Maryam Hosseinali - 610398209

Bioinformatics project1 report

Part A - (Q1)

1. Installing the SRA Toolkit:
   I installed the SRA Toolkit on my Ubuntu computer with the help of a tutorial video Downloading sequencing data on ubuntu/linux - SRA toolkit.
2. Downloading the Data:
   After installing the toolkit, I used a command called fastq-dump to download the data. The command I used was:

```
@maryam:~/Desktop/test$ fastq-dump SRR8185316
```

Part A - (Q2)

First, I uploaded the downloaded FASTQ file, SRR8185316.fastq, into my Jupyter Notebook environment for accessing and analysing the file.

I. How many reads are in the fastq file?

Each read in a FASTQ file is represented by four lines therefore, the total number of reads is equal to the total number of lines in the file divided by four:

part A - (Q2) - I. How many reads are in the fastq file?

```
filename = "SRR8185316.fastq"

with open(filename, 'r') as file:
    number_of_lines = sum(1 for line in file)

number_of_reads = number_of_lines // 4
print(f"Total number of reads: {number_of_reads}")
```

```
Total number of reads: 409268
```

So the total number of reads are: **409268**

II. Print the identifier, quality, and sequence of the first read of the fastq file.

The first line: identifier (starting with '@')
The second line: sequence
The fourth line: quality scores of the sequence.

part A - (Q2) - II. Print the identifier, quality, and sequence of the first read of the fastq file.

```
filename = "SRR8185316.fastq"

with open(filename, 'r') as file:
    identifier = next(file).strip()
    sequence = next(file).strip()
    next(file)  # Skip the separator line
    quality = next(file).strip()

    print(f"Identifier: {identifier}")
    print(f"Sequence: {sequence}")
    print(f"Quality: {quality}")
```

```
Identifier: @SRR8185316.1 ERR022075.10741970 length=100
Sequence: AGCGGTACACATTATGGGTCTGCTCTCCGCAGGCGGCGTACACAGCCACGAAGATCACATCATGGCGATGGTAGAACTGGCAGCTGAACGCGGCGCAGAA
Quality: IIGIIIIIIIIIIIIHIIIIIIIIIIIIIIIIII@IHHEHIIIIIIIIIIIHIIHIIIIIIIIGIHIIHFIIEIIGIHHGH@BE3BB>@>>2?@8?>?A@1
```

Identifier:
@SRR8185316.1 ERR022075.10741970 length=100

Sequence:
AGCGGTACACATTATGGGTCTGCTCTCCGCAGGCGGCGTACACAGCCACGAAGATCACATCATGGCGATGGTAGAACTGGCAGCTGAACGCGGCGCAGAA

Quality:
IIGIIIIIIIIIIIHIIIIIIIIIIIIIIIIII@IHHEHIIIIIIIIIIIHIIHIIIIIIIIGIHIIHFIIEIIGIHHGH@BE3BB>@>>2?@8?>?A@1

III. How many times does the TTAAATGGAA subsequence appear in the file?

```
filename = "SRR8185316.fastq"
subsequence = "TTAAATGGAA"
count = 0

with open(filename, 'r') as file:
    for line_number, line in enumerate(file):
        if (line_number - 1) % 4 == 0:  # selecting sequence lines
            count += line.count(subsequence)

print(f"subsequence '{subsequence}' appears {count} times in the file.")
```

```
subsequence 'TTAAATGGAA' appears 23 times in the file.
```

The subsequence 'TTAAATGGAA' appears 23 times in the file.

IV. Extract the first 1000 sequences of the fastq files (4000 lines).
I extracted the first 4000 lines from the 'SRR8185316.fastq' file and saved them in a new file: "first_1000_sequences.fastq" . The file is available in the attached notebook.

```
filename = "SRR8185316.fastq"
output = "first_1000_sequences.fastq"

with open(filename, 'r') as file:
    with open(output, 'w') as outfile:
        for _ in range(1000 * 4):
            line = next(file)
            outfile.write(line)
```

The following screenshot displays the first 28 lines of the first_1000_sequences.fastq file

```
@SRR8185316.1 ERR022075.10741970 length=100
AGCGGTACACATTATGGGTCTGCTCTCCGCAGGCGGCGTACACAGCCACGAAGATCACATCATGGCGATGGTAGAACTGGCAGCTGAACGCGGCGCAGAA
+SRR8185316.1 ERR022075.10741970 length=100
IIGIIIIIIIIIIIIHIIIIIIIIIIIIIIIII@IHHEHIIIIIIIIIIIHIIHIIIIIIIIGIHIIHFIIEIIGIHHGH@BE3BB>@>>2?@8?>?A@1
@SRR8185316.2 ERR022075.13329732 length=100
TAATGGCCGGCGACCTTATCCGCTGGAAACCATGCTACGCATTCACTGCATGCAGCATTGGTACAACCTGAGCGATGGCGCGATGGAAGATGCTCTGTAC
+SRR8185316.2 ERR022075.13329732 length=100
HHHHHHHHHHDHHDHHHGFGHDGGHG>GGGG<GDGHHEGH@EGGGHBHBHHHGHHFHHHHDEDFFHBHEFHHHH<FHDBH@HCBCG3DDB+?:=8DDBBD
@SRR8185316.3 ERR022075.365654 length=100
ATGGTGGCAACCTGGCCTGCCGATACTAAAGAGGGCTGGCGGCGCTATCGTCTGAAAGGGTCAACCTTGTTAGGTCCCCATCCTGCCGATCCGCTGAAAT
+SRR8185316.3 ERR022075.365654 length=100
=HHDHH>HFDDBGEG@EDEGECACC2A=A>A2<B=8?<;/?############################################################
@SRR8185316.4 ERR022075.14519231 length=100
CTCCAGACTCGCTGAAAGCGCTAGCGTTGTGGATGAAGCGCAATGCCAAAAATGAAGCCGTAGAGACTGAAACGGCAGAATGAAAAAAGAGCCGCAATTG
+SRR8185316.4 ERR022075.14519231 length=100
IIIIIIIIIIIIEIHIIIGIIIIIFIIIIIFIHIIIIIIIIFHIIIIIIIEIIIIHFEGIEIIFIFIHIIHGIIIIII+IIGHHHIEICCA>AB@8B@B8
@SRR8185316.5 ERR022075.15547901 length=100
TCCGGTCGAAAAACTGCTGGCAGTGGGGCATTACCTCGAATCTACCGTCGATATTGCTGAGTCCACCCGCCGTATTGCGGCAAGCCGCATTCCGGCTGAT
+SRR8185316.5 ERR022075.15547901 length=100
HHHHHHFHHHHHHFBHHHHGHHGHHGEGDGGGDBGGFG@GGGGEDEGCGGHHHEGHFHG@EFFHDFDDCC>9BBDEEBEEEDGDDBBC:@##########
@SRR8185316.6 ERR022075.2221983 length=100
TTGCCAGTATCAGATGCAGTTCCCAGGTTGAGCCCGGGGATTTCACATCTGACTTAACAAACCGCCTGCGTGCGCTTTACGCCCAGTAATTCCGATTAAC
+SRR8185316.6 ERR022075.2221983 length=100
GHHHHHHHHHHGHHEHHHHHHHGHDGHHHHHHHHEH>HHHHHGHGHHFHHHDFHH@GGDGHFFD>GFGBFHHHGHHHDHEHHDEFGHFHBDEDGEBEGE8
@SRR8185316.7 ERR022075.1027069 length=100
ATCCCTCAGCAGTGCGCCAATTTATCATCCAGTATAACATTGCGATGCAGAAGCAGCTTGCTGCACACCCTGAGTTAGCAAACGATGAAGTTGCTCTGCA
+SRR8185316.7 ERR022075.1027069 length=100
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIGIIIIIIHIIHIIIIEHGHIHHGHGIGDIEGGGHEIH@3DEDF<BEBFBD
```

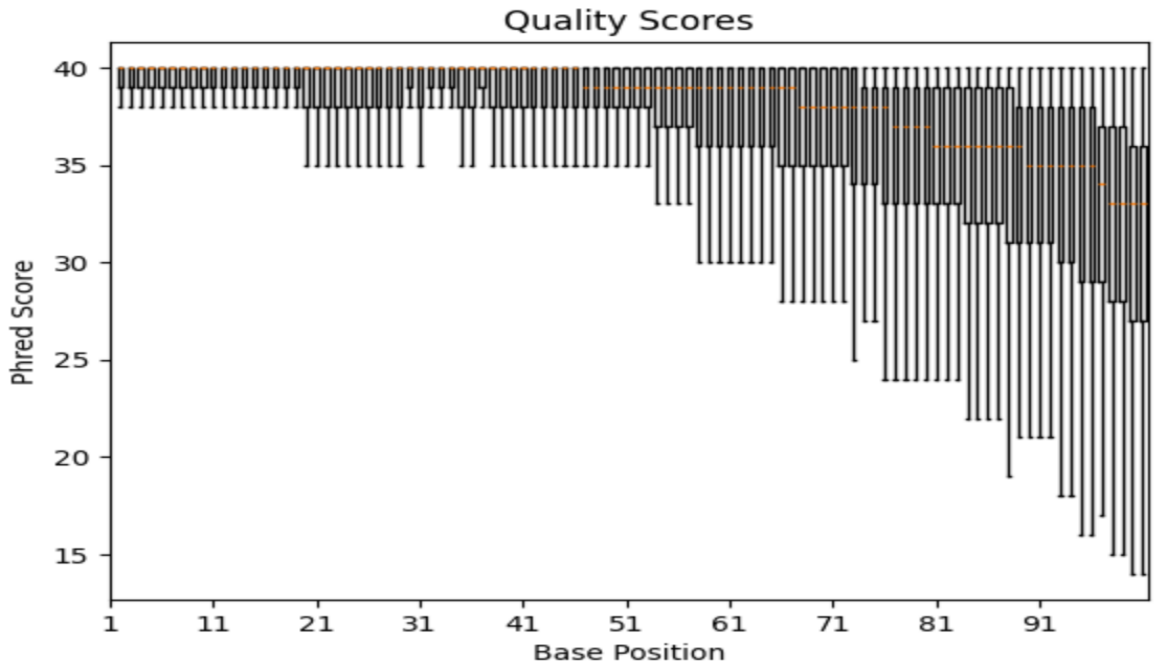
V. Plot the quality of the reads in the fastq file using a box plot.

For each sequence, I converted its ASCII-encoded quality string into Phred scores, based on Module 4: Lecture 2. Sequence Mapping I used the Sanger standard (Phred+33) for decoding quality scores.

I then rearranged these scores to align them by their base positions.Using this data, created a box plot to visually represent the quality scores at each base position.

```python
import matplotlib.pyplot as plt
filename = "SRR8185316.fastq"
# Store the quality scores.
all_qualities = []
# Read the first 100,000 reads from the file
with open(filename, 'r') as file:
    for _ in range(100000):
        # Skip the non-quality lines.
        file.readline()
        file.readline()
        file.readline()
        # Read the quality line for the current read.
        quality = file.readline().strip()
        # Convert ASCII characters in the quality string to Phred scores (Phred+33 encoding) and store them.
        phred_scores = [ord(char) - 33 for char in quality]
        all_qualities.append(phred_scores)
# Rearrange the list of quality scores for having one list for each base position
# Transpose scores for plotting.
transposed_qualities = list(map(list, zip(*all_qualities)))
# Create a box plot of the quality scores.
plt.boxplot(transposed_qualities, showfliers=False)
plt.title('Quality Scores')
plt.xlabel('Base Position')
plt.ylabel('Phred Score')
# Set the x-axis labels to show specific intervals
plt.xticks(ticks=range(0, len(transposed_qualities), 10), labels=range(1, len(transposed_qualities) + 1, 10))
# Display the plot.
plt.show()
```

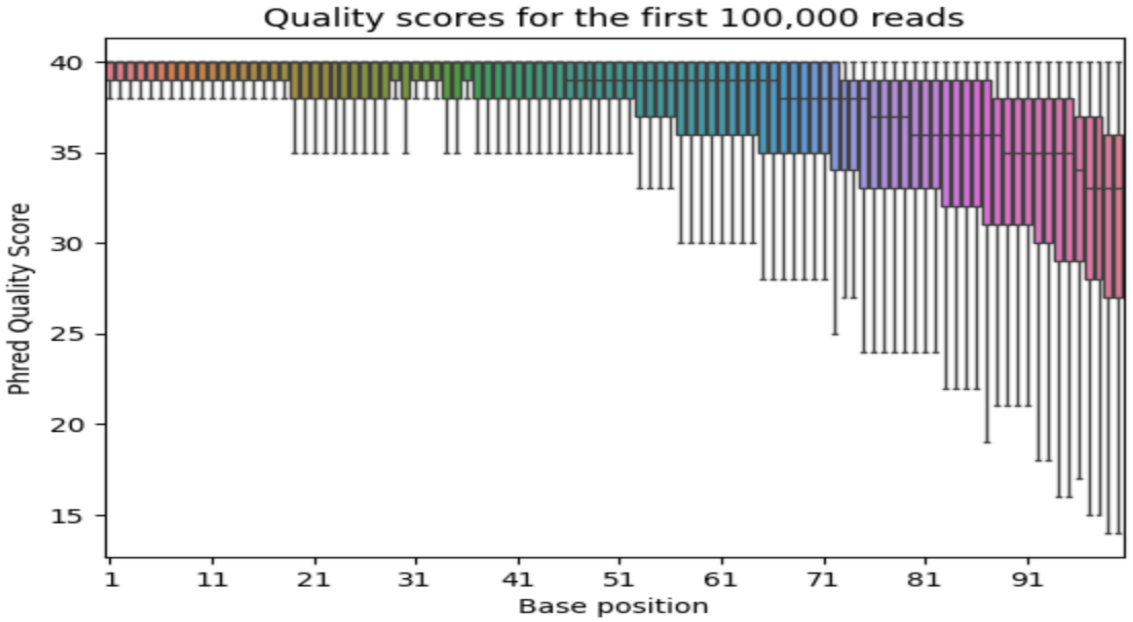
Box plot:



We can use Seaborn to add colour to the box plot, making it simpler to spot trends and differences between base positions:

```
sns.boxplot(data=transposed_qualities, showfliers=False)
plt.title('Quality scores for the first 100,000 reads')
plt.xlabel('Base position')
plt.ylabel('Phred Quality Score')
plt.xticks(ticks=range(0, len(transposed_qualities), 10), labels=range(1, len(transposed_qualities) + 1, 10))

plt.show()
```

Box Plot:



Results:

By looking at the median lines (dark grey in colourful plot and orange in black-white plot) we can see at lower base positions, such as 1 to 10, the Phred quality scores are very high, often at or near the maximum score of 40. This means that the initial readings are very accurate.

In the middle of the sequence, around base positions 30 to 50, the median quality scores show a slight decrease, but still having a high score, generally above 30. The readings are still good but start to get a bit less reliable.

At higher base positions, such as 60 to 100, there is a noticeable decline in the median quality scores; this means the quality starts to vary more and usually gets worse along the sequence.

The box plot shows that the quality of the sequencing is very good at first but drops off toward the end of the reads.

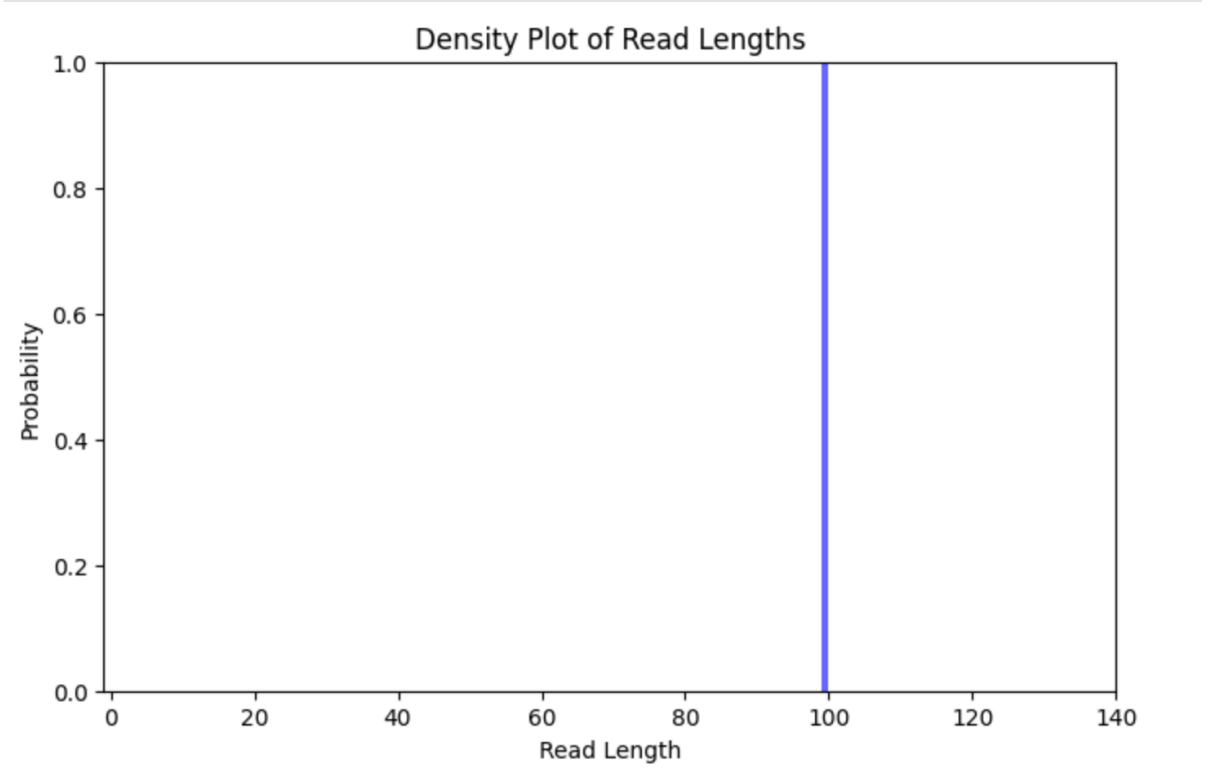
VI. Show the distribution of read lengths using a density plot.

Density plot code:

```python
import matplotlib.pyplot as plt


filename = "SRR8185316.fastq"

# a list to hold the lengths of each read
read_lengths = []

with open(filename, 'r') as file:
    while True:
        line = file.readline()
        # If the line is empty, we've reached the end of the file
        if not line:
            break

        # If the line starts with '@', it's the start of a new read
        if line.startswith('@'):

            # Reading the sequence line
            sequence = file.readline().strip()

            # Adding the length of the sequence to our list
            read_lengths.append(len(sequence))
            file.readline()  # Skip '+'
            file.readline()  # Skip quality line
```

```
plt.figure(figsize=(8, 5))
plt.hist(read_lengths, bins=range(min(read_lengths), max(read_lengths) + 1), density=True, alpha=0.6, color='b')


plt.title('Density Plot of Read Lengths')
plt.xlabel('Read Length')
plt.ylabel('Probability')

plt.xlim(min(read_lengths) - 1, max(read_lengths) + 40)
plt.ylim(0, 1)
plt.grid(True)
plt.show()
```

Density plot:



Results:
The density plot displays a single, sharp peak at a read length of 100 nucleotides. This means that all the reads in the fastq file are of uniform length and there are no other read lengths.

Part A - (Q3)

First, I installed SRA FastQC:

```
@maryam:~/Desktop/test$ sudo apt install fastqc
package lists... Done
 dependency tree... Done
state information... Done
```

Then I used command below to create an HTML report:

```
@maryam:~/Desktop/test$ fastqc SRR8185316.fastq
```

```
null
Started analysis of SRR8185316.fastq
Approx 5% complete for SRR8185316.fastq
Approx 10% complete for SRR8185316.fastq
Approx 15% complete for SRR8185316.fastq
Approx 20% complete for SRR8185316.fastq
Approx 25% complete for SRR8185316.fastq
Approx 30% complete for SRR8185316.fastq
Approx 35% complete for SRR8185316.fastq
Approx 40% complete for SRR8185316.fastq
Approx 45% complete for SRR8185316.fastq
Approx 50% complete for SRR8185316.fastq
Approx 55% complete for SRR8185316.fastq
Approx 60% complete for SRR8185316.fastq
Approx 65% complete for SRR8185316.fastq
Approx 70% complete for SRR8185316.fastq
Approx 75% complete for SRR8185316.fastq
Approx 80% complete for SRR8185316.fastq
Approx 85% complete for SRR8185316.fastq
Approx 90% complete for SRR8185316.fastq
Approx 95% complete for SRR8185316.fastq
Analysis complete for SRR8185316.fastq
```

For opening HTML, I used following command to view the results:

```
@maryam:~/Desktop/test$ xdg-open SRR8185316_fastqc.html
```

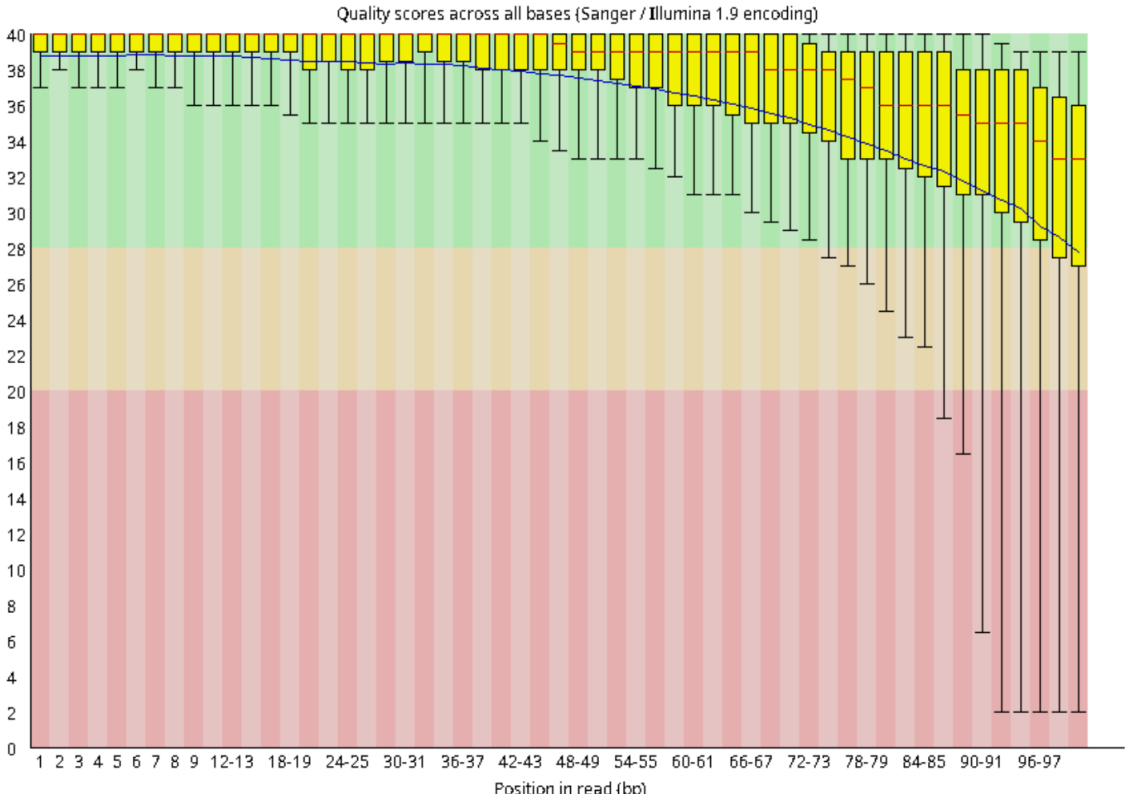
The results are as follows:

FastQC Report

Summary

- Basic Statistics
- Per base sequence quality
- Per sequence quality scores
- Per base sequence content
- Per sequence GC content
- Per base N content
- Sequence Length Distribution
- Sequence Duplication Levels
- Overrepresented sequences
- Adapter Content

Per base sequence quality



This view shows an overview of the range of quality values across all bases at each position in the FASTQ file.

The quality scores are high at the start, with most median scores (the horizontal red line inside each box) above the quality score of 35, which shows very accurate base

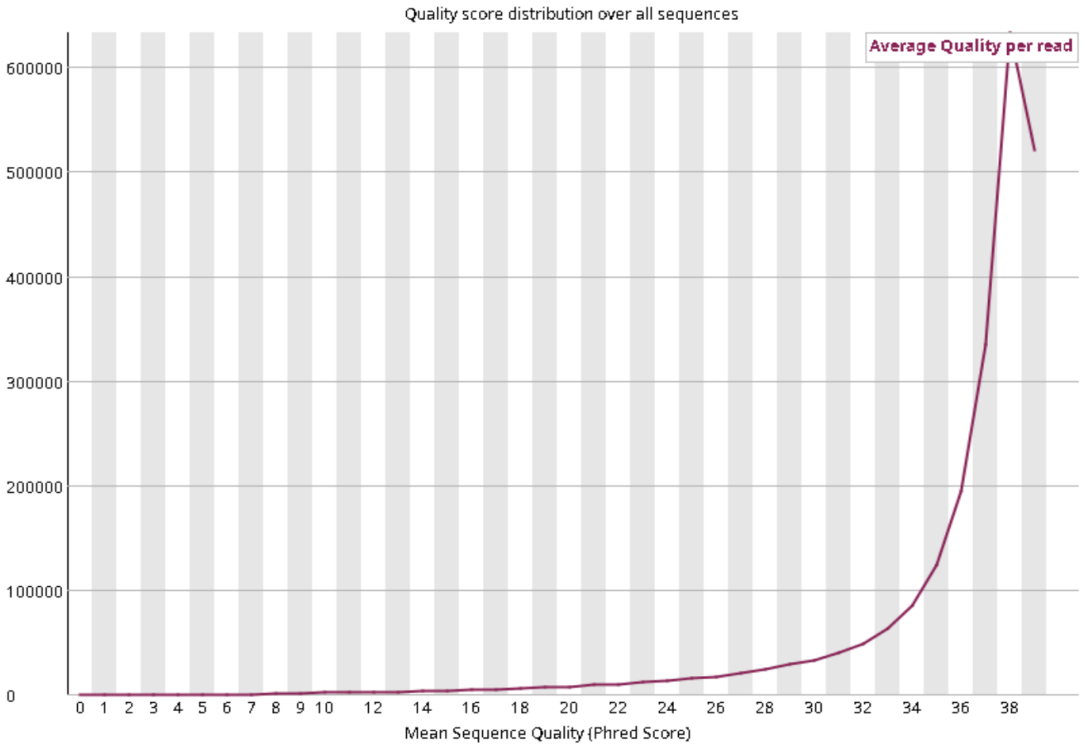
calling. This high level of quality is consistent across the initial two-thirds of the reads.
As we move right on the x-axis, there's a gradual decline in quality, with the median scores falling into the 30s and the lower quartiles reaching down into the yellow and orange zones.
Towards the end of the reads, after position 60, the quality decreases more significantly. The median scores (the red line in the box) become closer to the 30 mark, and the spread of the quality scores increases, which we can see by the expansion of the boxes.
The meaning of the colours is as follows: green shows very good quality, yellow is reasonable, orange is poor, and red is very poor.
We can see that even at the lowest, the median quality scores are in the orange zone and not red and no individual positions have scores that fall into the red zone.

So the quality of the sequencing data is very good at the start with a slight decline towards the end and the decline does not fall into low quality levels, which means the majority of the data has high quality and we have good illumina data ✅.

Basic Statistics

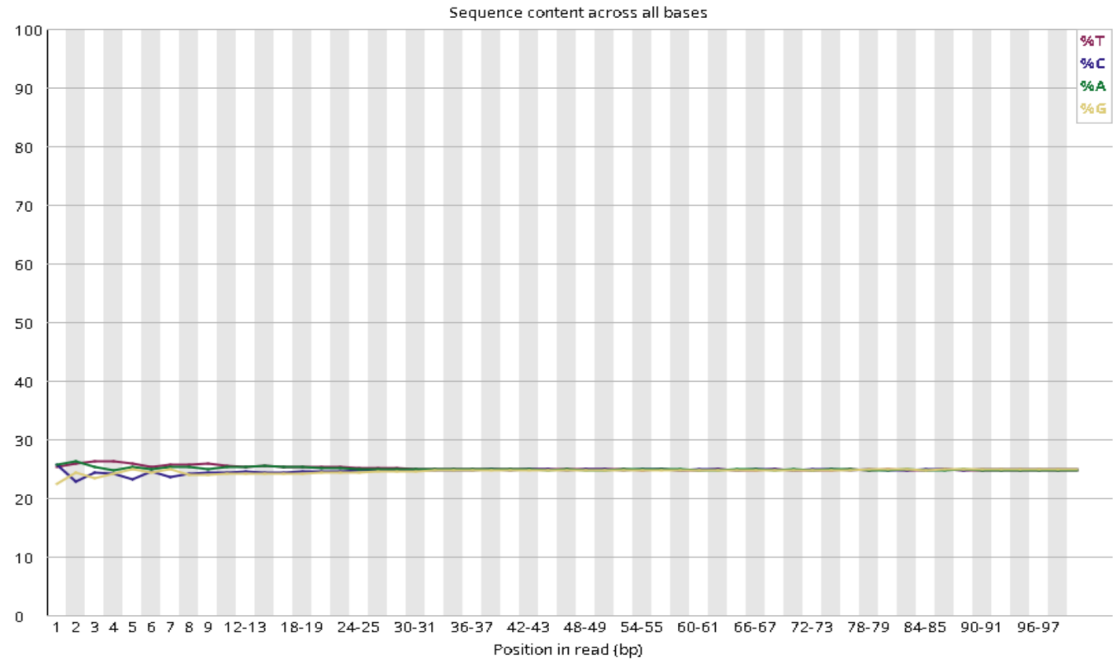
Measure	Value
Filename	SRR8185316.fastq
File type	Conventional base calls
Encoding	Sanger / Illumina 1.9
Total Sequences	2297280
Total Bases	229.7 Mbp
Sequences flagged as poor quality	0
Sequence length	100
%GC	49

Per sequence quality scores



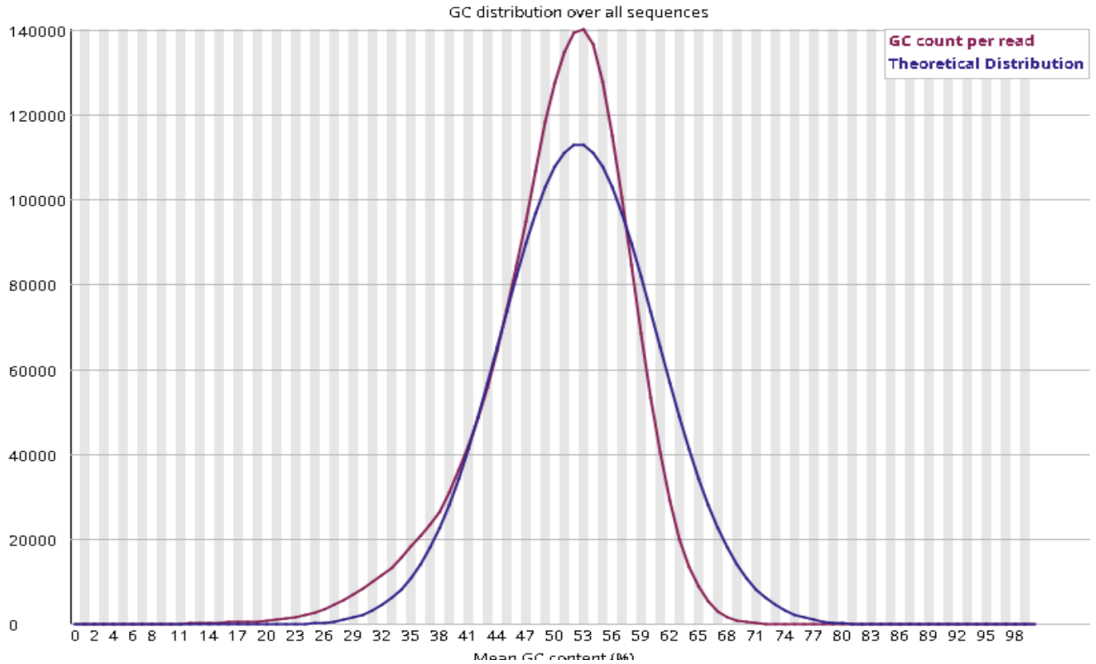
The "Per sequence quality scores" plot shows that the majority of the reads in our dataset are high quality. With most average scores close to the top end, it tells us that the sequencing resulting data is reliable.

Per base sequence content



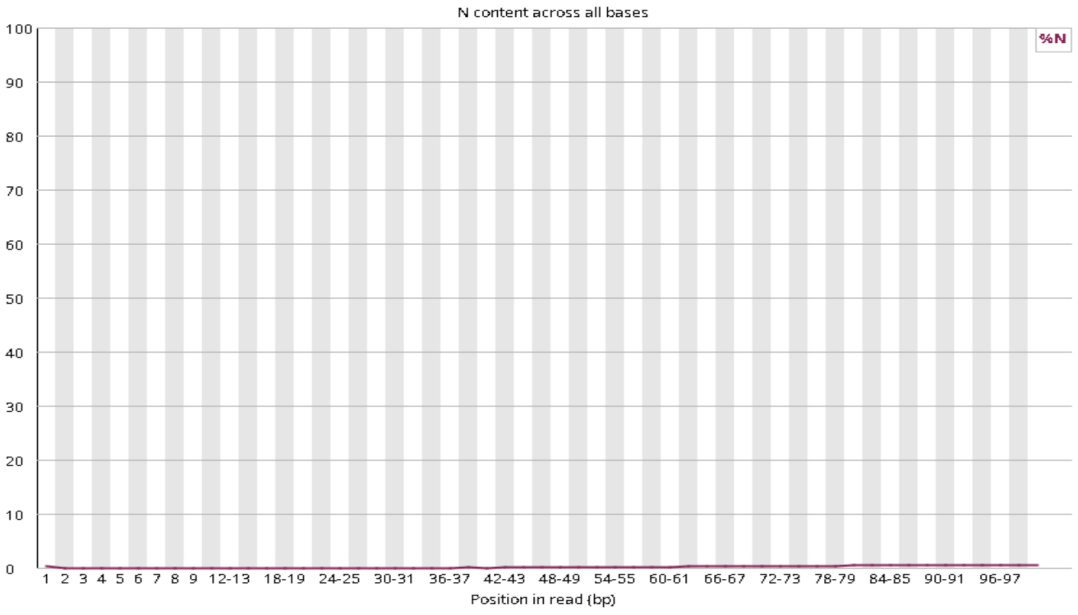
The "Per base sequence content" plot shows a balanced distribution of nucleotide bases across the read lengths, This means there's no significant bias in the composition of bases at different positions in the reads.

Per sequence GC content



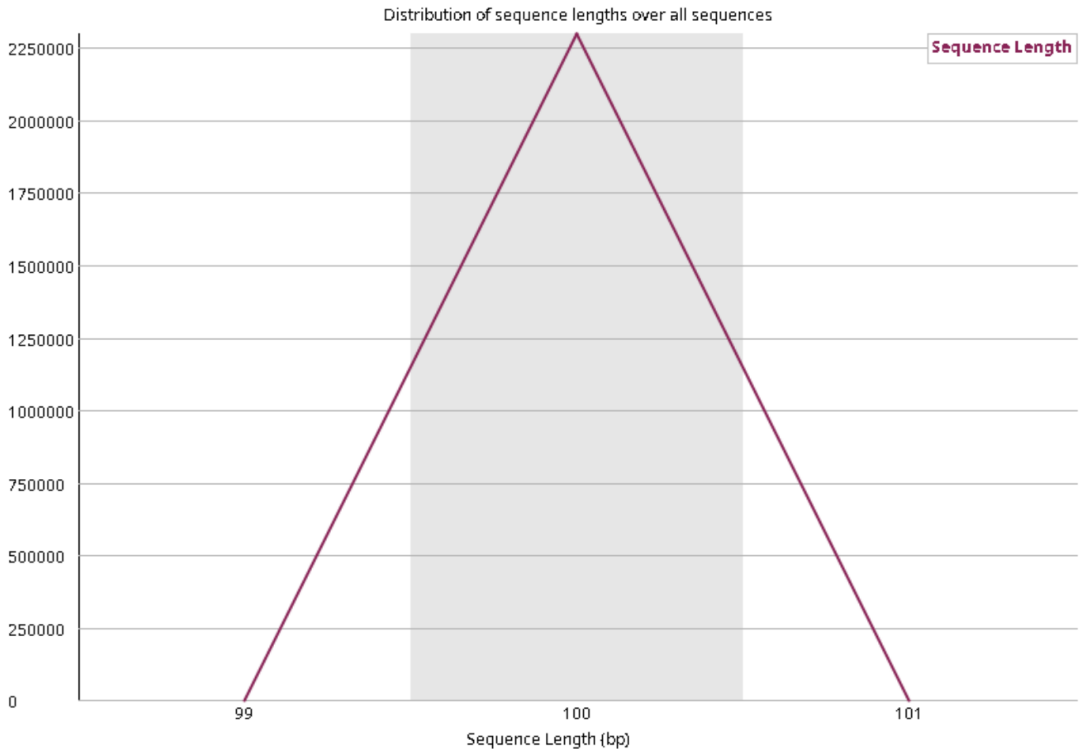
The observed values represented by the purple line and the expected distribution by the red line. The close alignment of the two shows that the GC content in the sequences closely matches what's expected, this means we have good quality data.

Per base N content



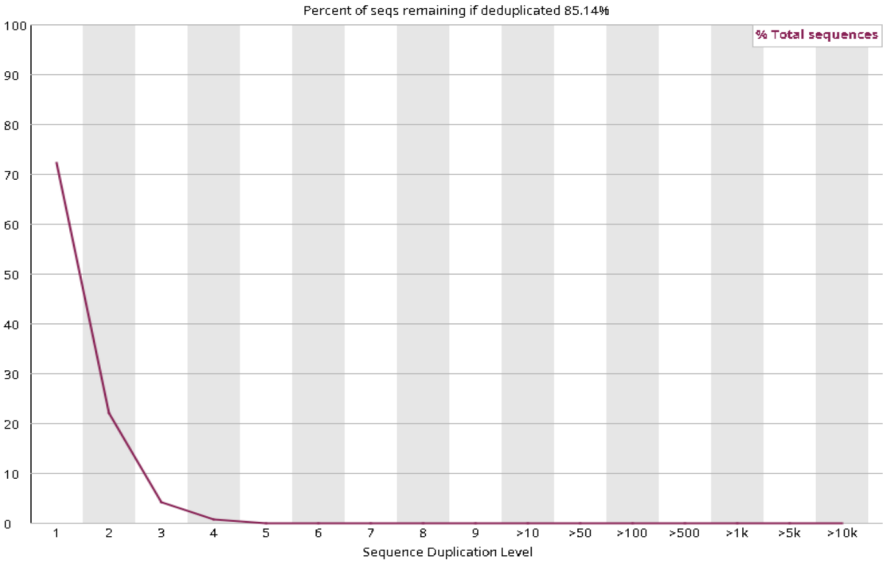
The plot shows that the sequences have almost no uncertain bases, which is good because it means the sequencing data is very clear and reliable.

Sequence Length Distribution



The plot shows that all the sequences are 100 bases long(as we saw in part VI too), indicating a uniform read length across the dataset, which is good for analysis.

Sequence Duplication Levels



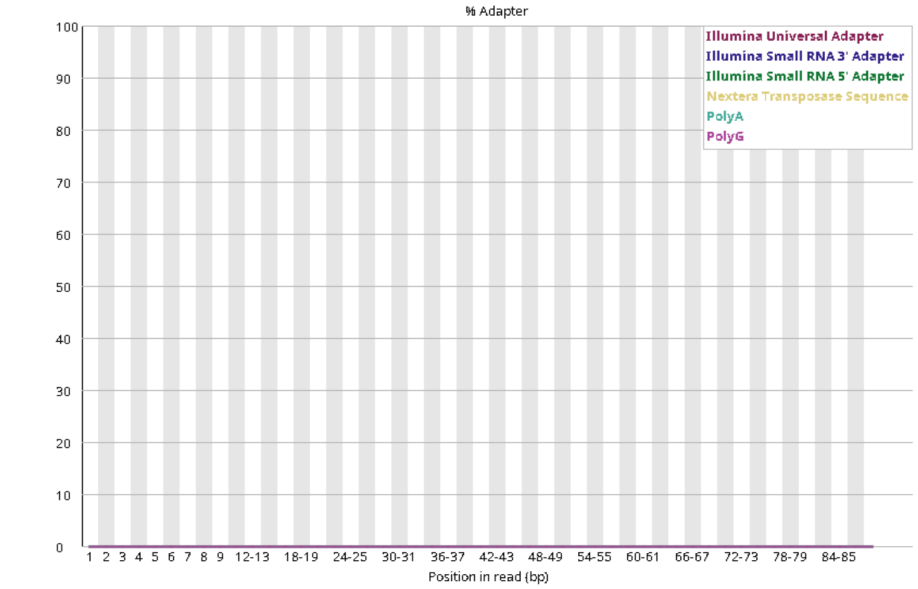
This plot shows that most of the sequences in the dataset appear only once, which is good because it means there is a lot of variety in the data and not many repeated sequences.

Overrepresented sequences

No overrepresented sequences

This means there are no sequences that show up more often than usual,

Adapter Content



The plot shows the sequencing data is clean and free from unwanted adapter sequences.

Part B - (Q1)

First, I installed spades:

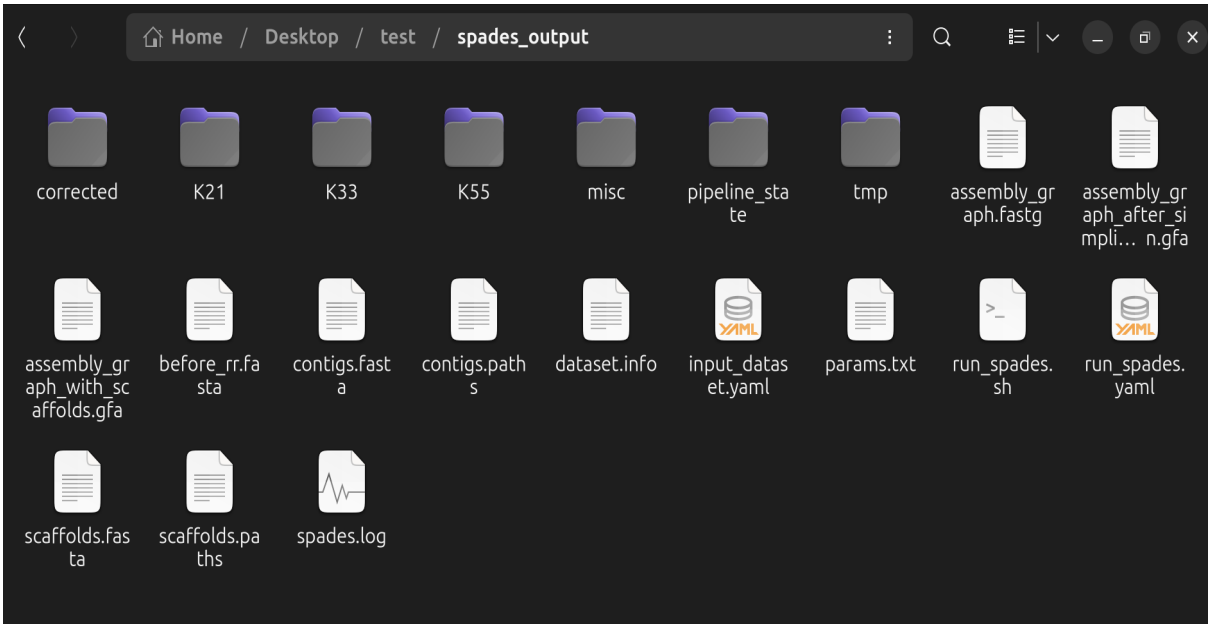
```
@maryam:~/Desktop/test$ sudo apt-get install spades
```

Then I used following command to apply genome assembly process to my fastq file to get the output (spades_output which is a draft genome assemblies from short reads):

```
@maryam:~/Desktop/test$ spades.py -s SRR8185316.fastq -o spades_output
```

```
===== Terminate finished.

 * Corrected reads are in /home/ghadyani/Desktop/test/spades_output/corrected/
 * Assembled contigs are in /home/ghadyani/Desktop/test/spades_output/contigs.fa
sta
 * Assembled scaffolds are in /home/ghadyani/Desktop/test/spades_output/scaffold
s.fasta
 * Paths in the assembly graph corresponding to the contigs are in /home/ghadyan
i/Desktop/test/spades_output/contigs.paths
 * Paths in the assembly graph corresponding to the scaffolds are in /home/ghady
ani/Desktop/test/spades_output/scaffolds.paths
 * Assembly graph is in /home/ghadyani/Desktop/test/spades_output/assembly_graph
.fastg
 * Assembly graph in GFA format is in /home/ghadyani/Desktop/test/spades_output/
assembly_graph_with_scaffolds.gfa

======= SPAdes pipeline finished.
```

After finishing process we can see list of key output files from genome assembly analysis in our spades_output file:



Part B - (Q2)

First, I installed Quast by following steps
downloading the quast-5.2.0.tar.gz file:

```
@maryam:~/Desktop/test$ wget https://github.com/ablab/quast/releases/dow
nload/quast_5.2.0/quast-5.2.0.tar.gz
```

Then, I unpack the downloaded file:

```
@maryam:~/Desktop/test$ tar -xzf quast-5.2.0.tar.gz
```

Then I downloaded the reference genome from ncbi website and save it as sequence.fasta file.

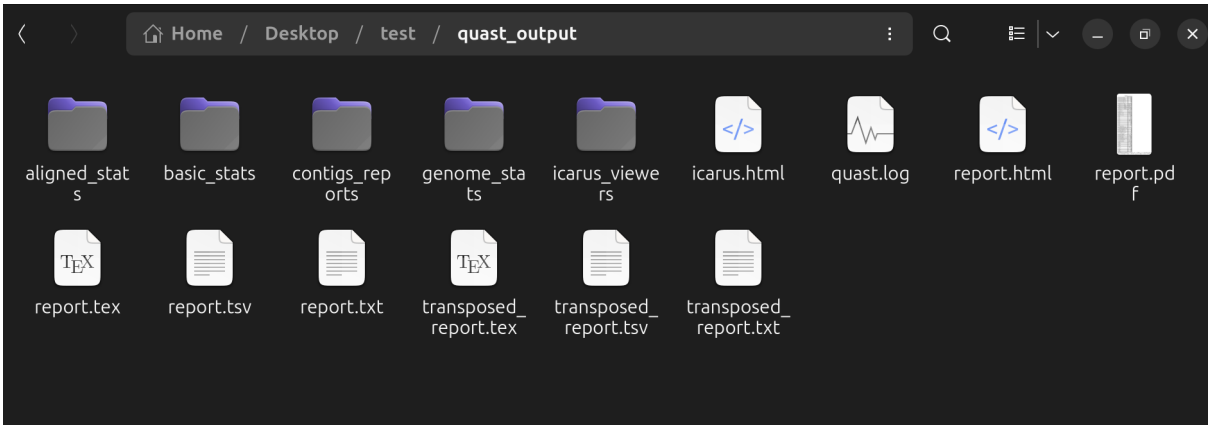
The following command runs QUAST, quast_output is the output file, sequence.fasta is the reference genome file, contigs.fasta provides the assembled contigs file for analysis:

```
@maryam:~/Desktop/test/quast-5.2.0$ ./quast.py -o ~/Desktop/test/quast_o
utput -r ~/Downloads/sequence.fasta ~/Desktop/test/spades_output/contigs.fasta
```

The process finished:

```
Finished: 2023-12-23 14:47:33
Elapsed time: 0:00:21.700487
NOTICEs: 5; WARNINGs: 0; non-fatal ERRORs: 0

Thank you for using QUAST!
```

After finishing process we can see list of key output files from evaluating the quality of a draft genome assembly using Quast in our quast_output file:

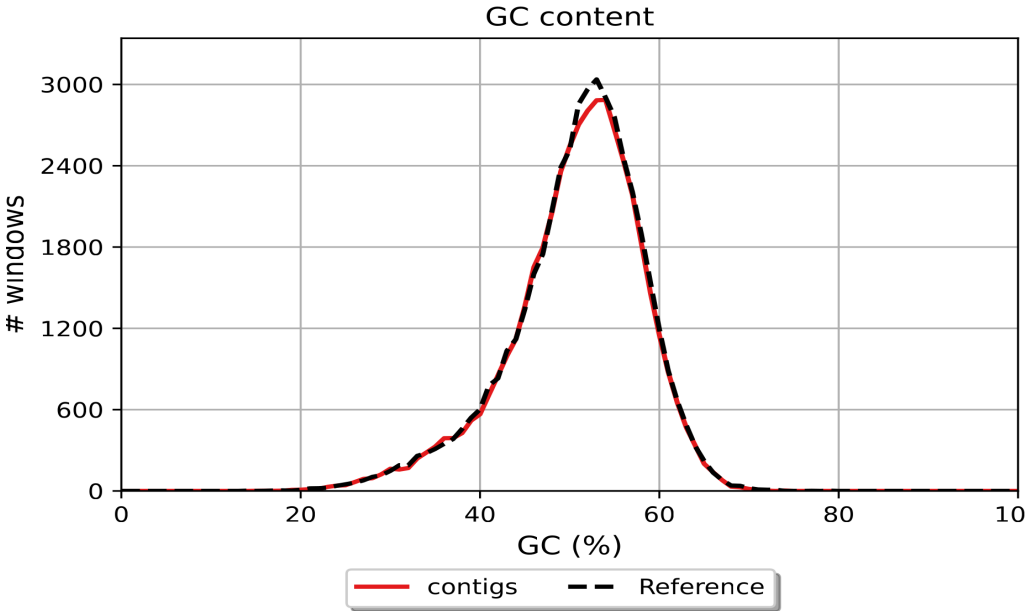


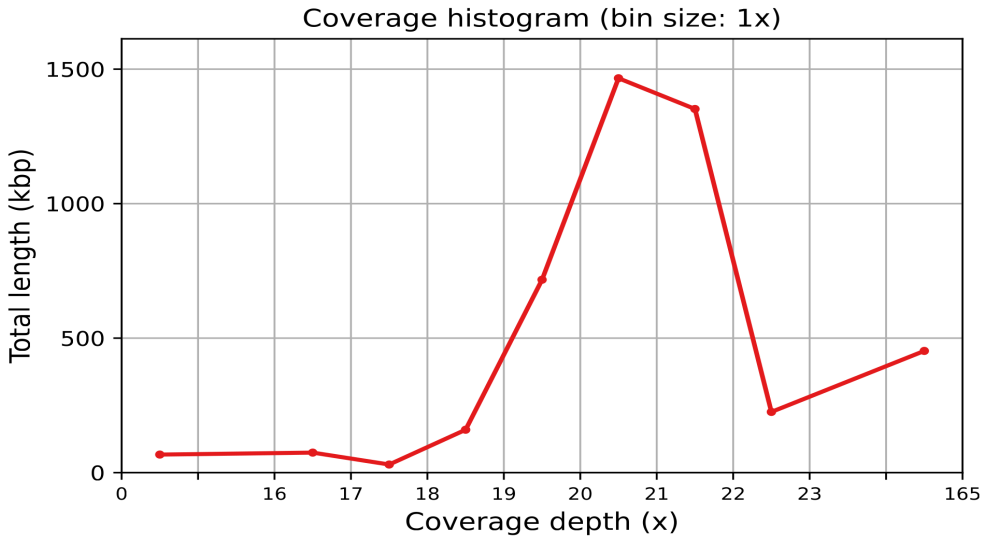
Here are some screenshots from report.pdf but the complete report has been attached to uploaded project file

Report	contigs
# contigs (>= 0 bp)	205
# contigs (>= 1000 bp)	118
# contigs (>= 5000 bp)	83
# contigs (>= 10000 bp)	75
# contigs (>= 25000 bp)	58
# contigs (>= 50000 bp)	37
Total length (>= 0 bp)	4559356
Total length (>= 1000 bp)	4540236
Total length (>= 5000 bp)	4462702
Total length (>= 10000 bp)	4406717
Total length (>= 25000 bp)	4140684
Total length (>= 50000 bp)	3413284
# contigs	124
Largest contig	221601
Total length	4544603
Reference length	4639675
GC (%)	50.72
Reference GC (%)	50.79
N50	71324
NG50	69051
N90	26205
NG90	21591
auN	94028.9

Misassemblies report

	contigs
# misassemblies	1
# contig misassemblies	1
# c. relocations	1
# c. translocations	0
# c. inversions	0
# scaffold misassemblies	0
# s. relocations	0
# s. translocations	0
# s. inversions	0
# misassembled contigs	1
Misassembled contigs length	182275
# local misassemblies	0
# scaffold gap ext. mis.	0
# scaffold gap loc. mis.	0
# unaligned mis. contigs	0
# mismatches	61
# indels	10
# indels (<= 5 bp)	4
# indels (> 5 bp)	6
Indels length	1059





Part C - (Q1)

First I installed bwa:

```
@maryam:~/Desktop/test$ sudo apt install bwa
```

Then I index reference genome with BWA, to allow BWA to perform the alignment more efficiently:

```
@maryam:~/Downloads$ bwa index sequence.fasta
[bwa_index] Pack FASTA... 0.13 sec
[bwa_index] Construct BWT for the packed sequence...
[bwa_index] 3.86 seconds elapse.
[bwa_index] Update BWT... 0.09 sec
[bwa_index] Pack forward-only FASTA... 0.07 sec
[bwa_index] Construct SA from BWT and Occ... 1.88 sec
[main] Version: 0.7.17-r1188
[main] CMD: bwa index sequence.fasta
[main] Real time: 6.118 sec; CPU: 6.042 sec
```

Then I aligned short reads file to the reference genome:

```
@maryam:~/Desktop/test$ bwa mem sequence.fasta SRR8185316.fastq > SRR8185316_aligned.sam
```

After finishing process out sam file has been created:

Part C - (Q2)

Print the head of the obtained SAM file

Using following command we got head of our sam file:

```
@maryam:~/Desktop/test$ head SRR8185316_aligned.sam
```

This command gave the first few lines of the SAM file which include the SAM header and the first few alignments:

```
@SQ     SN:U00096.2     LN:4639675
@PG     ID:bwa  PN:bwa  VN:0.7.17-r1188 CL:bwa mem sequence.fasta SRR8185316.fastq
SRR8185316.1    0       U00096.2        3783615 60      100M    *       0       0       AGCGGTACACATTATGGGTCTGCTCTCCGC
AGGCGGCGTACACAGCCACGAAGATCACATCATGGCGATGGTAGAACTGGCAGCTGAACGCGGCGCAGAA  IIGIIIIIIIIIIIIHIIIIIIIIIIIIIIIII@IHHEHIIIIIII
IIIIHIIHIIIIIIIIGIHIIHFIIEIIGIHHGH@BE3BB>@>>2?@8?>?A@1  NM:i:0  MD:Z:100        AS:i:100        XS:i:0
SRR8185316.2    0       U00096.2        1394282 0       100M    *       0       0       TAATGGCCGGCGACCTTATCCGCTGGAAAC
CATGCTACGCATTCACTGCATGCAGCATTGGTACAACCTGAGCGATGGCGCGATGGAAGATGCTCTGTAC  HHHHHHHHHHDHHDHHHGFGHDGGHG>GGGG<GDGHHEGH@EGGGH
BHBHHHGHHFHHHHDEDFFHBHEFHHHH<FHDBH@HCBCG3DDB+?:=8DDBBD  NM:i:0  MD:Z:100        AS:i:100        XS:i:100
SRR8185316.3    16      U00096.2        3679372 60      100M    *       0       0       ATTTCAGCGGATCGGCAGGATGGGGACCTA
ACAAGGTTGACCCTTTCAGACGATAGCGCCGCCAGCCCTCTTTAGTATCGGCAGGCCAGGTTGCCACCAT  ########################################
##############?/;<?8=B<2A>A=A2CCACEGEDE@GEGBDDFH>HHDHH=  NM:i:4  MD:Z:0G20C1T36A39       AS:i:84 XS:i:0
SRR8185316.4    16      U00096.2        2416639 60      100M    *       0       0       CAATTGCGGCTCTTTTTTCATTCTGCCGTT
TCAGTCTCTACGGCTTCATTTTTGGCATTGCGCTTCATCCACAACGCTAGCGCTTTCAGCGAGTCTGGAG  8B@B8@BA>ACCIEIHHHGII+IIIIIIGHIIHIFIFIIEIGEFHI
IIIEIIIIIIIHFIIIIIIIIHIFIIIIIFIIIIIGIIIHIEIIIIIIIIIIII  NM:i:0  MD:Z:100        AS:i:100        XS:i:0
SRR8185316.5    0       U00096.2        783     60      100M    *       0       0       TCCGGTCGAAAAACTGCTGGCAGTGGGGCA
TTACCTCGAATCTACCGTCGATATTGCTGAGTCCACCCGCCGTATTGCGGCAAGCCGCATTCCGGCTGAT  HHHHHHFHHHHHHFBHHHHGHHGHHGEGDGGGDBGGFG@GGGGEDE
GCGGHHHEGHFHG@EFFHDFDDCC>9BBDEEBEEEDGDDBBC:@##########  NM:i:0  MD:Z:100        AS:i:100        XS:i:0
SRR8185316.6    16      U00096.2        4034103 0       100M    *       0       0       GTTAATCGGAATTACTGGGCGTAAAGCGCA
CGCAGGCGGTTTGTTAAGTCAGATGTGAAATCCCCGGGCTCAACCTGGGAACTGCATCTGATACTGGCAA  8EGEBEGDEDBHFHGFEDHHEHDHHHGHHHFBGFG>DFFHGDGG@H
HFDHHHFHHGHGHHHH>HEHHHHHHHHGDHGHHHHHHHEHHGHHHHHHHHHHG  NM:i:0  MD:Z:100        AS:i:100        XS:i:100
SRR8185316.7    0       U00096.2        4043105 60      100M    *       0       0       ATCCCTCAGCAGTGCGCCAATTTATCATCC
AGTATAACATTGCGATGCAGAAGCAGCTTGCTGCACACCCTGAGTTAGCAAACGATGAAGTTGCTCTGCA  IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
IIIIGIIIIIHIHIIIIIEHGHIHHGHGIGDIEGGGHEIH@3DEDF<BEBFBD  NM:i:0  MD:Z:100        AS:i:100        XS:i:0
SRR8185316.8    16      U00096.2        75459   60      100M    *       0       0       GGGGCAGACATTTTTTTAACACTTTGCACC
TCAAAAAAGAGTGGCAAAGGACTTGAGAAGGAGCCTCAAATCCCTTCGCCGGCGTTATCCGGATCAGGTT  DEDEDDGBEFFAGHHCEEDE;E>CEBEBEEB;D>DHHHHHGDDD>B
GHHHHHHHHFHHHHHHHGHGHHHHHHHHHHHGDHHHBHHHFHHHGHHHGHHHH  NM:i:0  MD:Z:100        AS:i:100        XS:i:0
```

So the header of our sam file is:

```
@SQ     SN:U00096.2     LN:4639675
@PG     ID:bwa  PN:bwa  VN:0.7.17-r1188 CL:bwa mem sequence.fasta SRR8185316.fastq
```

The header section of a SAM file contains metadata about the alignment its elements are:

1. **@SQ:** this shows that the line contains information about the reference sequence used for alignment.
2. **SN:U00096.2:** SN stands for sequence name, and `U00096.2` is the identifier for the reference sequence.

3. **LN:4639675:** LN stands for the length of the sequence, showing that the reference sequence is 4,639,675 bases long.

4. **@PG:** this line shows that the following tags contain information about the programs used to perform the alignment.

5. **ID:bwa:** ID is an identifier for the program record and bwa shows that BWA was the alignment tool used.

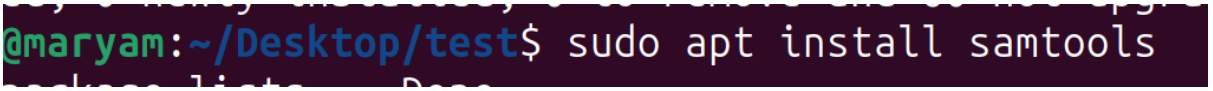
6. **PN:bwa:** PN stands for program name, again specifying BWA.

7. **VN:0.7.17-r1188:** VN is the version number of BWA that was used.

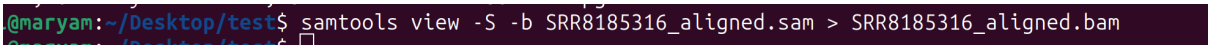
8. **CL:bwa mem sequence.fasta SRR8185316.fastq:** CL stands for command line, showing the exact command that was run. It shows that the BWA-MEM algorithm was used, with 'sequence.fasta' as the reference and 'SRR8185316.fastq' as the input read file.

Part C - (Q3)

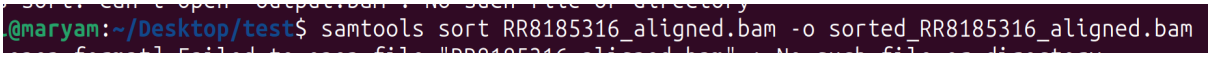
First i installed samtools:



Then I used samtools view to convert a SAM file to a binary BAM file:



Then I use samtools sort to sort the BAM file:

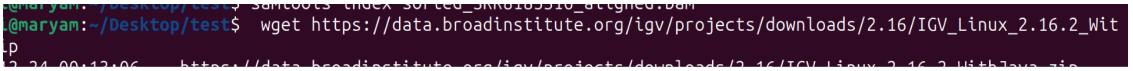


Then I used samtools index to create an index file for the sorted BAM file:

The output files (.bam, sorted, .bai, …) have been attached to the project file.

Part C - (Q4)

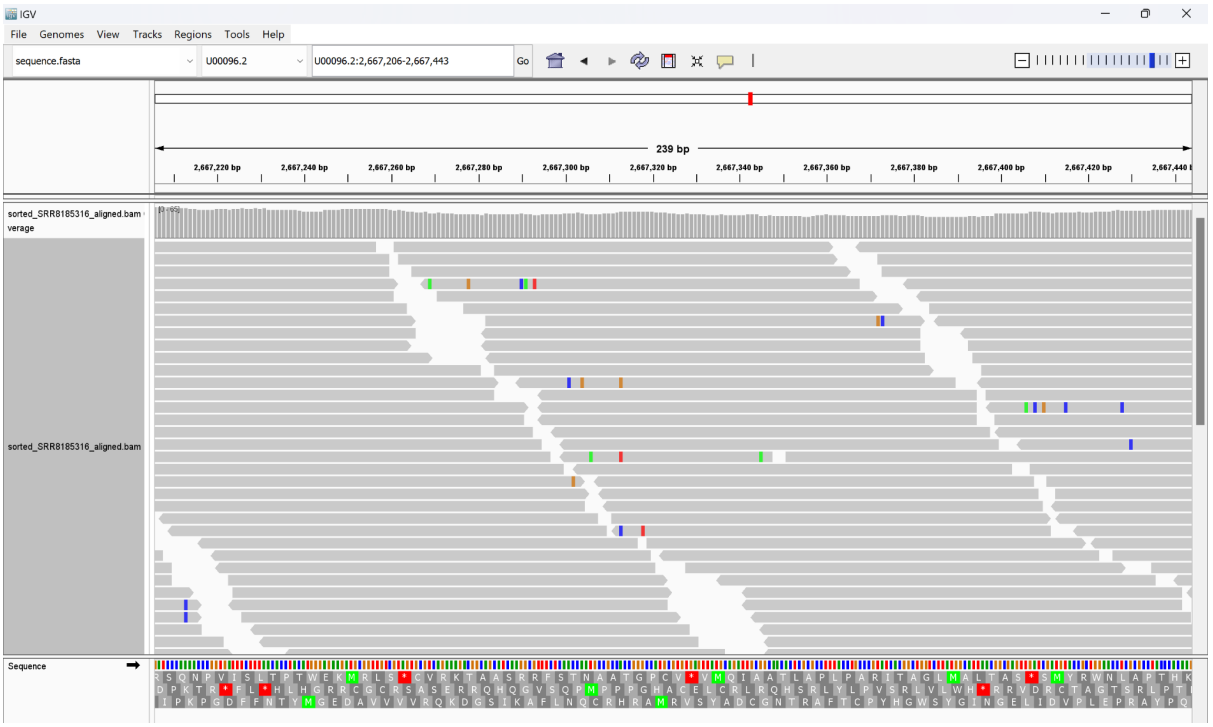
First i installed IGV:



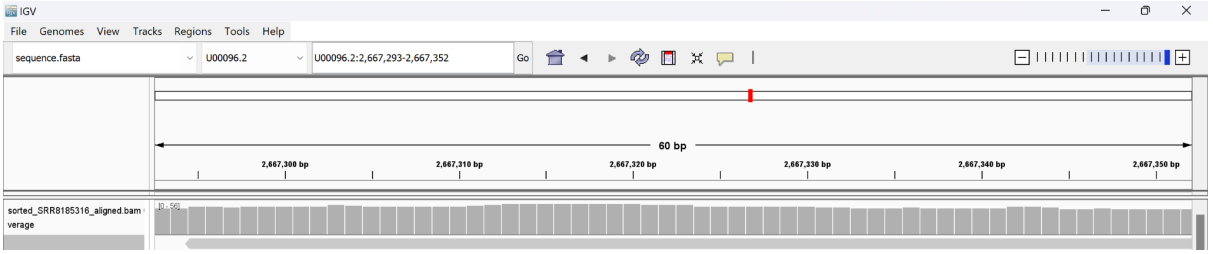
Then i ran the program:

```
@maryam:~/Desktop/test$ IGV_Linux_2.16.2/igv.sh
```

I loaded the reference genome(sequence.fasta) to genomes section and I loaded the sorted_SRR8185316_aligned.bam file and also locate sorted_SRR8185316_aligned.bam.bai in the same directory with .bam file to access the IGV for faster processing.
Here is what i got in the first look:

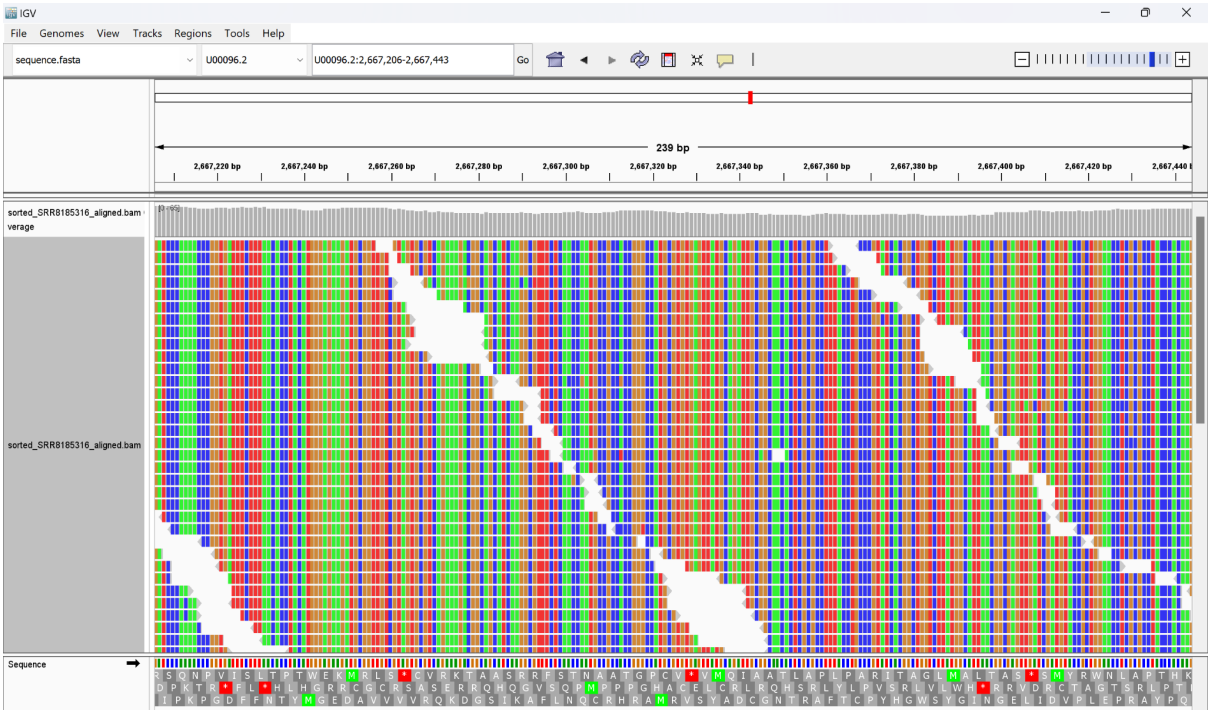


Each of these gray bars represent a read which has been aligned to a particular place in the reference genome.
 white spaces represent regions in the genome where there is no read coverage or sequencing data available. These spaces show genomic regions that may lack data or have low sequencing coverage, which can occur for various reasons such as repetitive sequences, gaps in the reference genome.
Coverage track:

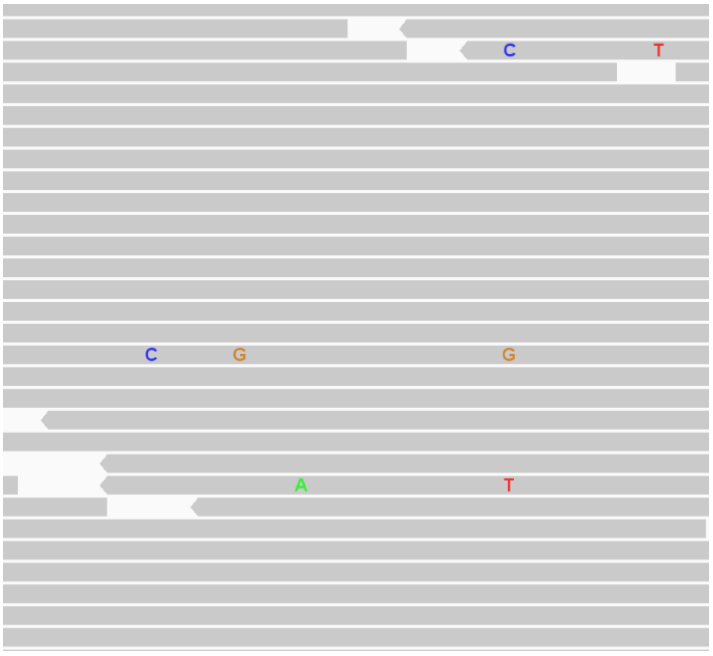


Above the reads there is a coverage track which is essentially a bar graph of the number of the reads aligned to a particular location.

We can use the option show all bases to color all bases instead of just the ones that differ from the reference.



The bases with bolder colors are higher-quality and more likely to be real mismatches



And the bases that are more faint are lower quality and less likely ro be real mismatches. (we can see this by clicking on the base to see theirs quality scores)

Part C - (Q5)

the percentage of short reads that are mapped to the reference genome:

```
@maryam:~/Desktop/test$ samtools flagstat SRR8185316_aligned.bam
2297765 + 0 in total (QC-passed reads + QC-failed reads)
2297280 + 0 primary
0 + 0 secondary
485 + 0 supplementary
0 + 0 duplicates
0 + 0 primary duplicates
2294728 + 0 mapped (99.87% : N/A)
2294243 + 0 primary mapped (99.87% : N/A)
0 + 0 paired in sequencing
0 + 0 read1
0 + 0 read2
0 + 0 properly paired (N/A : N/A)
0 + 0 with itself and mate mapped
0 + 0 singletons (N/A : N/A)
0 + 0 with mate mapped to a different chr
0 + 0 with mate mapped to a different chr (mapQ>=5)
```

We have 2297765 reads in total,
99.87 % of reads mapped to the reference genome.

Part C - (Q6)

6. read depth for the sorted BAM file at all positions of the reference genome:

```
@maryam:~/Desktop/test$ samtools depth sorted_SRR8185316_aligned.bam| awk '{sum+=$3} END {print "Mean depth:",
}'
```

The mean depth is:

```
Mean depth: 49.2245
```

We used samtools depth to calculate the read depth at all positions and then use awk to calculate the mean depth by summing up all depths and dividing by the total number of positions.

Part C - (Q7)

How do you interpret the following expressions?

1. **29S21=1X25=**:
   - 29S: Soft clipping of 29 bases on the read's start (not aligned).
   - 21=: 21 bases aligned and matched to the reference.
   - 1X: 1 base mismatched to the reference.
   - 25=: 25 bases aligned and matched to the reference.

2. **20M2I1M1D10M**:
   - 20M: 20 bases aligned and matched to the reference.
   - 2I: 2 bases inserted in the read.
   - 1M: 1 base aligned and matched to the reference.
   - 1D: 1 base deleted from the read.
   - 10M: 10 bases aligned and matched to the reference.

3. **5M10N25M**:
   - 5M: 5 bases aligned and matched to the reference.
   - 10N: 10 bases skipped in the read (a gap or intron).
   - 25M: 25 bases aligned and matched to the reference.