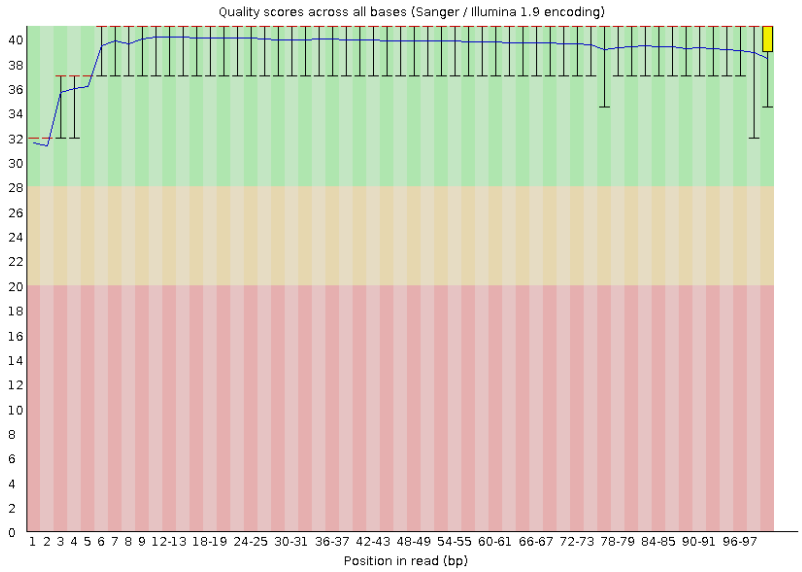
```
knitr::include_graphics("/Users/pranav_muthuraman/Desktop/QAA/fox_read2_N.png")
```



14_3B_control_S10_L008_R1_001.fastq.gz

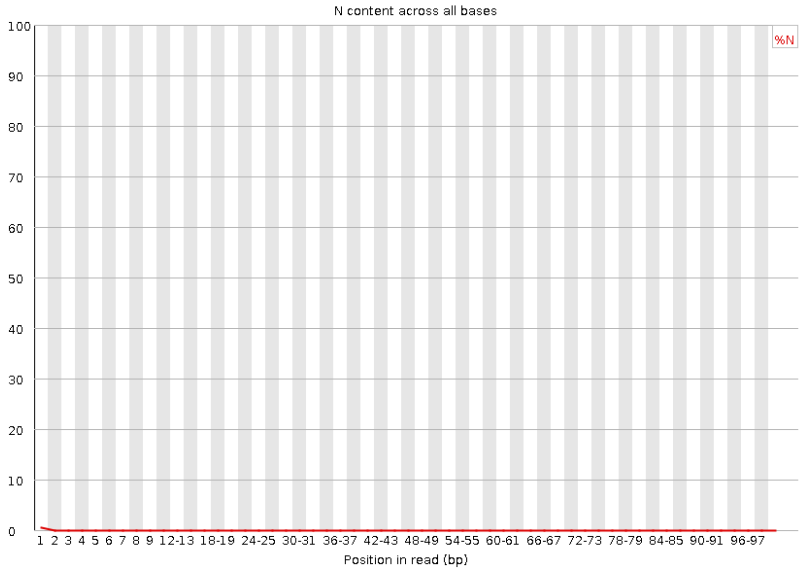
Quality Score Distribution for Control Read 1

```
knitr::include_graphics("/Users/pranav_muthuraman/Desktop/QAA/control_read1.png")
```



Per base N Distribution for Control Read 1

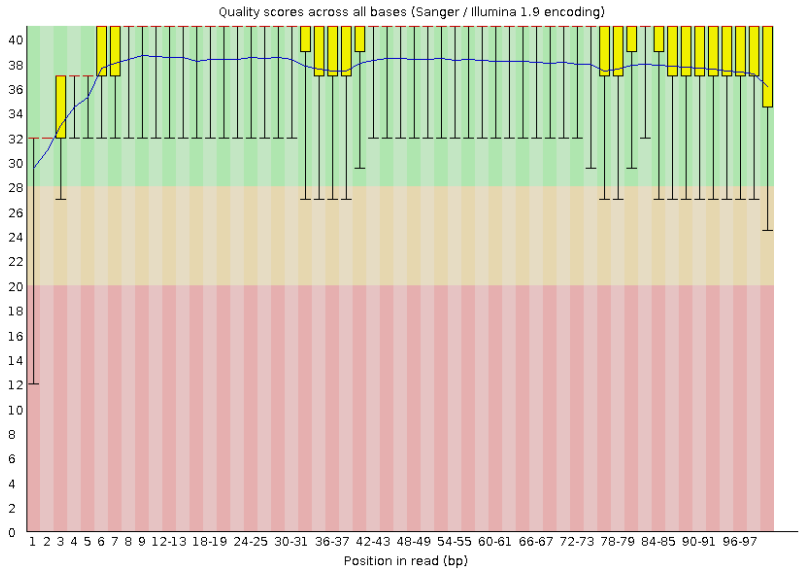
```
knitr::include_graphics("/Users/pranav_muthuraman/Desktop/QAA/control_read1_N.png")
```



14_3B_control_S10_L008_R2_001.fastq.gz

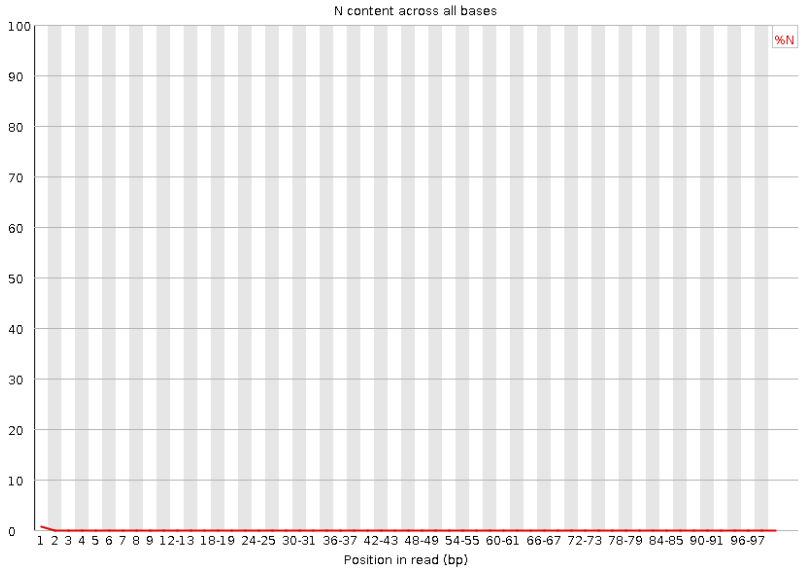
Quality Score Distribution for Control Read 2

```
knitr::include_graphics("/Users/pranav_muthuraman/Desktop/QAA/control_read2.png")
```



Per base N Distribution for Control Read 2

```
knitr::include_graphics("/Users/pranav_muthuraman/Desktop/QAA/control_read2_N.png")
```



QUESTION 1

> For the per-base quality scores, the read1 files for both fox and control start a bit lower, then remain very high for the entire read. For the read2 files, fox starts a bit lower and has consistently lower quality throughout. The control starts even lower than the fox file, but improves to the same level after the 5th base pair. All 4 quality scores received a green passing mark. In all files, the N distributions are very low.

I also produced these qualtity score plots using python; the script, the bioinfo module, and bash scripts used to run on talapas are shown here:

Bioinfo module:

```
cat Bioinfo.py
```

```
# Author: <PRANAV MUTHURAMAN> <pmuthura@uoregon.edu>
```

```
# Check out some Python module resources:
#   - https://docs.python.org/3/tutorial/modules.html
#   - https://python101.pythonlibrary.org/chapter36_creating_modules_and_packages.html
#   - and many more: https://www.google.com/search?q=how+to+write+a+python+module


'''This module is a collection of useful bioinformatics functions
written during the Bioinformatics and Genomics Program coursework.
__version__ = "0.5"         # Read way more about versioning here:
                            # https://en.wikipedia.org/wiki/Software_versioning
                            '''

DNA_bases = "ACTGN"
RNA_bases = "ACUGN"

def convert_phred(letter: str) -> int:
    '''
    This function takes a ASCII value representing the Quality score and converts it to the decimal value - 3
    Input: A
    Expected output: 32
    parameter: string score
    return: int decimal_score
    '''
    dec = ord(letter)
    decimal_score = dec - 33

    return decimal_score


def qual_score(phred_letter):
    '''
    This function take a quality string and returns the average quality score of that line
    Input: AAAIII
    Expected output: 36
    parameter: string qual_line
    return: float avg_qual
    '''

    qual_list = list()

    for i in phred_letter:
        qual_list.append(convert_phred(i))

    average = 0

    for i in qual_list:
        average += int(i)

    return (average/len(qual_list))


def contain_N(index: str) -> int:
    '''
    This function takes a string of 2 indexes seperated with a '-' and returns the number of Ns it contains
    Input: AACTTGCC-ANCTNGCC
    Expected output: 2
    parameter: string index
    return: int num_ns
```

```
    '''
    num_ns = 0
    for char in index:
        if char.upper() == "N":
            num_ns += 1

    return num_ns

def if_match(index: str) -> bool:
    '''
    This function takes a string of 2 indexes seperated with a '-' and returns True if they match and False i
    Input: AACTTGCC-AACTTGCC
    Expected output: True
    parameter: string index
    return: bool match_flag
    '''
    index1 = index[0:7]
    index2 = index[9:16]
    if index1 == index2:
        return True
    else:
        return False

def reverse_comp(index: str) -> str:
    '''
    This function takes a string of 1 index and returns a string of the reverse complement
    Input: AACTTGCC
    Expected output: GGCAAGTT
    parameter: string index
    return: string rev_comp
    '''
    reverse = index[::-1]
    complements = {'A': 'T', 'C': 'G', 'T': 'A', 'G': 'C', 'N': 'N'}
    rc = ''
    for base in reverse:
        rc = str(rc + complements[base])

    return rc


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)


def gc_content(DNA: str) -> int:
    '''Takes DNA (or RNA) sequence and returns GC content of the sequence in decimal format as a fraction of
    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)


def oneline_fasta (self):
    '''Reads through the fasta file and return the header and sequences for each record.'''
    header = ''
    sequence = ''

    with open(self.fname) as fh:
        header = ''
        sequence = ''

        # Read the First Line and Append header
        line = fh.readline()
        header = line[1:].rstrip()

        # Read the next line continuous and add up the squences until the next header is reach
        # If so return the header and sequence
        for line in fh:
            if line.startswith ('>'):
                yield header,sequence
                header = line[1:].rstrip()      # The next header
                sequence = ''                   # Clear Sequence
            else :
                sequence += ''.join(line.rstrip().split()).upper()

    yield header,sequence


if __name__ == "__main__":
    assert (convert_phred('A')) == 32, 'Wrong phred score'
    print('Phred Score Conversion Correct')

    assert (qual_score('AAAIII')) == 36, 'Wrong average quality score'
    print('Average Quality Score Correct')

    assert validate_base_seq("ACTGATA") == True, "Validate base seq does not work on DNA"
    assert validate_base_seq("AGUAUCA", True) == True, "Validate base seq does not work on RNA"
    assert validate_base_seq("Random String") == False, "Validate base seq fails to recognize nonDNA"
    print("Validate Base Seq Correct")

    assert gc_content('GCAGCGTTAA') == 0.5, 'gc_content does not find correct GC content'
    print('GC Content Correct')
```

Python script:

```
cat part1.py
```

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

import argparse
import gzip
import numpy as np
import bioinfo as df
```

```
def get_args():
    parse = argparse.ArgumentParser(description="A program to find the distribution of bases")
    parse.add_argument("-f", "--filename", help="the name of the file", required=True)
    parse.add_argument("-l", "--length", help="the length of the reads or indexes", type=int, required=True)
    parse.add_argument("-o", "--output", help="the name of the output file", required=True)
    return parse.parse_args()

args = get_args()

#use a numpy array for faster run time
score_list = np.zeros(args.length, dtype=int)
# open the file with gzip and read each quality line and keep a running total of each base
with gzip.open(args.filename, 'rt') as fh:
    counter = 0
    while True:
        header = fh.readline()
        if header == '':
            break
        seq = fh.readline()
        plus = fh.readline()
        qual = fh.readline()
        #add the quality score for each base to the numpy array
        for i,score in enumerate(qual.strip()):
            score_list[i] += df.convert_phred(score)
        counter += 1

#divide the whole list by the number of records to get the mean quality score at each base
score_list = score_list/counter

#create a list of base pair sites
bp = []
for i in range(args.length):
    bp.append(i)

#create a bar plot for the average quality score at each site
import matplotlib.pyplot as plt
plt.figure(figsize=(15,5))
plt.bar(bp, score_list, edgecolor='b')

plt.title("Distribution of Average Quality Scores: %s" % args.output)
plt.xlabel("Base Pair")
plt.ylabel("Average Quality Score")

plt.savefig('%s.png' % args.output)
```

Scripts run on Talapas:

```
cat 8_2F_fox_R1_qscore_dist_run.sh
```

```
#!/bin/bash

#SBATCH --account=bgmp
#SBATCH --partition=bgmp
#SBATCH --cpus-per-task=1
#SBATCH --time=1-0:00:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --nodelist=n226         ### Run on node 226
```

```
conda activate bgmp_py310

/usr/bin/time -v ./part1.py -f /projects/bgmp/shared/2017_sequencing/demultiplexed/8_2F_fox_S7_L008_R1_001.fa

exit
```

```
cat 8_2F_fox_R2_qscore_dist_run.sh
```

```
#!/bin/bash

#SBATCH --account=bgmp
#SBATCH --partition=bgmp
#SBATCH --cpus-per-task=1
#SBATCH --time=1-0:00:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --nodelist=n226         ### Run on node 226

conda activate bgmp_py310

/usr/bin/time -v ./part1.py -f /projects/bgmp/shared/2017_sequencing/demultiplexed/8_2F_fox_S7_L008_R2_001.fa

exit
```

```
cat 14_3B_control_R1_qscore_dist_run.sh
```

```
#!/bin/bash

#SBATCH --account=bgmp
#SBATCH --partition=bgmp
#SBATCH --cpus-per-task=1
#SBATCH --time=1-0:00:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --nodelist=n228         ### Run on node 228

conda activate bgmp_py310

/usr/bin/time -v ./part1.py -f /projects/bgmp/shared/2017_sequencing/demultiplexed/14_3B_control_S10_L008_R1_

exit
```

```
cat 14_3B_control_R2_qscore_dist_run.sh
```

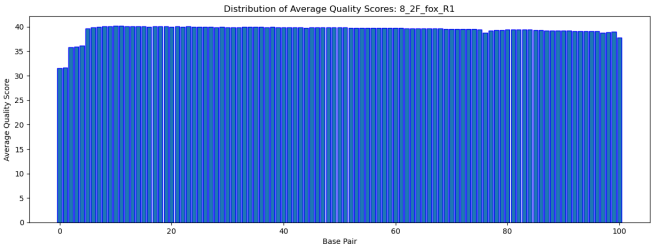
```
#!/bin/bash

#SBATCH --account=bgmp
#SBATCH --partition=bgmp
#SBATCH --cpus-per-task=1
#SBATCH --time=1-0:00:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --nodelist=n229        ### Run on node 229

conda activate bgmp_py310
```

```
/usr/bin/time -v ./part1.py -f /projects/bgmp/shared/2017_sequencing/demultiplexed/14_3B_control_S10_L008_R2_

exit
```

Python Generated Plots:

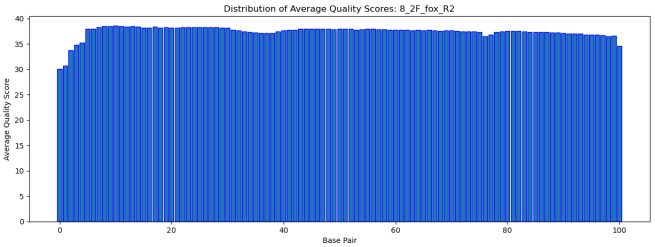
Python Gen: Per base N Distribution for Fox Read 1

```
knitr::include_graphics("/Users/pranav_muthuraman/Desktop/QAA/8_2F_fox_R1.png")
```



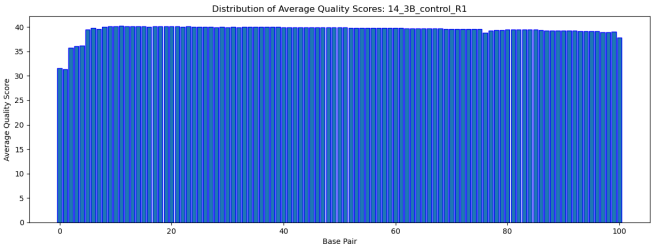
Python Gen: Per base N Distribution for Fox Read 2

```
knitr::include_graphics("/Users/pranav_muthuraman/Desktop/QAA/8_2F_fox_R2.png")
```



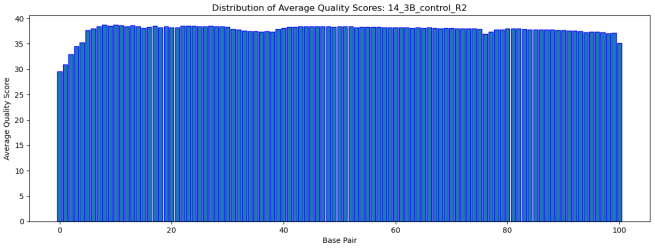
Python Gen: Per base N Distribution for Control Read 1

```
knitr::include_graphics("/Users/pranav_muthuraman/Desktop/QAA/14_3B_control_R1.png")
```



Python Gen: Per base N Distribution for Control Read 2

```
knitr::include_graphics("/Users/pranav_muthuraman/Desktop/QAA/14_3B_control_R2.png")
```



QUESTION 2

All the distribution plots created by the python script look very similar to the ones created by FASTQC. The runtime to create each of these plots were: 21 mins, 21 mins, 3 mins, and 3 mins, respectively. The runtime of the entire fastQC was only 8 mins. FastQC may have run more quickly because it is able to read multiple files at once.

QUESTION 3

Overall, the FOX files have better data quality than the CONTROL due to the quality scores. In both sets of data, the read 1 files have significantly better quality scores than the read 2 files.

Part 2

QUESTION 4

```
conda create --name QAA python=3.9
conda activate QAA
conda install -c bioconda cutadapt
conda install -c bioconda trimmomatic

cutadapt --version
4.1

trimmomatic -version
0.39
```

QUESTION 5

cutadapt script:

```
cat cutadapt_run.sh
```

```
#!/bin/bash

#SBATCH --account=bgmp
#SBATCH --partition=bgmp
#SBATCH --cpus-per-task=1
#SBATCH --time=1-0:00:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --nodelist=n226         ### Run on node 226

conda activate QAA

in_dir=/projects/bgmp/shared/2017_sequencing/demultiplexed
out_dir=/projects/bgmp/pmuthura/bioinfo/Bi623/QAA/cutadapt

/usr/bin/time -v cutadapt -a AGATCGGAAGAGCACACGTCTGAACTCCAGTCA -A AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT \
-o $out_dir/8_2F_fox_S7_L008_R1_001.adaptertrimmed.fastq.gz \
-p $out_dir/8_2F_fox_S7_L008_R2_001.adaptertrimmed.fastq.gz \
$in_dir/8_2F_fox_S7_L008_R1_001.fastq.gz $in_dir/8_2F_fox_S7_L008_R2_001.fastq.gz

/usr/bin/time -v cutadapt -a AGATCGGAAGAGCACACGTCTGAACTCCAGTCA -A AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT \
-o $out_dir/14_3B_control_S10_L008_R1_001.adaptertrimmed.fastq.gz \
```

```
-p $out_dir/14_3B_control_S10_L008_R2_001.adaptertrimmed.fastq.gz \
$in_dir/14_3B_control_S10_L008_R1_001.fastq.gz $in_dir/14_3B_control_S10_L008_R2_001.fastq.gz

exit

Cut adapt summary:
8_2F_fox:
Total read pairs processed:         36,482,601
  Read 1 with adapter:               2,145,600 (5.9%)
  Read 2 with adapter:               2,403,490 (6.6%)
Pairs written (passing filters):    36,482,601 (100.0%)

14_3B_control:
Total read pairs processed:          4,440,378
  Read 1 with adapter:                 264,208 (6.0%)
  Read 2 with adapter:                 299,716 (6.7%)
Pairs written (passing filters):     4,440,378 (100.0%)

SANITY CHECK:

forward adapters
zcat /projects/bgmp/shared/2017_sequencing/demultiplexed/8_2F_fox_S7_L008_R1_001.fastq.gz |
grep 'AGATCGGAAGAGCACACGTCTGAACTCCAGTCA'
zcat /projects/bgmp/shared/2017_sequencing/demultiplexed/8_2F_fox_S7_L008_R2_001.fastq.gz |
grep 'AGATCGGAAGAGCACACGTCTGAACTCCAGTCA' --> nothing
zcat /projects/bgmp/shared/2017_sequencing/demultiplexed/14_3B_control_S10_L008_R1_001.fastq.gz |
grep 'AGATCGGAAGAGCACACGTCTGAACTCCAGTCA'
zcat /projects/bgmp/shared/2017_sequencing/demultiplexed/14_3B_control_S10_L008_R2_001.fastq.gz |
grep 'AGATCGGAAGAGCACACGTCTGAACTCCAGTCA' --> nothing

reverse adapters
zcat /projects/bgmp/shared/2017_sequencing/demultiplexed/8_2F_fox_S7_L008_R1_001.fastq.gz |
grep 'AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT' --> nothing
zcat /projects/bgmp/shared/2017_sequencing/demultiplexed/8_2F_fox_S7_L008_R2_001.fastq.gz |
grep 'AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT'
zcat /projects/bgmp/shared/2017_sequencing/demultiplexed/14_3B_control_S10_L008_R1_001.fastq.gz |
grep 'AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT' --> nothing
zcat /projects/bgmp/shared/2017_sequencing/demultiplexed/14_3B_control_S10_L008_R2_001.fastq.gz |
grep 'AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT'
```

Checked for forward primer in read1 and read2 files. Adapter in every read1 file, but not in read2 files. Checked for reverse primer in read1 and read2 files. Adapter in every read2 file, but not in read1 files.

QUESTION 6

trimmomatic script

```
cat trimmomatic_run.sh
```

```
#!/bin/bash

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

```
#SBATCH --nodelist=n226         ### Run on node 226

conda activate QAA

in_dir=/projects/bgmp/pmuthura/bioinfo/Bi623/QAA/cutadapt
out_dir=/projects/bgmp/pmuthura/bioinfo/Bi623/QAA/trimmed

/usr/bin/time -v trimmomatic PE -phred33 \
$in_dir/8_2F_fox_S7_L008_R1_001.adaptertrimmed.fastq.gz \
$in_dir/8_2F_fox_S7_L008_R2_001.adaptertrimmed.fastq.gz \
$out_dir/8_2F_fox_S7_L008_R1_001.trimmed.paired.fastq.gz \
$out_dir/8_2F_fox_S7_L008_R1_001.trimmed.unpaired.fastq.gz \
$out_dir/8_2F_fox_S7_L008_R2_001.trimmed.paired.fastq.gz \
$out_dir/8_2F_fox_S7_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/14_3B_control_S10_L008_R1_001.adaptertrimmed.fastq.gz \
$in_dir/14_3B_control_S10_L008_R2_001.adaptertrimmed.fastq.gz \
$out_dir/14_3B_control_S10_L008_R1_001.trimmed.paired.fastq.gz \
$out_dir/14_3B_control_S10_L008_R1_001.trimmed.unpaired.fastq.gz \
$out_dir/14_3B_control_S10_L008_R2_001.trimmed.paired.fastq.gz \
$out_dir/14_3B_control_S10_L008_R2_001.trimmed.unpaired.fastq.gz \
LEADING:3 TRAILING:3 SLIDINGWINDOW:5:15 MINLEN:35

exit
```

QUESTION 7

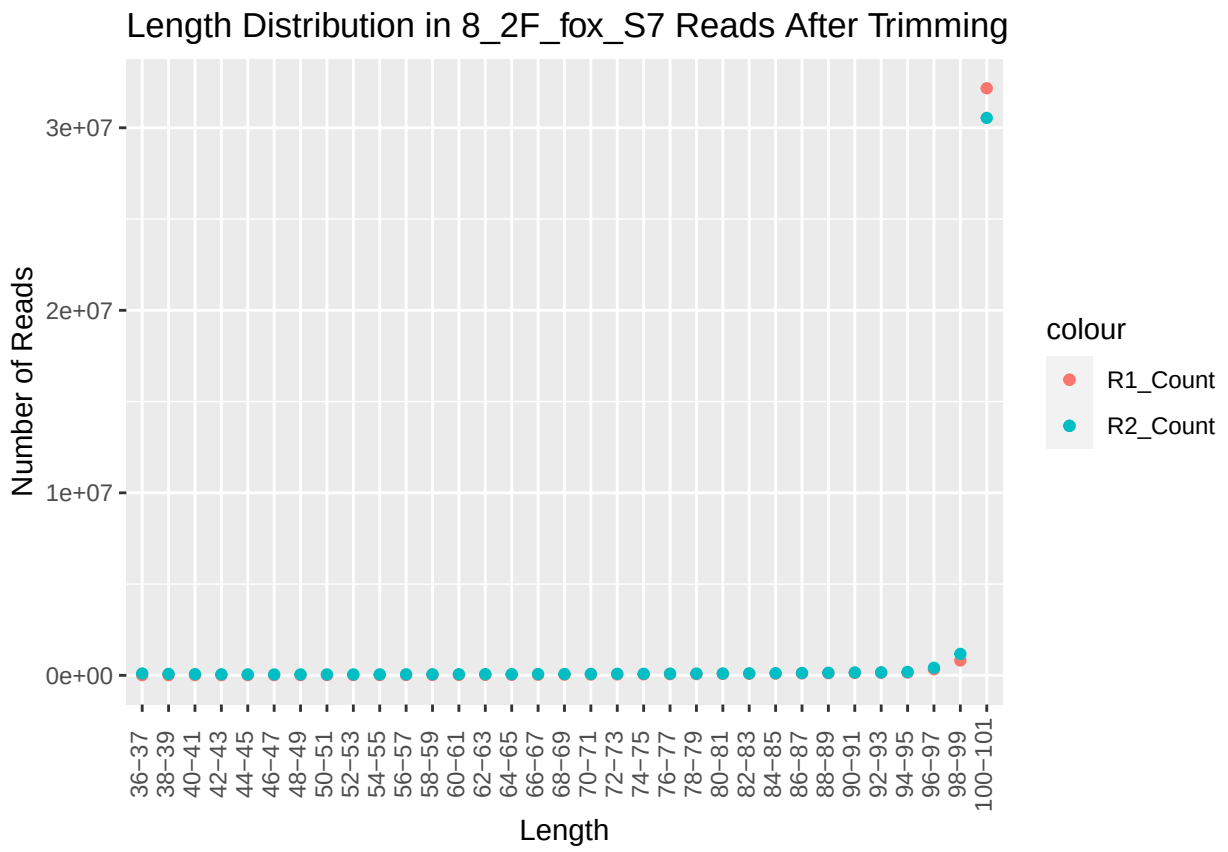
To plot read length distributions, I ran FASTQC on the trimmed reads outputted from question 6. I only used the paired reads for these plots and all subsequent downstream analysis.

trimmed FASTQC script

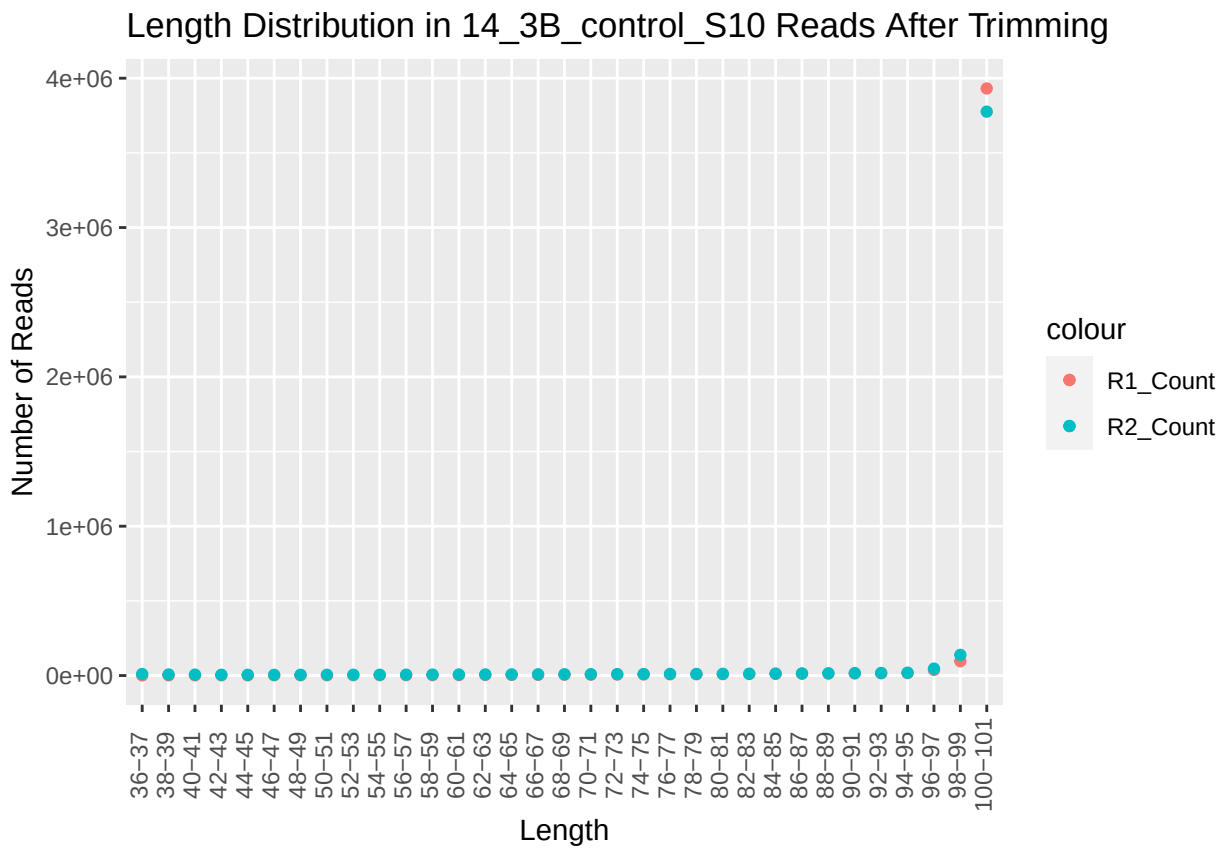
```
cat trimmed_fastqc_run.sh
```

```
#!/bin/bash

#SBATCH --account=bgmp
#SBATCH --partition=bgmp
#SBATCH --cpus-per-task=1
#SBATCH --time=1-0:00:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --nodelist=n226         ### Run on node 226

module load fastqc/0.11.5

input_dir=/projects/bgmp/pmuthura/bioinfo/Bi623/QAA/trimmed_paired
output_dir=/projects/bgmp/pmuthura/bioinfo/Bi623/QAA/trimmed_fastqc_output/

/usr/bin/time -v fastqc \
$input_dir/14_3B_control_S10_L008_R1_001.trimmed.paired.fastq.gz \
$input_dir/14_3B_control_S10_L008_R2_001.trimmed.paired.fastq.gz \
$input_dir/8_2F_fox_S7_L008_R1_001.trimmed.paired.fastq.gz \
$input_dir/8_2F_fox_S7_L008_R2_001.trimmed.paired.fastq.gz \
-o $output_dir

exit
```

```
len_levels = c("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")
ggplot(data = fox_trim_len, mapping = aes(x = factor(Length, levels = len_levels))) +
    geom_point(mapping = aes(y = R1_Count, color = "R1_Count")) + geom_point(mapping = aes(y = R2_Count,
    color = "R2_Count")) + labs(title = "Length Distribution in 8_2F_fox_S7 Reads After Trimming",
    x = "Length", y = "Number of Reads") + theme(axis.text.x = element_text(size = 9,
    angle = 90, vjust = 0.3))
```




```
ggplot(data = control_trim_len, mapping = aes(x = factor(Length, levels = len_levels))) +
    geom_point(mapping = aes(y = R1_Count, color = "R1_Count")) + geom_point(mapping = aes(y = R2_Count,
    color = "R2_Count")) + labs(title = "Length Distribution in 14_3B_control_S10 Reads After Trimming",
    x = "Length", y = "Number of Reads") + theme(axis.text.x = element_text(size = 9,
    angle = 90, vjust = 0.3))
```



We can see that in both libraries, read 2 is trimmed slightly more often than read 1. The adapter trimming rates should be about the same as they are dependent on insert length, and both reads of each library should have the same insert length. The small difference between read 1 and read 2 are probably due to the quality trimming rates, and we know that read 2 is of slighlty lower quality than read 1.

Part 3

QUESTION 8

```
conda activate QAA
conda install -c bioconda star
conda install -c conda-forge numpy
conda install -c bioconda pysam
conda install -c conda-forge matplotlib
conda install -c bioconda htseq
```

QUESTION 9

Generating alignment database script

```
cat star_makedb_run.sh
```

```
#!/bin/bash

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

```
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=8       ### Number of cpus (cores) per task
#SBATCH --nodelist=n226         ### Run on node 226

conda activate QAA

/usr/bin/time -v STAR \
--runThreadN 8 \
--runMode genomeGenerate \
--genomeDir Mus_musculus.GRCm39.dna.ens107.STAR_2.7.10a \
--genomeFastaFiles /projects/bgmp/pmuthura/bioinfo/Bi623/QAA/mus/Mus_musculus.GRCm39.dna.primary_assembly.fa
--sjdbGTFfile /projects/bgmp/pmuthura/bioinfo/Bi623/QAA/mus/Mus_musculus.GRCm39.107.gtf

exit
```

STAR alignment script

```
cat star_align_run.sh
```

```
#!/bin/bash

#SBATCH --account=bgmp
#SBATCH --partition=bgmp
#SBATCH --cpus-per-task=1
#SBATCH --time=1-0:00:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=8       ### Number of cpus (cores) per task
#SBATCH --nodelist=n226         ### Run on node 226

conda activate QAA

in_dir=/projects/bgmp/pmuthura/bioinfo/Bi623/QAA/trimmed_paired

/usr/bin/time -v STAR --runThreadN 8 --runMode alignReads \
--outFilterMultimapNmax 3 \
--outSAMunmapped Within KeepPairs \
--alignIntronMax 1000000 --alignMatesGapMax 1000000 \
--readFilesCommand zcat \
--readFilesIn $in_dir/8_2F_fox_S7_L008_R1_001.trimmed.paired.fastq.gz $in_dir/8_2F_fox_S7_L008_R2_001.trimmed
--genomeDir Mus_musculus.GRCm39.dna.ens107.STAR_2.7.10a \
--outFileNamePrefix 8_2F_fox_S7_L008_

/usr/bin/time -v STAR --runThreadN 8 --runMode alignReads \
--outFilterMultimapNmax 3 \
--outSAMunmapped Within KeepPairs \
--alignIntronMax 1000000 --alignMatesGapMax 1000000 \
--readFilesCommand zcat \
--readFilesIn $in_dir/14_3B_control_S10_L008_R1_001.trimmed.paired.fastq.gz $in_dir/14_3B_control_S10_L008_R2
--genomeDir Mus_musculus.GRCm39.dna.ens107.STAR_2.7.10a \
--outFileNamePrefix 14_3B_control_S10_L008_

exit
```

QUESTION 10

Python script from PS8 to report mapped/unmapped reads from SAM files

```
cat PS8.py
```

```
#!/usr/bin/env python
'''
PS8.py
'''

import sys
'''
Read through the sam file and create of list of reads
This code assume that all reads with the same names have the same flag (map/unmap reads)

Usage:
python3 PS8.py <file.sam> > count_reads_map_notmap.txt
'''
def parseHeader(file):
    fin = open(file, 'r')
    lines = fin.readlines()
    reads_dict = dict()
    for count, line in enumerate(lines):
        if line.startswith ('@'):
            pass
        else:
            read_split = line.split('\t')
            flag = read_split[1]
            name = read_split[0]
            name = name + str(count)
            reads_dict[name] = flag
    return reads_dict

'''
Check if current read is map
Given bitwise flag
'''
def mapCheck(flag):
    mapped = False
    if((flag & 4) != 4):
      mapped = True
    return mapped

def checkPrimary(flag):
    primary = False
    if((flag & 256) == 256):
      primary = True
    return primary

'''
Return number of mapped and unmapped reads
'''
def mapCount(reads_dict):
    numMapped = 0
    numUnmmaped = 0

    for name, flag in reads_dict.items():

        if checkPrimary(int(flag)):
            continue
        if mapCheck(int(flag)):
```

```
            numMapped += 1
        else:
            numUnmmaped += 1
    return numMapped, numUnmmaped

'''
Main Function
'''
def main():
    samfile = sys.argv[1]
    reads_dict = parseHeader(samfile)
    #pprint.pprint(reads_dict)
    numMapped, numUnmmaped = mapCount(reads_dict)
    Total = numMapped + numUnmmaped
    print('Total: {}'.format(Total))
    print('Number of Mapped Reads: {}, Number of Unmapped Reads {}'.format(numMapped, numUnmmaped))

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

Output:

```
For 8_2F_fox_S7
Total: 69582314
Number of Mapped Reads: 69582314, Number of Unmapped Reads 0

For 14_3B_control_S10
Total: 8493304
Number of Mapped Reads: 8312390, Number of Unmapped Reads 180914
```

QUESTION 11

```
conda install -c bioconda samtools
```

Script to sort SAM files before htseq

```
cat sort_sam_run.sh
```

```
#!/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/8_2F_fox_S7_L008_Aligned.out.sam | \
samtools sort -n | \
samtools view -h -o aligned/8_2F_fox_S7_L008_Aligned.sorted.sam

/usr/bin/time -v samtools view -u aligned/14_3B_control_S10_L008_Aligned.out.sam | \
samtools sort -n | \
samtools view -h -o aligned/14_3B_control_S10_L008_Aligned.sorted.sam

exit
```

Script to run htseq

```
cat htseq_count_run.sh
```

```
#!/bin/bash

#SBATCH --account=bgmp
#SBATCH --partition=bgmp
#SBATCH --cpus-per-task=1
#SBATCH --time=1-0:00:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --nodelist=n226         ### Run on node 226

conda activate QAA

/usr/bin/time -v htseq-count -s yes \
aligned/8_2F_fox_S7_L008_Aligned.sorted.sam mus/Mus_musculus.GRCm39.107.gtf \
> aligned/8_2F_fox_S7_L008.stranded.genecount

/usr/bin/time -v htseq-count -s reverse \
aligned/8_2F_fox_S7_L008_Aligned.sorted.sam mus/Mus_musculus.GRCm39.107.gtf \
> aligned/8_2F_fox_S7_L008.reverse.genecount

/usr/bin/time -v htseq-count -s yes \
aligned/14_3B_control_S10_L008_Aligned.sorted.sam mus/Mus_musculus.GRCm39.107.gtf \
> aligned/14_3B_control_S10_L008.stranded.genecount

/usr/bin/time -v htseq-count -s reverse \
aligned/14_3B_control_S10_L008_Aligned.sorted.sam mus/Mus_musculus.GRCm39.107.gtf \
> aligned/14_3B_control_S10_L008.reverse.genecount

exit
```

QUESTION 12

Script to analyze htseq output for strand specificity

```
cat analyze_htseq_count_run.sh
```

```
#!/bin/bash

#SBATCH --account=bgmp
#SBATCH --partition=bgmp
#SBATCH --cpus-per-task=1
#SBATCH --time=1-0:00:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --nodelist=n226         ### Run on node 226

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

# FOX STRANDED
echo '8_2F_fox_S7_L008.stranded.genecount:' > aligned/htseq_count_stats.txt
echo 'reads mapping to feature:' >> aligned/htseq_count_stats.txt
grep -v '^_' aligned/8_2F_fox_S7_L008.stranded.genecount | awk '{sum+=$2}END{print sum}' >> aligned/htseq_cou
```

```
echo 'total reads:' >> aligned/htseq_count_stats.txt
awk '{sum+=$2}END{print sum}' aligned/8_2F_fox_S7_L008.stranded.genecount >> aligned/htseq_count_stats.txt
echo '' >> aligned/htseq_count_stats.txt

#FOX REVERSE
echo '8_2F_fox_S7_L008.reverse.genecount:' >> aligned/htseq_count_stats.txt
echo 'reads mapping to feature:' >> aligned/htseq_count_stats.txt
grep -v '^_' aligned/8_2F_fox_S7_L008.reverse.genecount | awk '{sum+=$2}END{print sum}' >> aligned/htseq_coun
echo 'total reads:' >> aligned/htseq_count_stats.txt
awk '{sum+=$2}END{print sum}' aligned/8_2F_fox_S7_L008.reverse.genecount >> aligned/htseq_count_stats.txt
echo '' >> aligned/htseq_count_stats.txt

#CONTROL STRANDED
echo '14_3B_control_S10_L008.stranded.genecount:' >> aligned/htseq_count_stats.txt
echo 'reads mapping to feature:' >> aligned/htseq_count_stats.txt
grep -v '^_' aligned/14_3B_control_S10_L008.stranded.genecount | awk '{sum+=$2}END{print sum}' >> aligned/hts
echo 'total reads:' >> aligned/htseq_count_stats.txt
awk '{sum+=$2}END{print sum}' aligned/14_3B_control_S10_L008.stranded.genecount >> aligned/htseq_count_stats.
echo '' >> aligned/htseq_count_stats.txt

#CONTROL REVERSE
echo '14_3B_control_S10_L008.reverse.genecount:' >> aligned/htseq_count_stats.txt
echo 'reads mapping to feature:' >> aligned/htseq_count_stats.txt
grep -v '^_' aligned/14_3B_control_S10_L008.reverse.genecount | awk '{sum+=$2}END{print sum}' >> aligned/htse
echo 'total reads:' >> aligned/htseq_count_stats.txt
awk '{sum+=$2}END{print sum}' aligned/14_3B_control_S10_L008.reverse.genecount >> aligned/htseq_count_stats.t

exit
```

Output:

```
8_2F_fox_S7_L008.stranded.genecount:
reads mapping to feature:
1282235
total reads:
34791157

8_2F_fox_S7_L008.reverse.genecount:
reads mapping to feature:
28041293
total reads:
34791157

14_3B_control_S10_L008.stranded.genecount:
reads mapping to feature:
167859
total reads:
4246652

14_3B_control_S10_L008.reverse.genecount:
reads mapping to feature:
3667140
total reads:
4246652
```

I can propose that these libraries are indeed strand specific as 80% of the fox reads mapped to features on the reverse strand and 86% of the control reads mapped to features on the reverse strand. This tells us that a majority of the reads are specific to the reverse strand. If this was an unstranded kit, we would have roughly the same amount on both the forward and reverse strand.