BI 622: Index Hopping

Sally Claridge

13 September 2017

How I Made My Test Files

I used subsets of the test data from `/projects/bgmp/test_data` and `index.tsv` to develop some python script (`index_hopping.py`) in a Jupyter notebook (called `index_hopping.ipynb`) on my local machine. I did not include my python script in this markdown because it was too long, so see separate `index_hopping.py` file.

```
head -10000 Undetermined_S0_R1_001_trunc.fastq > R1_test.fastq
head -10000 Undetermined_S0_R2_001_trunc.fastq > R2_test.fastq
head -10000 Undetermined_S0_R3_001_trunc.fastq > R3_test.fastq
head -10000 Undetermined_S0_R4_001_trunc.fastq > R4_test.fastq
```

I manually edited the `indexes.txt` file provided in `/projects/bgmp/2017_sequencing` to match the format in the script I wrote using the test indices file, `index.tsv`, that only had two columns and did not have a column header line (used `ctrl+v+tab` to make the tab character in the cut call):

```
cat indexes.txt | grep -v "^sample" | cut -d "  " -f 4,5 > indices_new.tsv
```

Made symbolic links to FASTQ files in my working directory:

```
ln -s /projects/bgmp/2017_sequencing/1294_S1_L008_R1_001.fastq /home/sclaridg/bi622/R1.fastq
ln -s /projects/bgmp/2017_sequencing/1294_S1_L008_R2_001.fastq /home/sclaridg/bi622/R2.fastq
ln -s /projects/bgmp/2017_sequencing/1294_S1_L008_R3_001.fastq /home/sclaridg/bi622/R3.fastq
ln -s /projects/bgmp/2017_sequencing/1294_S1_L008_R4_001.fastq /home/sclaridg/bi622/R4.fastq
```

I made test files from the sequencing files to run my `index_hopping.py` script on (which includes commands to account for reverse complements in the indices). I moved these test files to my local machine to use in conjunction with Jupyter notebook:

```
head -10000 R1.fastq > R1_test.fastq
head -10000 R2.fastq > R2_test.fastq
head -10000 R3.fastq > R3_test.fastq
head -10000 R4.fastq > R4_test.fastq
```

Part 1

Instructions: Generate per base call distribution of quality scores for read1, read2, index1, and index2. Generate a per nucleotide distribution as you did in part 1 of PS4 (in Leslie's class). Next, average the Quality scores for each read (for each of the four files) and plot frequency of the Quality Scores.

Obtaining Read Stats: Command Line and `reads_stats.py`

I checked the length of the original files, which yielded 1,452,986,940 lines. This corresponds to 363,246,735 reads!

```
$ wc -l R1.fastq
1452986940
```

I made a script called `stats_onefile.py` (see GitHub repository for the script) to calculate the quality score distributions for a single FASTQ file. Here is the slurm script, `R1_stats.srun`, to run the python script on `R1.fastq`. I repeated this slurm script for `R4.fastq`, and changed the `--read_length` flag to 8 for `R2.fastq` and `R3.fastq`.

```
#!/bin/bash
#SBATCH --partition=long
#SBATCH --job-name=R1_stats
#SBATCH --output=/home/sclaridg/bi622/logs/R1_stats.out
#SBATCH --error=/home/sclaridg/bi622/logs/R1_stats.err
#SBATCH --time=2-00:00:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=28
#SBATCH --mail-user=sclaridg@uoregon.edu
#SBATCH --mail-type=ALL

# Load modules
ml  easybuild  GCC/6.3.0-2.27  OpenMPI/2.0.2 Python/3.6.1
ml list

# Sanity check
echo "This is SLURM: Starting python script."

# Call python script
/home/sclaridg/bi622/stats_onefile.py --read_file /home/sclaridg/bi622/R1.fastq
--read_length 101 --output_bpqual /home/sclaridg/bi622/R1_bp.tsv
--output_readqual /home/sclaridg/bi622/R1_rq.tsv

exit
```

Question 1A: Turn in Plots

Read In Output from `stats_onefile.py`

These files were the output from my quality distribution script. `bp` files contain per-base mean quality score across the file, and `rq` files contain the count distributions for per-read mean quality scores.

Set working directory:

```
setwd("/Users/sally_claridge/Desktop/index_hopping/")
```

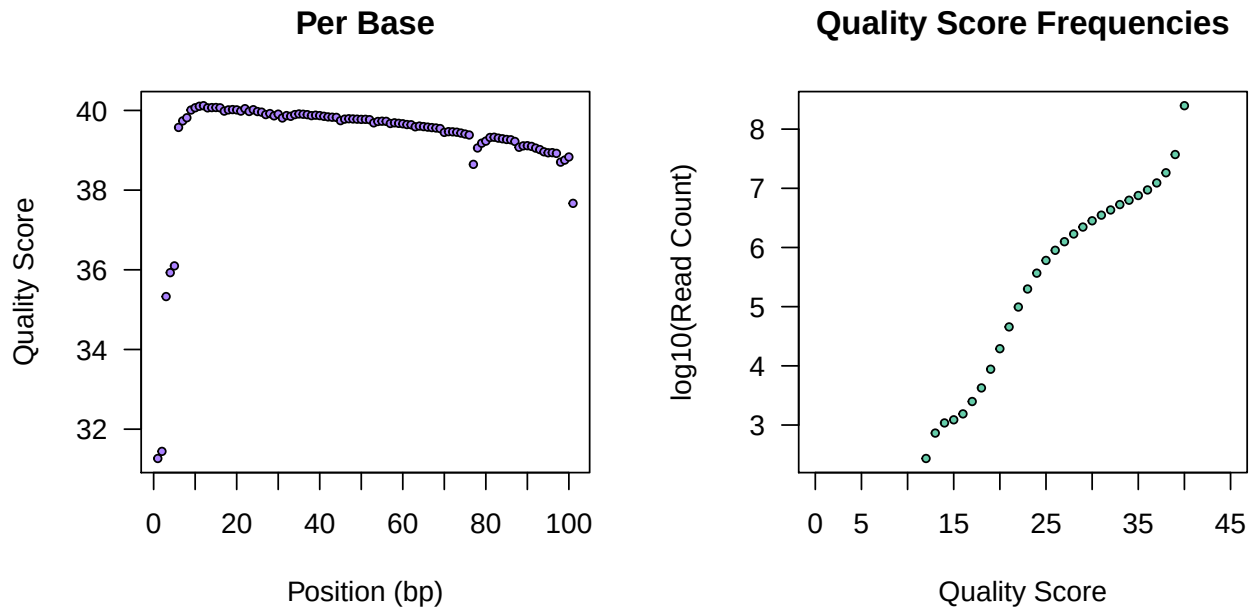
Read in the data:

```
R1_bp <- read.table("./outputs/R1_bp.tsv", header = TRUE, sep = "\t")
R1_rq <- read.table("./outputs/R1_rq.tsv", header = TRUE, sep = "\t")

R2_bp <- read.table("./outputs/R2_bp.tsv", header = TRUE, sep = "\t")
R2_rq <- read.table("./outputs/R2_rq.tsv", header = TRUE, sep = "\t")

R3_bp <- read.table("./outputs/R3_bp.tsv", header = TRUE, sep = "\t")
R3_rq <- read.table("./outputs/R3_rq.tsv", header = TRUE, sep = "\t")

R4_bp <- read.table("./outputs/R4_bp.tsv", header = TRUE, sep = "\t")
R4_rq <- read.table("./outputs/R4_rq.tsv", header = TRUE, sep = "\t")
```

R1 Plots

Make dotplots for `R1_bp.tsv`, a per base call distribution of quality scores, and for `R1_rq.tsv`, a per-read average quality score distribution:

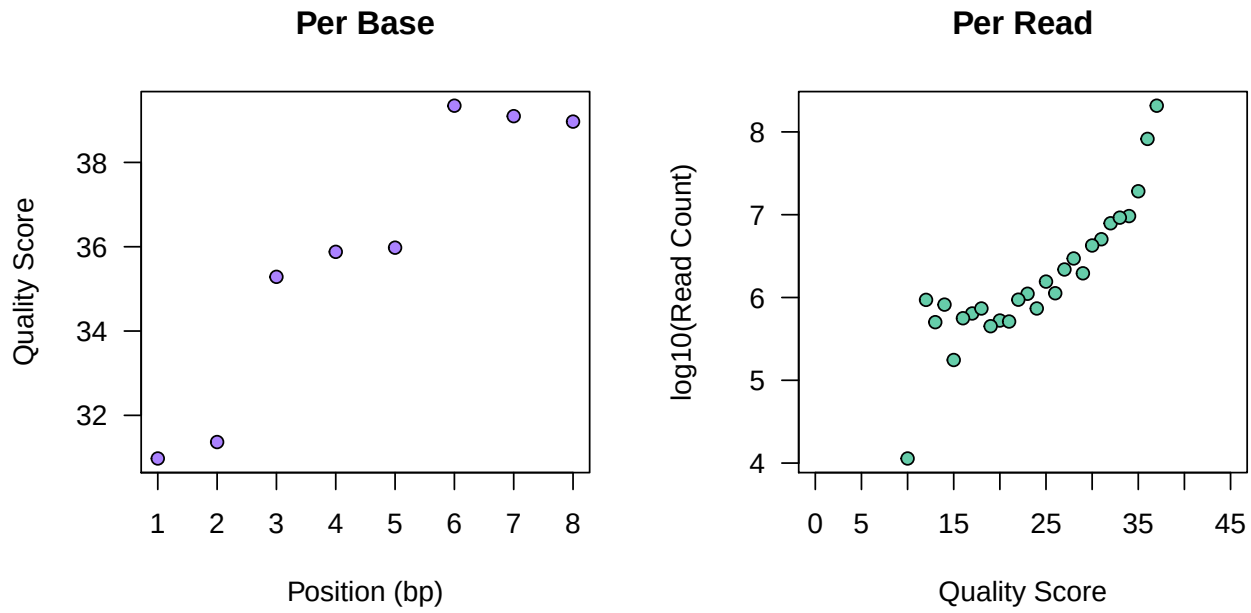
```
par(mfrow = c(1,2), oma=c(0,1,2,0))
# Per base call
plot(R1_bp$Base_Pair, R1_bp$Mean_Score,
     xlab = "Position (bp)",
     ylab = "Quality Score",
     pch = 21,
     bg = "mediumpurple1",
     cex = 0.6,
     xaxt = "n",
     yaxt = "n",
     main = "Per Base")
axis(side = 1, at = seq(0, 100, by = 10), labels = seq(0, 100, by = 10))
axis(side = 2, las = 2)
# Per read mean quality
plot(R1_rq$Quality_Score, log10(R1_rq$Read_Count),
     xlab = "Quality Score",
     ylab = "log10(Read Count)",
     xlim = c(0, 45),
     pch = 21,
     bg = "mediumaquamarine",
     cex = 0.6,
     xaxt = "n",
     yaxt = "n",
     main = "Quality Score Frequencies")
axis(side = 1, at = seq(0, 45, by = 5), labels = seq(0, 45, by = 5))
axis(side = 2, las = 2)
mtext("Mean Quality Score Distributions for R1", font = 2, side = 3,
      line = 0, outer = TRUE, cex = 1.5)
```



R2 Plots

Make dotplot for `R2_bp.tsv`, per base call distribution of quality scores, and for `R2_rq.tsv`, a per-read average quality score distribution:

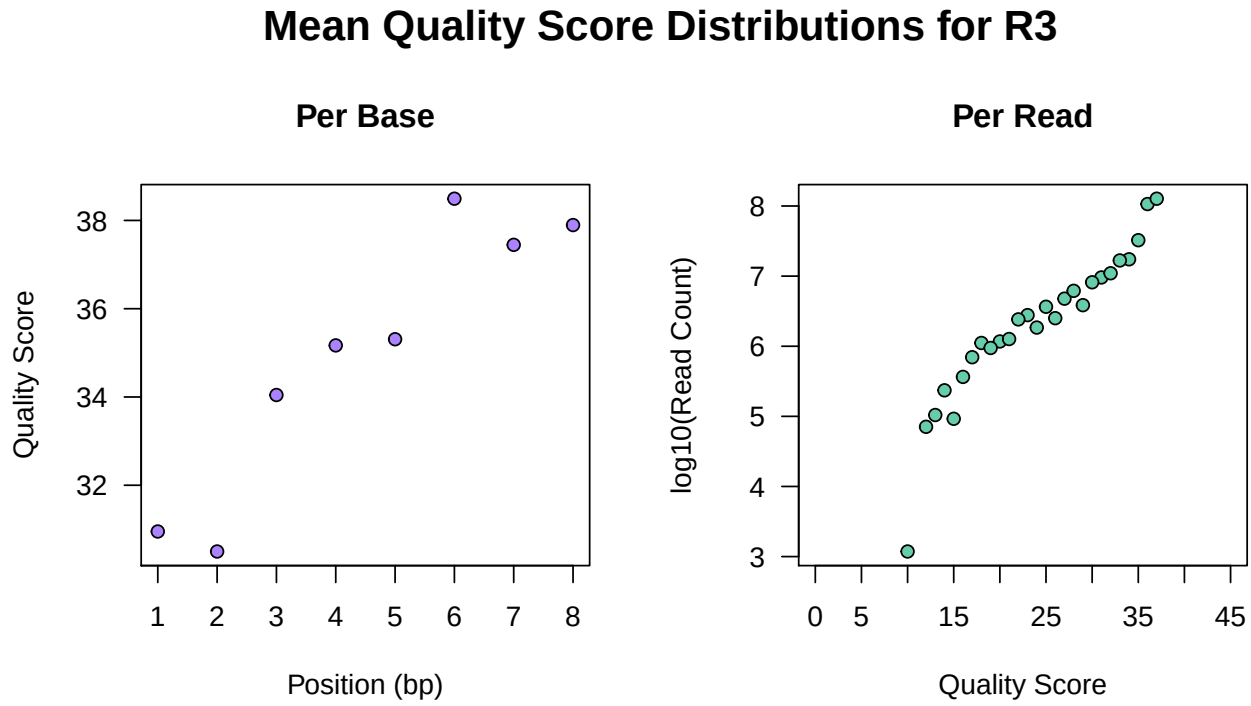
```
par(mfrow = c(1,2), oma=c(0,1,2,0))
# Per base call
plot(R2_bp$Base_Pair, R2_bp$Mean_Score,
     xlab = "Position (bp)",
     ylab = "Quality Score",
     pch = 21,
     bg = "mediumpurple1",
     xaxt = "n",
     yaxt = "n",
     main = "Per Base")
axis(side = 1, at = seq(0, 8, by = 1), labels = seq(0, 8, by = 1))
axis(side = 2, las = 2)
# Per read mean quality
plot(R2_rq$Quality_Score, log10(R2_rq$Read_Count),
     xlab = "Quality Score",
     ylab = "log10(Read Count)",
     xlim = c(0, 45),
     pch = 21,
     bg = "mediumaquamarine",
     xaxt = "n",
     yaxt = "n",
     main = "Per Read")
axis(side = 1, at = seq(0, 45, by = 5), labels = seq(0, 45, by = 5))
axis(side = 2, las = 2)
mtext("Mean Quality Score Distributions for R2", font = 2, side = 3,
      line = 0, outer = TRUE, cex = 1.5)
```



R3 Plots

Make dotplot for `R3_bp.tsv`, per base call distribution of quality scores, and for `R3_rq.tsv`, a per-read average quality score distribution:

```
par(mfrow = c(1,2), oma=c(0,1,2,0))
# Per base call
plot(R3_bp$Base_Pair, R3_bp$Mean_Score,
     xlab = "Position (bp)",
     ylab = "Quality Score",
     pch = 21,
     bg = "mediumpurple1",
     xaxt = "n",
     yaxt = "n",
     main = "Per Base")
axis(side = 1, at = seq(0, 8, by = 1), labels = seq(0, 8, by = 1))
axis(side = 2, las = 2)
# Per read mean quality
plot(R3_rq$Quality_Score, log10(R3_rq$Read_Count),
     xlab = "Quality Score",
     ylab = "log10(Read Count)",
     xlim = c(0, 45),
     pch = 21,
     bg = "mediumaquamarine",
     xaxt = "n",
     yaxt = "n",
     main = "Per Read")
axis(side = 1, at = seq(0, 45, by = 5), labels = seq(0, 45, by = 5))
axis(side = 2, las = 2)
mtext("Mean Quality Score Distributions for R3", font = 2, side = 3,
      line = 0, outer = TRUE, cex = 1.5)
```

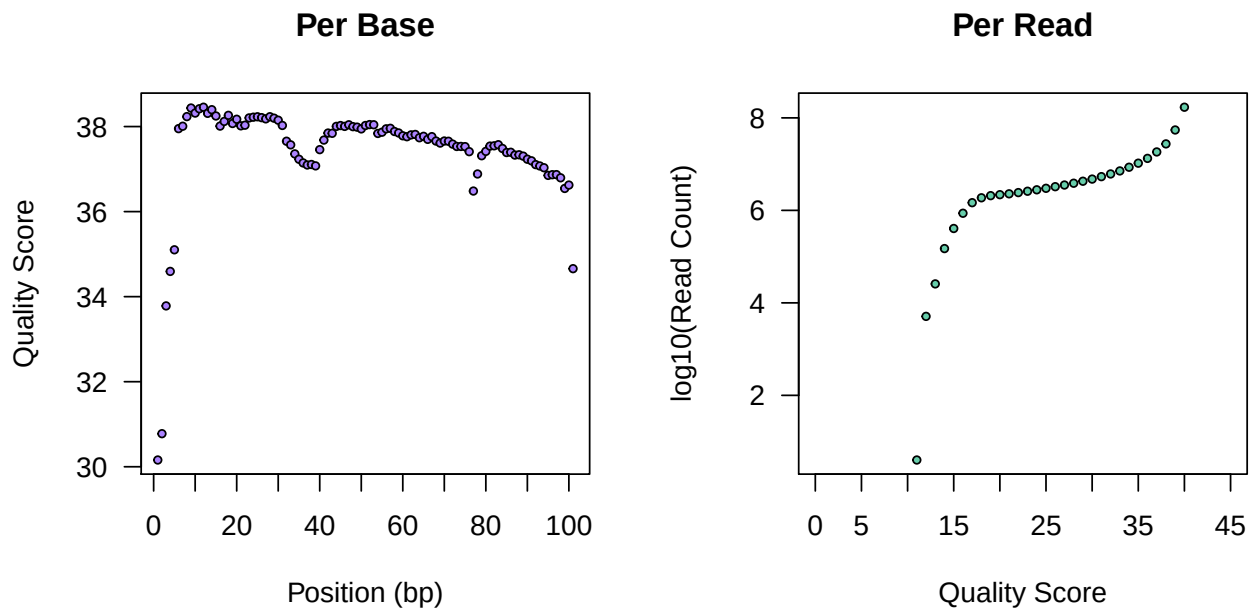


R4 Plots

Make dotplot for `R4_bp.tsv`, per base call distribution of quality scores, and for `R4_rq.tsv`, a per-read average quality score distribution::

```
par(mfrow = c(1,2), oma=c(0,1,2,0))
# Per base call
plot(R4_bp$Base_Pair, R4_bp$Mean_Score,
     xlab = "Position (bp)",
     ylab = "Quality Score",
     pch = 21,
     bg = "mediumpurple1",
     cex = 0.6,
     xaxt = "n",
     yaxt = "n",
     main = "Per Base")
axis(side = 1, at = seq(0, 100, by = 10), labels = seq(0, 100, by = 10))
axis(side = 2, las = 2)
# Per read mean quality
plot(R4_rq$Quality_Score, log10(R4_rq$Read_Count),
     xlab = "Quality Score",
     ylab = "log10(Read Count)",
     xlim = c(0, 45),
     pch = 21,
     bg = "mediumaquamarine",
     cex = 0.6,
     xaxt = "n",
     yaxt = "n",
     main = "Per Read")
axis(side = 1, at = seq(0, 45, by = 5), labels = seq(0, 45, by = 5))
axis(side = 2, las = 2)
```

```
mtext("Mean Quality Score Distributions for R4", font = 2, side = 3, line = 0, outer = TRUE, cex = 1.5)
```



Question 1B: Quality Score Cutoffs

What is a good quality score cutoff for index reads and pairs to utilize for sample identification and downstream analysis, respectively?

For sample identification, I think a minimum quality score of 30 is on the high end, since the lowest mean nucleotide scores for the indices were just a fraction above 30. Thus, I chose 25 as a cutoff, since it seems reasonable to not want very low quality reads, but is low enough to capture those reads that are just under the lowest mean quality score observed in the indices. For the sake of seeing what happens, I also ran the index hopping script with a minimum quality score of 30. A quick Google search revealed that 30 is a common quality minimum threshold for data sets with high coverage. I'm making an assumption that our coverage is very high (which I will not know in the scope of this assignment), but I think it will be interesting to see what the data looks like when filtered that stringently.

For downstream analyses of the R1 and R4 sequence reads, I think 30 is also a good quality cut off for the same reasons as above. The per-base mean quality score was above 30 for all bases for both R1 and R4. Q30 gives a 99.9% inferred base call accuracy, meaning that 1 in every 1,000 bases might be called incorrectly. Accoring to Illumina's technical note on sequencing, Q30 is a benchmark for for NGS sequencing since basically all reads won't have any zeros or ambiguities.

Question 1C: How many indexes have Undetermined (N) base calls?

Instructor Note: Utilize your command line tool knowledge. Submit the command you used. CHALLENGE: use a one line command.

Prior to filtering, I used the shell commands below to determine how many indexes have **Undetermined (N)** base calls. `R2.fastq` contains 3,976,613 indices containing an "N" and `R3.fastq` contains 3,328,051 indices.

```
$ awk '(NR%4==2)' R2.fastq | grep "N" | wc -l
3976613

$ awk '(NR%4==2)' R3.fastq | grep "N" | wc -l
3328051
```

My `index_hopping2.py` script contains code that makes a dictionary of index pairs that contain at least one N and the counts for each of these pairs. Theoretically, this means that one index could have zero Ns while the other has eight. None of the index pairs contained an N.

Question 1D

What do the averaged Quality Scores across the reads tell you? Interpret your data specifically.

In theory, had we tracked the mean read quality over time, this metric tells us how the read quality changes over time throughout the run, kind of like a temporal check. This also will provide information on the distribution of low quality bases across a read, e.g. if low quality bases are present only in some of the reads or in all of the reads. In the per-read plots for all reads (1 trough 4), we see that there is a data point over a Q10. Having a subset of sequences with low average quality score is normal, but it should be only a small percentage of the reads. From the plots, we can see that there are orders of magnitude more reads that fall in the range of Q20 and greater, which is a low but acceptable quality score (for some studies).

Part 2

Instructions: Write a program to de-multiplex the samples and document index swapping and number of reads retained per sample.

Run `index_hopping.py` Script

Made an `index_hopping.srun` script to run `index_hopping.py`, which will demultiplex the samples and produce a TSV file that contains index pair counts. I ran this script with the `--min_qual` set to both 25 and 30. My index hopping script used a very stringent filtering method where if a single base in either one of index reads had a quality score below the `min_qual` flagged value, then the read was discarded and not included in the final count of expected and unexpected index pair reads. This stringent filtering method results in more extensive read loss compared to other filtering methods that were utilized by other students on the project, i.e. discarding reads whose mean quality fell below the `min_qual` flag.

Here is the slurm script with a `--min_qual 30` flag. I also used a flag of 25 for a second set of data to compare.

```
#!/bin/bash
#SBATCH --partition=long
#SBATCH --job-name=/home/sclaridg/bi622/index_hopping
#SBATCH --output=/home/sclaridg/bi622/index_hopping.out
#SBATCH --error=/home/sclaridg/bi622/index_hopping.err
#SBATCH --time=2-00:00:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=28
#SBATCH --mail-user=sclaridg@uoregon.edu
#SBATCH --mail-type=ALL

# Load modules
ml  easybuild  GCC/6.3.0-2.27  OpenMPI/2.0.2 Python/3.6.1
```

```
ml list

# Sanity check
echo "This is SLURM: Starting python script."

# Set dir variable to make command easier to read
dir="/home/sclaridg/bi622"

# Run index_hopping.py for min_qual of 30
python index_hopping.py --read2 $dir/R2.fastq --read3 $dir/R3.fastq
--min_qual 30 --indices $dir/indices_new.tsv --output $dir/ih30_out.tsv

exit
```

The output has this format:

```
@@@EXP_INDEX_PAIRS___PAIRS_ARE_ASSOCIATED_WITH_SAMPLES
expected    GTAGCGTA_GTAGCGTA   6613841
expected    CGATCGAT_CGATCGAT   4751884
# 22 more expected index pairs
@@@UNEXP_INDEX_PAIRS___POTENTIAL_INDEX_SWAPPING
unexpected  GTAGCGTA_CGATCGAT   111
unexpected  GTAGCGTA_GATCAAGG   124
# 274 more unexpected index pairs
@@@N_CONTAINING_INDEX_PAIRS
# None of these were observed with either min_qual flag
```

I parsed my `index_hopping2.py` output and saved expected and unexpected output for both quality score minimums:

```
grep "^expected" ih25_out.tsv > ih25_expected.tsv
grep "^expected" ih30_out.tsv > ih30_expected.tsv
grep "^unexpected" ih30_out.tsv > ih30_unexpected.tsv
grep "^unexpected" ih25_out.tsv > ih25_unexpected.tsv
```

Read the files in to R:

```r
# Read files
exp25 <- read.table("./outputs/ih25_expected.tsv", sep = "\t")
exp30 <- read.table("./outputs/ih30_expected.tsv", sep = "\t")

unexp25 <- read.table("./outputs/ih25_unexpected.tsv", sep = "\t")
unexp30 <- read.table("./outputs/ih30_unexpected.tsv", sep = "\t")

# Rename columns
colnames(exp25) <- c("expected", "index_pair", "count")
colnames(exp30) <- c("expected", "index_pair", "count")
colnames(unexp25) <- c("unexpected", "index_pair", "count")
colnames(unexp30) <- c("unexpected", "index_pair", "count")
```

Question 2A

How many reads are retained for each expected index pair? What is the percentage?

I created a merged data frame of the output from above, counted how many reads were retained for each expected index pair, and calculated the percentage of read that were retained. I used vectorized calculations

that are a feature of R. The script in the box below swhos my calculations and prints out a table of expected index pairs, their counts for the two quality cutoffs, and the percentage retained.

```
# Merge expected index pair counts from the two quality score cutoffs
all_exp <- merge(exp25, exp30, by = "index_pair")

# Subset table to remove the "expected columns" that were used for grepping
all_exp_sub <- subset(all_exp, select = c("index_pair", "count.x", "count.y"))

# Calculate percentages on now-merged data frame
total_reads <- 363246735 # Calculated from command line
all_exp_sub$perc_25 <- all_exp_sub$count.x / total_reads * 100
all_exp_sub$perc_30 <- all_exp_sub$count.y / total_reads * 100

# Rename columns
colnames(all_exp_sub) <- c("Expected Index Pair","Counts Q25", "Counts Q30",
                           "Percent Retained Q25", "Percent Retained Q30")

# Print table
library(knitr)
kable(all_exp_sub, align = c(rep('c', 5)), caption = "Read Counts Observed for Each Expected Index Pair
```

Table 1: Read Counts Observed for Each Expected Index Pair Under Two Quality (Q) Score Cutoffs

Expected Index Pair	Counts Q25	Counts Q30	Percent Retained Q25	Percent Retained Q30
AACAGCGA_AACAGCGA	7164150	6368144	1.9722545	1.7531180
ACGATCAG_ACGATCAG	6691726	5933528	1.8421985	1.6334704
AGAGTCCA_AGAGTCCA	9038127	7602663	2.4881509	2.0929749
AGGATAGC_AGGATAGC	7097978	5861709	1.9540377	1.6136990
ATCATGCG_ATCATGCG	8136952	6927867	2.2400620	1.9072070
ATCGTGGT_ATCGTGGT	5547098	4730009	1.5270882	1.3021477
CACTTCAC_CACTTCAC	3197780	2577666	0.8803328	0.7096185
CGATCGAT_CGATCGAT	4751884	4237854	1.3081698	1.1666599
CGGTAATC_CGGTAATC	3514828	2393021	0.9676145	0.6587867
CTAGCTCA_CTAGCTCA	14756290	13034311	4.0623325	3.5882803
CTCTGGAT_CTCTGGAT	28358707	24515040	7.8070095	6.7488673
GATCAAGG_GATCAAGG	5338704	4628196	1.4697184	1.2741191
GATCTTGC_GATCTTGC	3070473	2636332	0.8452858	0.7257689
GCTACTCT_GCTACTCT	5424215	4301318	1.4932591	1.1841312
GTAGCGTA_GTAGCGTA	6613841	5774439	1.8207572	1.5896740
GTCCTAAG_GTCCTAAG	7193114	6200133	1.9802281	1.7068654
TACCGGAT_TACCGGAT	58427310	49686876	16.0847447	13.6785472
TAGCCATG_TAGCCATG	8655836	7148151	2.3829081	1.9678500
TATGGCAC_TATGGCAC	8925572	7651472	2.4571651	2.1064118
TCGACAAG_TCGACAAG	3083259	2644260	0.8488057	0.7279515
TCGAGAGT_TCGAGAGT	8892653	7448070	2.4481027	2.0504162
TCGGATTC_TCGGATTC	3501066	2874320	0.9638259	0.7912858
TCTTCGAC_TCTTCGAC	34584564	30089660	9.5209566	8.2835321
TGTTCCGT_TGTTCCGT	13284156	11450553	3.6570614	3.1522797

I was curious to see what percentage of the total reads were assigned to expected index pairs under both quality filters, so I ran the code below to sum up the percentages from the table above.

```
perc25_all <- sum(all_exp_sub$`Percent Retained Q25`)
print(paste("The percent of total reads retained for Q25 was", perc25_all, "%"))
```

```
## [1] "The percent of total reads retained for Q25 was 73.0220694206653 %"
```

```
perc30_all <- sum(all_exp_sub$`Percent Retained Q30`)
print(paste("The percent of total reads retained for Q30 was", perc30_all, "%"))
```

```
## [1] "The percent of total reads retained for Q30 was 62.413662713307 %"
```

By eye, I can see that the counts (and therefore percentage retained) in the table above are larger for the Q25 cutoff than for the Q30 cutoff. It so follows that the total percentage retained for the Q25 cutoff is greater than for the Q30 cutoff (~73% compared to 62%). These numbers make sense because a lower quality score cutoff should yield a greater number of retained reads since more of the reads should pass the filter.

Question 2B

How many reads are indicative of index swapping?

I summed the counts for unexpected index pairs for a quality score cutoff of 25 and 30. This resulted in 225,712 reads for Q25 and 179,624 reads for Q30.

```
sum(unexp25$count)
```

```
## [1] 225712
```

```
sum(unexp30$count)
```

```
## [1] 179624
```

Question 2C: Create a distribution of swapped indexes

What does this histogram tell you/what is your interpretation of this data?

These histograms tell me that there was some level of index hopping. To think more about quantifying the amount of index hopping, I calculated the percentage of reads that had index hopping out of all reads and out of reads in the unexpected and expected dictionaries, i.e. non-N-containing reads.

The percentages (calculated below) tell me that a small percentage (less than 0.1%) of reads that did not contain Ns (thus were in the expected or unexpected index pair dictinaries) experienced index hopping. In the grand scheme of things, this seems like a low percentage that still leaves behind enough reads assigned to each sample.

Make Plots

I used RColorBrewer to set up a color vector to use in my plots, which allows for the gradient coloring scheme:

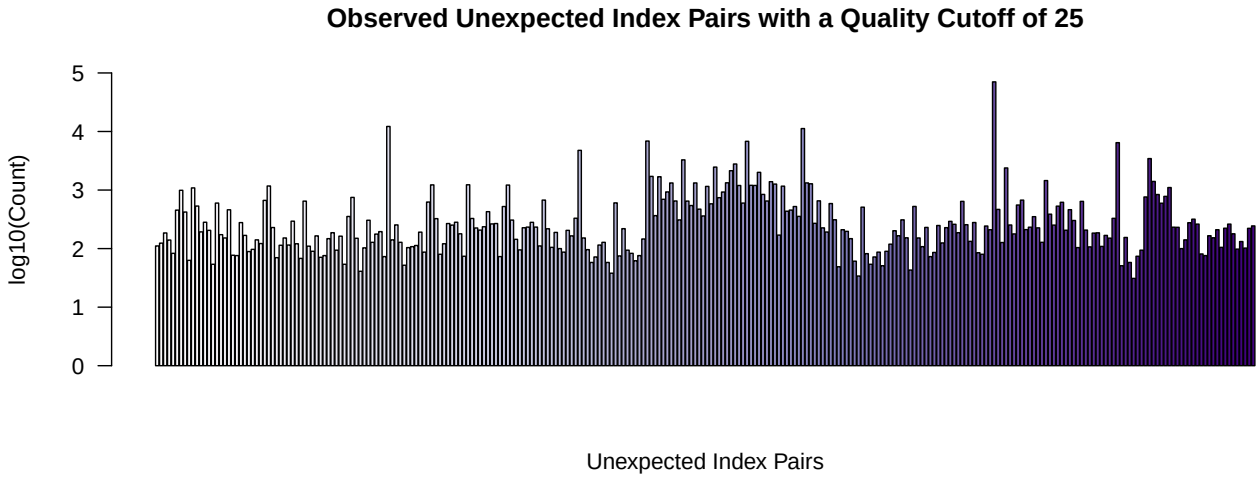
```
if (!require("RColorBrewer")) {
  install.packages("RColorBrewer")
  library(RColorBrewer)
}
```

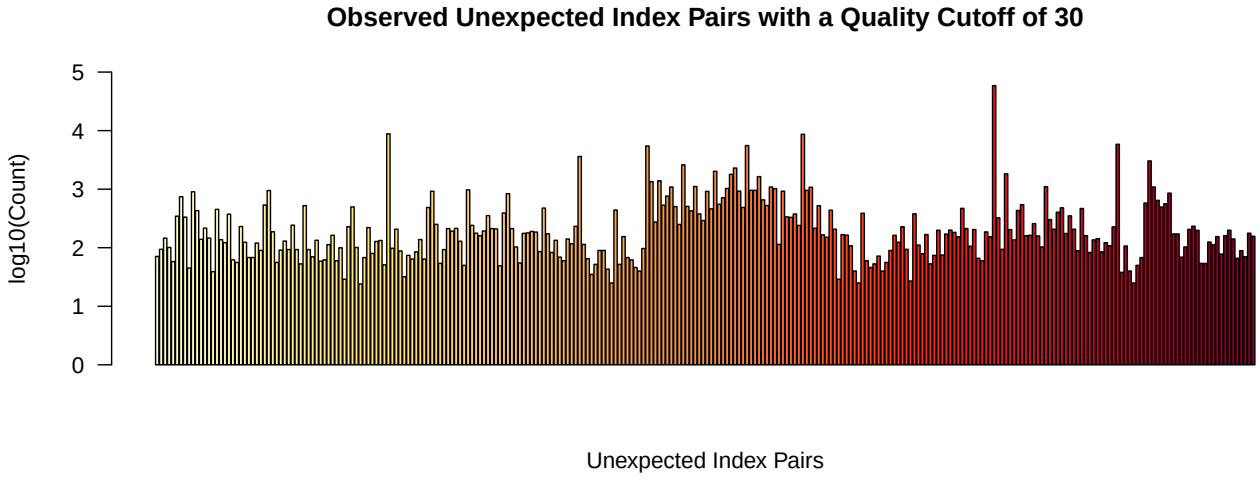
```
## Loading required package: RColorBrewer
```

```
# Make vector of colors
colors25 <- colorRampPalette(brewer.pal(9,"Purples"))(276)
colors30 <- colorRampPalette(brewer.pal(9,"YlOrRd"))(276)
```

Here is the code for the plot unexpected index pair counts. I removed the x-axis labels containing the index pairs due to messiness and cluttering.

```
par(mfrow = c(2,1))
# Quality score cutoff of 25
barplot(height = log10(unexp25$count),
     xlab = "Unexpected Index Pairs",
     ylab = "log10(Count)",
     col = colors25,
     ylim = c(0,5),
     yaxt = "n",
     xaxt = "n",
     main = "Observed Unexpected Index Pairs with a Quality Cutoff of 25")
axis(side = 2, at = c(0:5), labels = c(0:5), las = 2)
# Quality score cutoff of 25
barplot(height = log10(unexp30$count),
     xlab = "Unexpected Index Pairs",
     ylab = "log10(Count)",
     col = colors30,
     ylim = c(0,5),
     yaxt = "n",
     xaxt = "n",
     main = "Observed Unexpected Index Pairs with a Quality Cutoff of 30")
axis(side = 2, at = c(0:5), labels = c(0:5), las = 2)
```





Calculating Percentages

Below is the (large) amount of data maneuvering I did to create a data frame/table of the percentage of total reads and of non-N-containing reads that fell within index hopping:

```
# Calculate % out of all reads
total_reads <- 363246735 # Calculated from command line
prop25_tot <- sum(unexp25$count) / total_reads * 100
prop30_tot <- sum(unexp30$count) / total_reads * 100

# Calculate % out of reads in the unexpected and expected dictionaries
# (non-N-containing)
dict_reads25 <- sum(unexp25$count) + sum(exp25$count)
dict_reads30 <- sum(unexp30$count) + sum(exp30$count)
prop25_dict <- sum(unexp25$count) / dict_reads25 * 100
prop30_dict <- sum(unexp30$count) / dict_reads30 * 100

# Make vector to build data frame
qual <- c("25", "30", "25", "30")
readcount <- c(sum(unexp25$count), sum(unexp30$count),
               sum(unexp25$count), sum(unexp30$count))
```

```
out_of <- c("All Reads", "All Reads", "Non-N Reads Only", "Non-N Reads Only")
tot_readcount <- c(total_reads, total_reads, dict_reads25, dict_reads30)
props <- c(prop25_tot, prop30_tot, prop25_dict, prop30_dict)


# Make data frame of unexpected reads
props_df <- cbind.data.frame(qual, readcount, out_of, tot_readcount, props)
colnames(props_df) <- c("Quality Cutoff", "Unexpected Readcounts",
                        "Out Of", "Total Readcounts", "Percent (%)")

# Print results
kable(props_df, align = c(rep('c', 5)), caption = "Proportion of Reads that had Unexpected Index Pairs")
```

Table 2: Proportion of Reads that had Unexpected Index Pairs

Quality Cutoff	Unexpected Readcounts	Out Of	Total Readcounts	Percent (%)
25	225712	All Reads	363246735	0.0621374
30	179624	All Reads	363246735	0.0494496
25	225712	Non-N Reads Only	265475995	0.0850216
30	179624	Non-N Reads Only	226895216	0.0791661