Project 1- Read mapping and genome assembly

Algorithms in Bioinformatics

Mahboobeh (Mariya) Golchinpour leili

Part A - Downloading and Analyzing and Quality Control of E. Coli WGS Data First, download the data according to the following steps and then perform the relevant analyzes according to the obtained data.

1. **Download SRR8185316 (short-read WGS of E.coli) and SRR10538956 (long-read WGS of E.coli) from SRA using the SRA Toolkit in Linux or Windows. (Hint: you may use fastq-dump [options] <accession>).**

There are two sequencing approaches:

- **Illumina**
- **PacBio**

the main difference between them is that:

-**Illumina specializes in short-read raw sequence data** while

-**PacBio focuses on long-read raw sequence data**.

SRR8185316 (short-read WGS of E.coli) with accession number SRX5005276 has the following characteristics:

- The NCBI **Sequence Read Archive (SRA)** record has **accession number SRX5005276** represents a whole genome sequencing(WGS) project of E. coli with a **coverage of 50x.**
- The **sequencing** was performed using an **Illumina Genome Analyzer IIx platform.**
- single run was conducted, which generated 2.3 million sequencing spots and **229.7 million bases of DNA sequence.**
- The sequencing was conducted at a **coverage of 50x**, meaning that each base in the genome was sequenced, on average, 50 times.
- **reads were single-end**, meaning that only one end of each DNA fragment was sequenced.

SRR10538956 (long-read WGS of E.coli) with accession number SRX5005276 has the following characteristics:

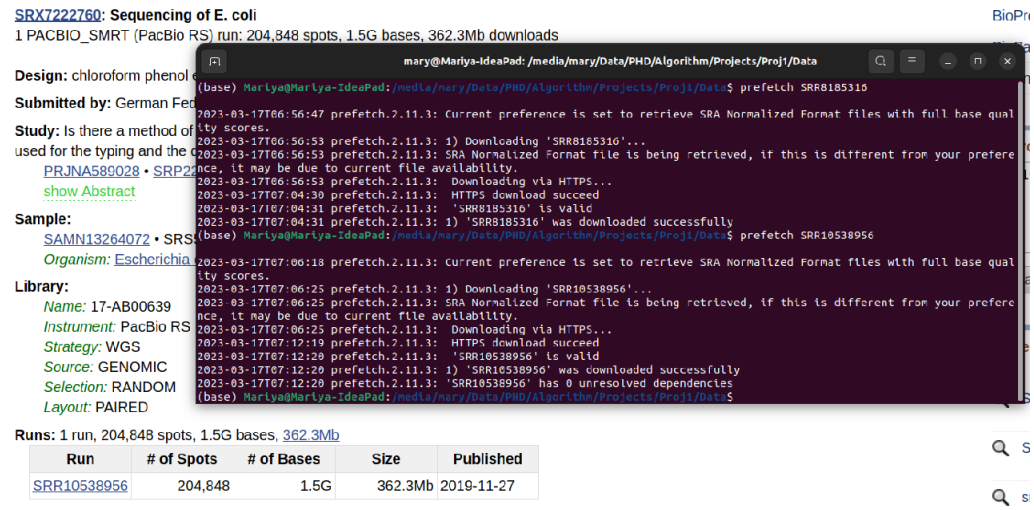
- The NCBI Sequence Read Archive (SRA) record with accession number SRX7222760 represents a sequencing project of E. coli.
- The sequencing was performed using a **PacBio RS platform.**

- A single run was conducted, which generated 204,848 sequencing spots and 1.5 billion bases of DNA sequence.
- The library was prepared using genomic DNA as the source material.
- **random fragments of DNA** were selected for sequencing.
- the reads were **paired-end**, meaning that **both ends of each DNA fragment were sequenced.**

Download sra file from ncbi by:

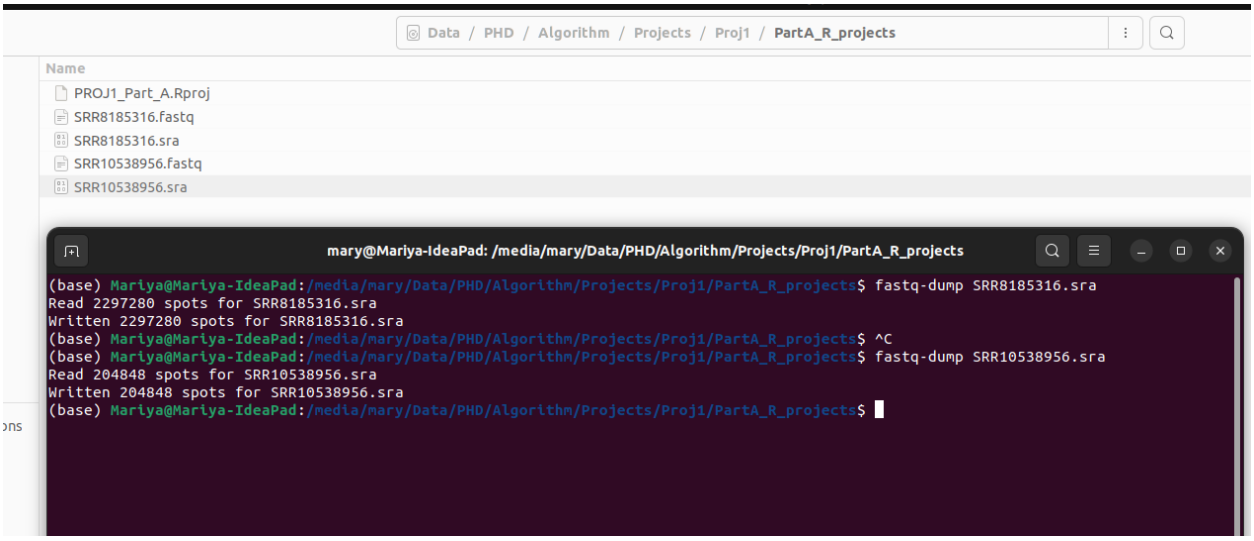
```
prefetch SRR8185316.sra

prefetch SRR10538956.sra
```



Convert .sra file to .fastq

```
fastq-dump SRR8185316.sra
```



2. In each file, find the strain of E.coli and the type of reads (single-end or paired-end) (refer to NCBI or EBI ENA (European Nucleotide Archive)). Explain what is the difference between paired-end and interleaved paired-end files.

`SRR8185316`

strain: K-12 substr. MG1655

`Layout :` single-end

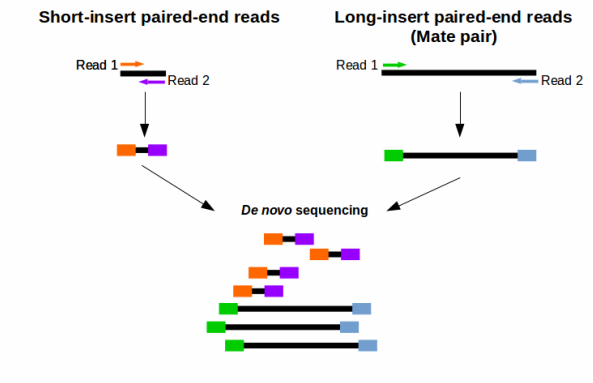
single-end, meaning that only one end of each DNA fragment was sequenced.

`SRR10538956`

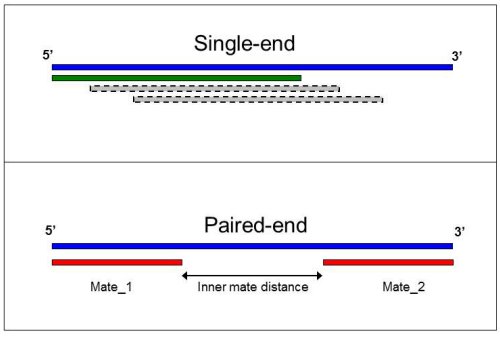
strain: 17-AB00639

`Layout:` paired-end

paired-end, meaning that both ends of each DNA fragment were sequenced.



https://www.ecseq.com/support/ngs/what-is-mate-pair-sequencing-useful-for



https://www.researchgate.net/publication/305320151_Molecular_systematics_phylogenetics_and_evolution_of_Phyllostomid_bats_Mammalia_Chiroptera_a throughput_sequencing_technologies

Difference between paired-end and interleaved paired-end files:

Paired-end sequencing **generates two separate reads from opposite ends of a DNA fragmen**t, which are typically **stored in separate files**. Paired-end format stores the forward and reverse reads in separate files, while **interleaved paired-end format stores both reads in a single file**, with each read pair separated by a fixed number of lines. The main **advantage of interleaved paired-end files** is that they are **more compact and easier to work** with than standard paired-end files, as they require only a single file to store all the reads.

3. Answer the following questions about the fastq file for short read data only (use existing R packages such as ShortRead to answer these questions):

`SRR8185316 (short-read WGS of E.coli) is short read file.`

I. How many reads are in the fastq file?

```
####### load library #########
library(ShortRead)
library(ggplot2)
library(tidyr)
####### read data ############
fastq= readFastq('SRR8185316.fastq')# SRR8185316 (short-read WGS of E.coli)
print(fastq)
```

```
class: ShortReadQ
length: 2297280 reads; width: 100 cycles
```

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

```
#sread- look at dna sequence
reads=sread(fastq)

id(fastq)[1] #first read identifier
quality(fastq)[1] #first read quality
reads[1] #first read sequence
sread(fastq)[1] #first read sequence
```

```
> id(fastq)[1] #first read identifier
BStringSet object of length 1:
    width seq
[1]    42 SRR8185316.1 ERR022075.10741970 length=100
> quality(fastq)[1] #first read quality
class: FastqQuality
quality:
BStringSet object of length 1:
    width seq
[1]   100 IIGIIIIIIIIIIIHIIIIIIIIIIIIIIIII@IHHEHIIIIIIIIIIIHIIHIIIIIIIIGIHIIHFIIEIIGIHHGH@BE3BB>@>>2?@8?>?A@1
> reads[1] #first read sequence
DNAStringSet object of length 1:
    width seq
[1]   100 AGCGGTACACATTATGGGTCTGCTCTCCGCAGGCGGCGTACACAGCCACGAAGATCACATCATGGCGATGGTAGAACTGGCAGCTGAACGCGGCGCAGAA
```

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

```
subseq_rep<- fastq[grepl("TTAAATGGAA", sread(fastq))]
print("Numbr of times TTAAATGGAA subsequence appear in fastq file?")
print(subseq_rep)
```

```
class: ShortReadQ
length: 179 reads; width: 100 cycles
```

```
without_subseq_rep<- fastq[!grepl("TTAAATGGAA", sread(fastq))]
print("Numbr of times TTAAATGGAA subsequence Dont appear in fastq file?")
print(without_subseq_rep)
```

class: ShortReadQ
length: 2297101 reads; width: 100 cycles

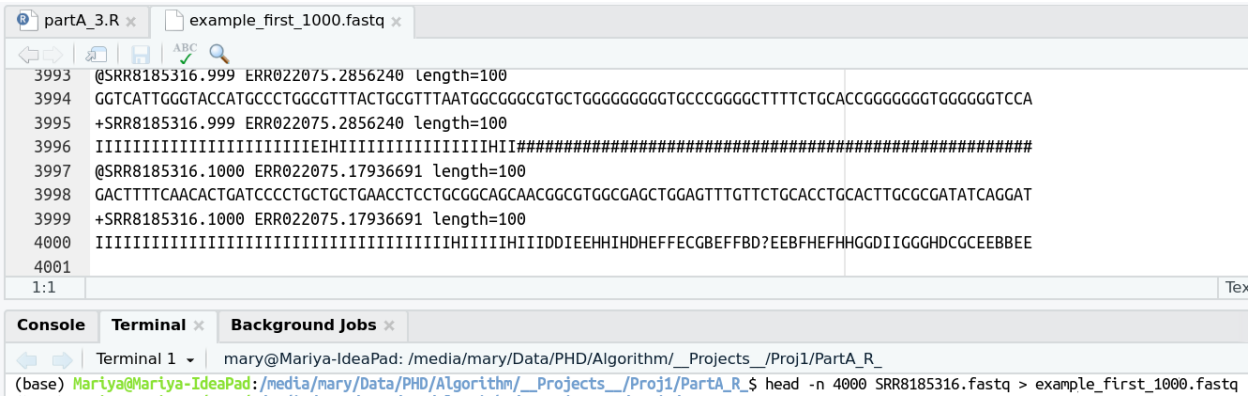
IV. Extract the first 1000 sequences of the fastq files (4000 lines).

Terminal:

```
head -n 4000 SRR8185316.fastq > example_first_1000.fastq
```

This **terminal** command will extract the first 4000 lines of the `SRR8185316.fastq` file and save them to a new file called `example_first_1000.fastq.`

The `example_first_1000.fastq` contain only the first 1000 sequences from the `SRR8185316.fastq` file.



R :

```
file <- tempfile()
writeFastq(fastq, file)
readLines(file,8) #print sample
```

```
> readLines(file,8)
[1] "@SRR8185316.1 ERR022075.10741970 length=100"
[2] "AGCGGTACACATTATGGGTCTGCTCTCCGCAGGCGGCGTACACAGCCACGAAGATCACATCATGGCGATGGTAGAACTGGCAGCTGAACGCGGCGCAGAA"
[3] "+"
[4] "IIGIIIIIIIIIIIIHIIIIIIIIIIIIIIIIII@IHHEHIIIIIIIIIIIHIIHIIIIIIIIGIHIIHFIIEIIGIHHGH@BE3BB>@>>2?@8?>?A@1"
[5] "@SRR8185316.2 ERR022075.13329732 length=100"
[6] "TAATGGCCGGCGACCTTATCCGCTGGAAACCATGCTACGCATTCACTGCATGCAGCATTGGTACAACCTGAGCGATGGCGCGATGGAAGATGCTCTGTAC"
[7] "+"
[8] "HHHHHHHHHHDHHDHHHGFGHDGGHG>GGGG<GDGHHEGH@EGGGHBHBHHHGHHFHHHHDEDFFHBHEFHHHH<FHDBH@HCBCG3DDB+?:=8DDBBD"
>
```

4000 lines

.....

```
readLines(file,4000) #4000 line
```

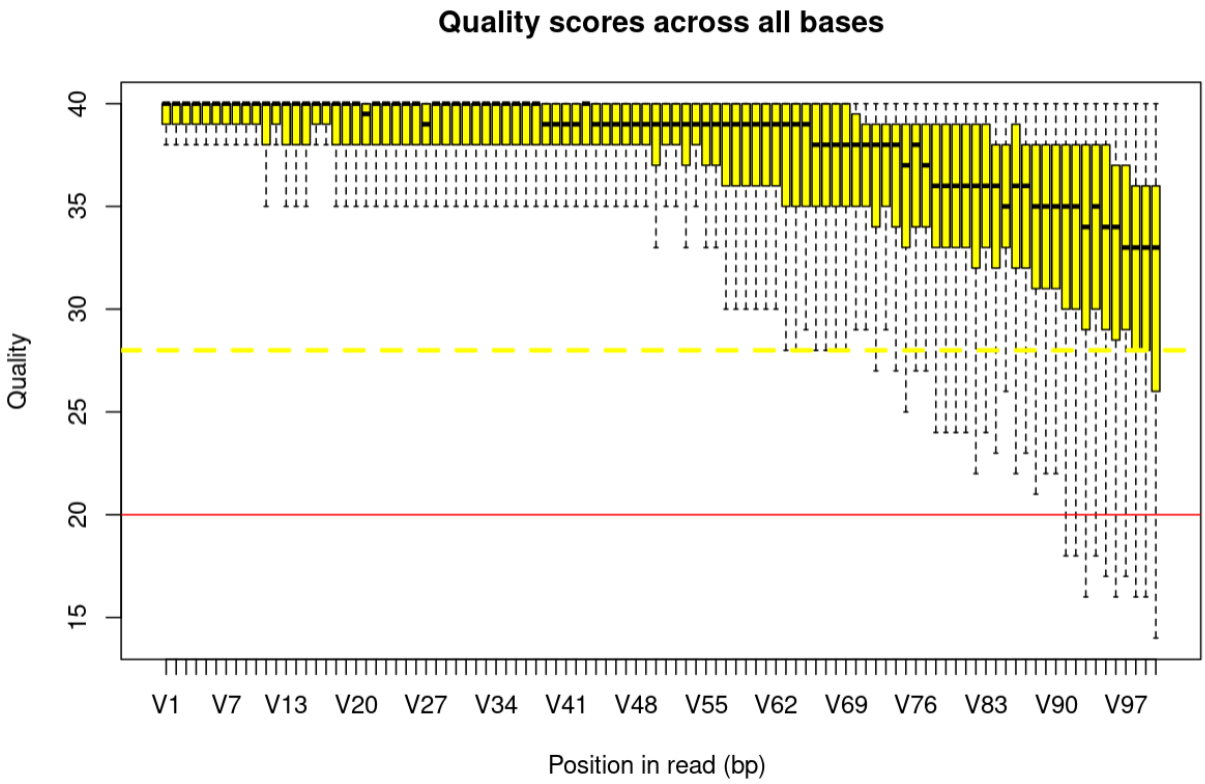
.....

[998]
"ATGGTAACACCAGAGTGACCGCCAATAACCGGCACTTCAACTTCGCCTGGCTGTTTGCCTTTCAGTTCCGCAACAAAGGTGTTG

```
[999] "+"
[1000]
"GDGBGGGGGGGGDBDGGGGGEEGGGHHHHHHHGHHHHHHHHGEGE>HHHH@FEDGDGBEGEGDEDEGGG>GEEE@E?
7=:5?8?:?DDBD>3<???BAD##"
[ reached getOption("max.print") -- omitted 3000 entries ]
```

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

```
quality_matrix <- as(quality(fastq), "matrix")
t_m <- t(quality_fastq)
df_mtx <- as.data.frame(quality_matrix[1:10000,1:100])
boxplot(df_mtx,
        col="YELLOW" ,
        main="Quality scores across all bases",
        ylab="Quality" ,
        xlab="Position in read (bp) "
        ,outline=FALSE)
# abline(h=20,28, col="blue")
abline(h=c(20,28), col=c("red", "yellow"), lty=c(1,2), lwd=c(1, 3))
```



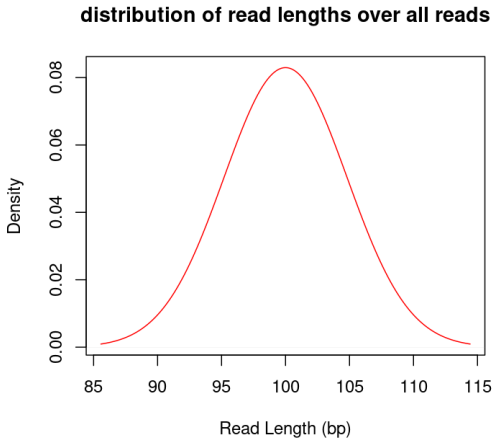
VI. Show the distribution of read lengths using a density plot.

```
#width - length of individual reads

#read_lengths
readlength= width(fastq)
print(readlength)
```

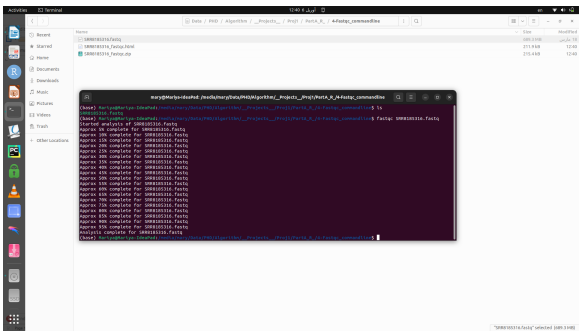
```
#another way of calculating read_lengths
widths= reads@ranges@width

class(widths)
plot(density(read_lengths),
     main="distribution of read lengths over all reads",
     xlab="Read Length (bp)",
     ylab="Density",
     col='red')
```

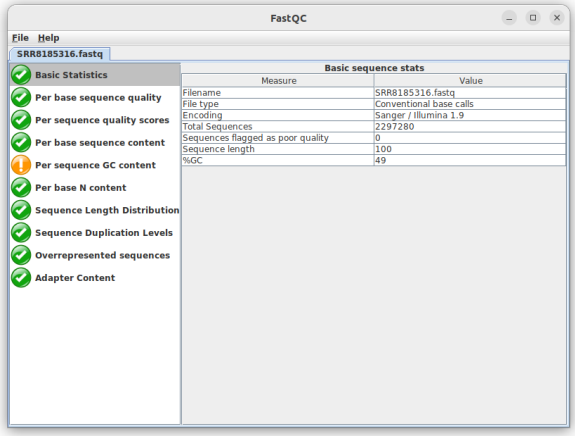


4. briefly explain the quality metrics of the FastQC report, then, Perform quality control of the reads using FastQC and interpret the results. Complete this task using both the command-line function as well as the FastQC graphical interface. (you may download FastQC from here.)

FastQC command-line

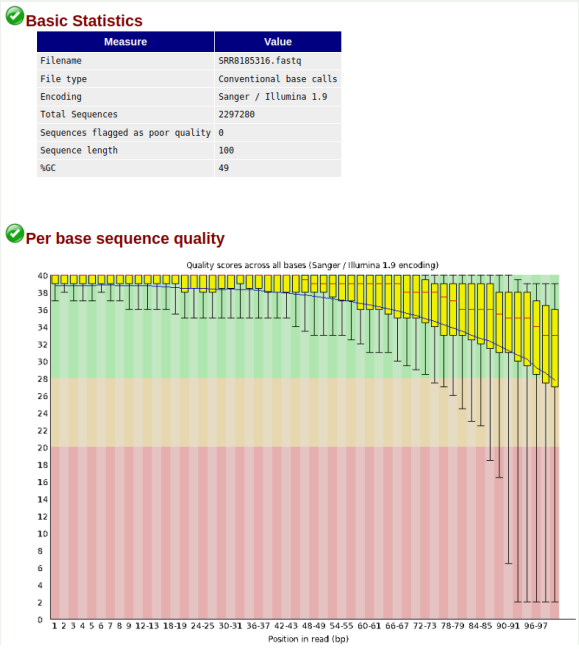


FastQC graphical interface

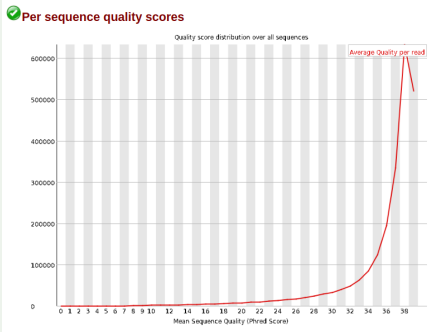


FastQC used for quality control of high-throughput sequencing data. It generates a report that provides various metrics to evaluate the quality of the sequencing data.

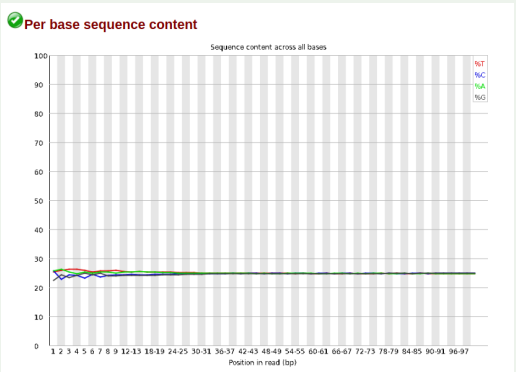
1. **Per base sequence quality:** It measures the **quality of each base in the sequence**. The quality score reflects the confidence of the base call. **Low quality scores** can indicate **sequencing errors or poor quality data**.



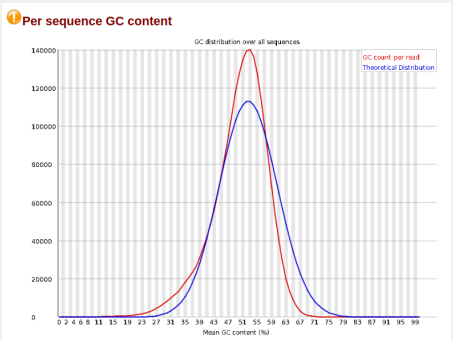
2. **Per sequence quality scores:** This metric provides the **distribution of the quality scores** for **all the sequences.** A good quality score distribution should be high and narrow, indicating that most of the sequences have high quality.



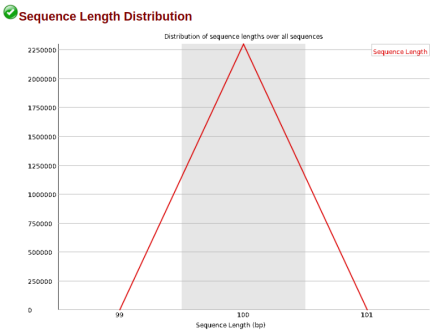
3. **Per base sequence content:** This graph displays the percentage of each base (A, C, G, T) at each position in the read. A smooth and uniform distribution of base content across the read length is indicative of high-quality data.



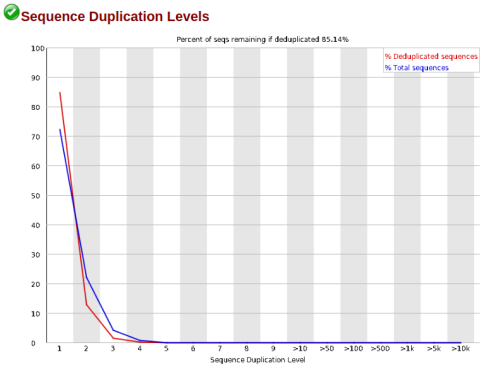
4. **GC content distribution:** It measures the **percentage of G and C bases in the sequence**. A balanced GC content indicates that the sequencing library is of high quality.



5. **Sequence length distribution:** It provides information about the **length distribution of the sequences**. A narrow distribution indicates that the sequencing library is uniform in size and quality.



6. **Sequence duplication levels:** It measures the **level of redundancy in the sequencing data**. High levels of sequence duplication can indicate PCR bias or library preparation errors.



7. **Overrepresented sequences:** This metric reports the sequences that are overrepresented in the data. These sequences can come from adapters,primers, or contamination.

Part B - Denovo Genome Assembly

Install SPAdes, Canu, Quast, BWA, Samtools, Pilon, or any alternative that you want. Then follow the steps below.

1. Run SPAdes to generate draft genome assemblies from short reads.

+SPAdes (genome assembler) is a popular genome assembly tool that can assemble genomes from **short-read data** generated by next-generation sequencing platforms like Illumina.

+SPAdes uses a combination of **de Bruijn graph** and **overlap-layout-consensus (OLC) algorithms** to **assemble the reads into contigs,** and can also generate scaffolds and perform gap filling.

+SPAdes supports various input types, including paired-end, mate-pair, and single-end reads, as well as hybrid data sets consisting of multiple sequencing technologies.

+It also includes features like error correction, adapter trimming, and quality control to improve the accuracy and completeness of the assembly.

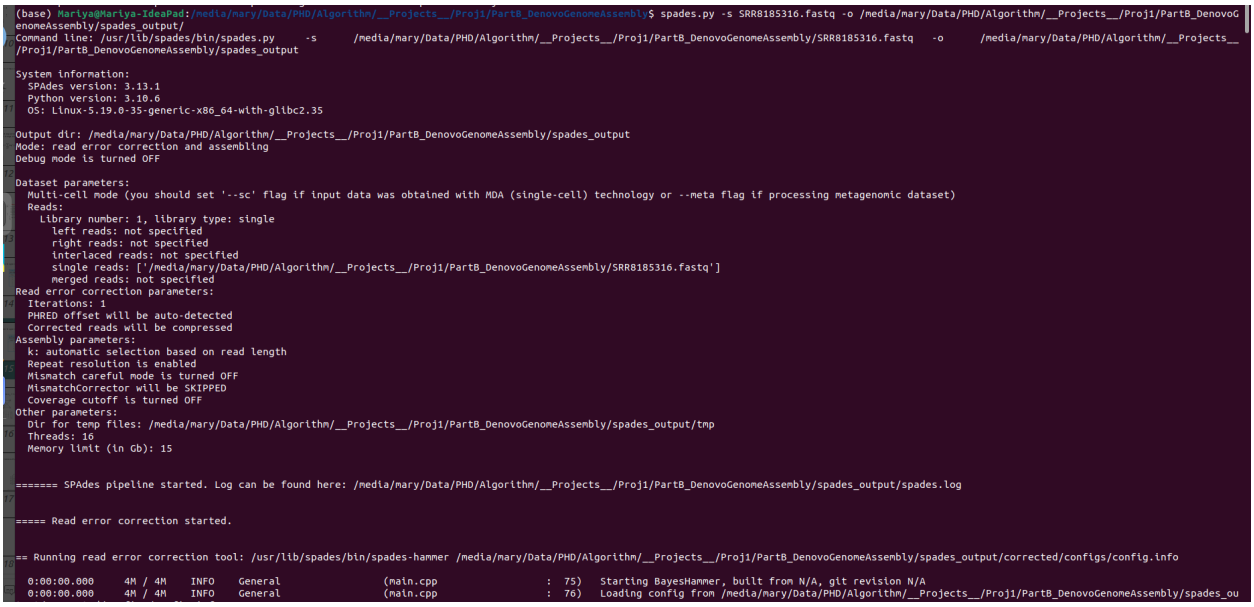
SPAdes parameters:

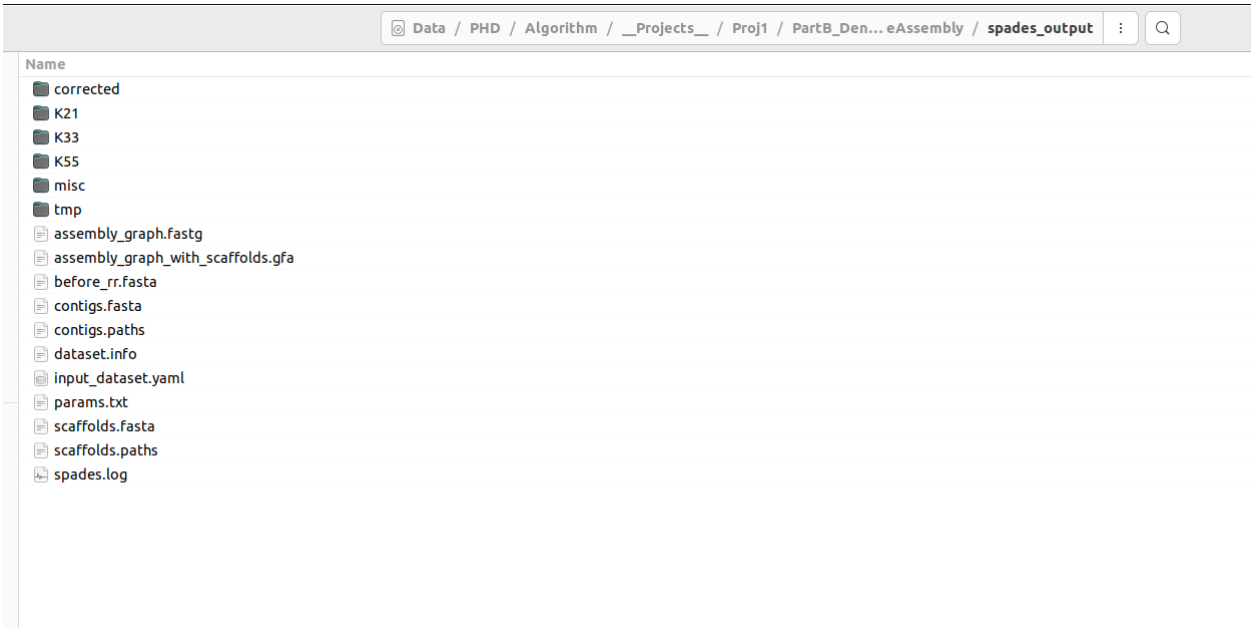
- `-s`: specifies the input reads as single-end reads instead of paired-end reads

- `-o`: specifies the output directory for the assembly results

SPAdes also provides various output files: including the **assembled contigs in FASTA format,** scaffolds, and various statistics and quality metrics.

```
spades.py -s SRR8185316.fastq -o /media/mary/Data/PHD/Algorithm/__Projects__/Proj1/PartB_DenovoGenomeAssembly/spades_output/
```





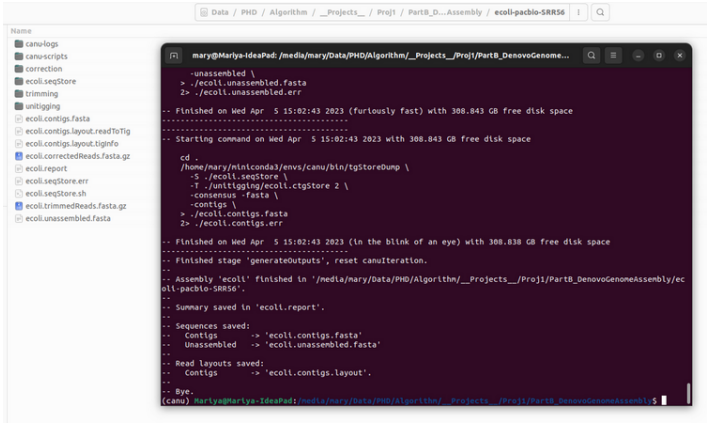
2. Run **Canu** to generate draft genome assemblies from long reads.

Canu is a popular genome assembly tool that can generate draft genome assemblies from **long-read sequencing** data, such as those generated by PacBio(SRR10538956.fastq) or Oxford Nanopore Technologies.

Canu parameters:

- `-p ecoli`: This specifies the prefix for the output filenames.
- `-d ecoli-pacbio-SRR56`: This specifies the output directory for the assembly.
- `genomeSize=4.8m`: This specifies the expected genome size in megabases (Mb). This is used by Canu to estimate the appropriate assembly parameters.
- `-pacbio-raw SRR10538956.fastq`: Path to the input PacBio long-read sequencing data in FASTQ format.

```
canu -p ecoli -d ecoli-pacbio-SRR56 genomeSize=4.8m -pacbio-raw SRR10538956.fastq
```



3. Assess the quality of the draft genome assembly (**long reads**) using **Quast** and compare it to the reference genome (Download the reference genome from here)

+QUAST (QUality ASsessment Tool) is a popular software tool for **evaluating the quality of genome** assemblies.

QUAST can **compare the assembly to a reference genome** or **a set of conserved genes**, and generate various statistics and visualizations to assess the completeness, accuracy, and contiguity of the assembly.

downloaded reference genome and put it in this path:

references/GCF_genomic.fna

copying canu contig to this assembly folder:

assemblies/canu_longread_ecoli.contigs.fasta

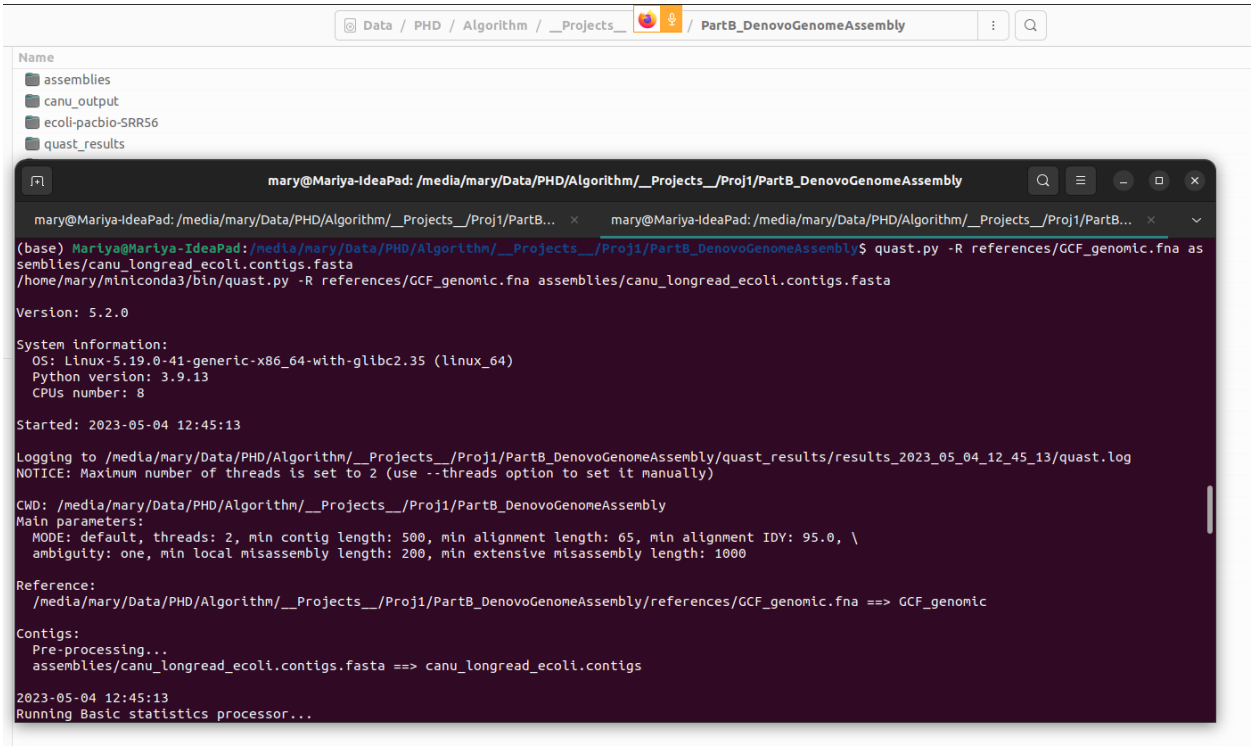
quest:

`-r:` specifies the reference genome in FASTA format (`references/GCF_genomic.fna`)

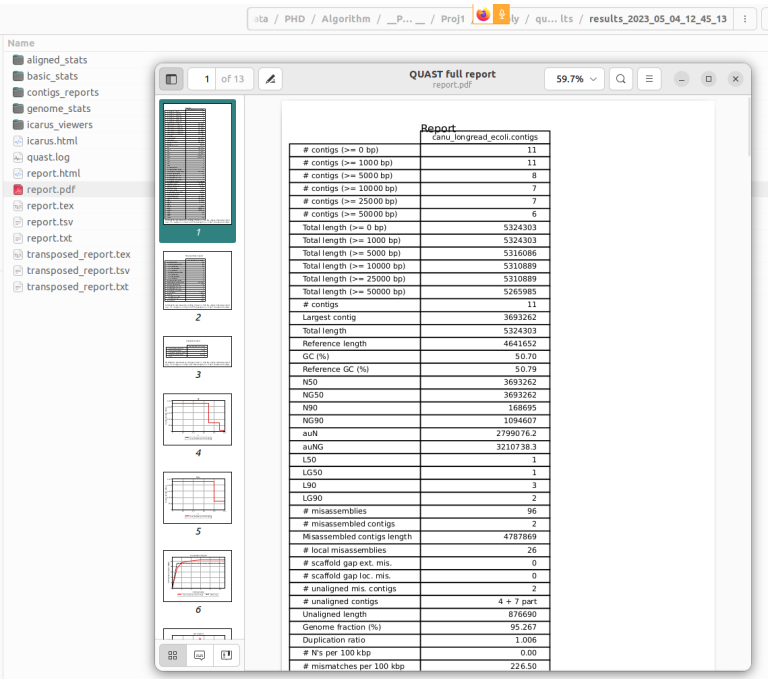
`-o` specifies the output directory for the evaluation results. I forgot to specify the output directory, but Canu created an output file for me anyway.

The assembly contigs generated by Canu (`assemblies/canu_longread_ecoli.contigs.fasta`) are specified as the input to be evaluated.

```
quast.py -R references/GCF_genomic.fna assemblies/canu_longread_ecoli.contigs.fasta
```



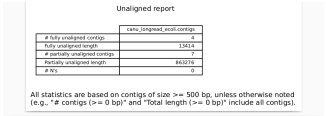
quast PDF report:

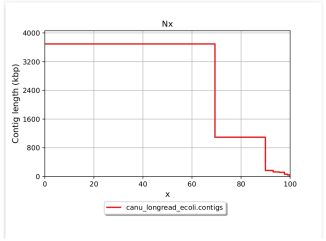


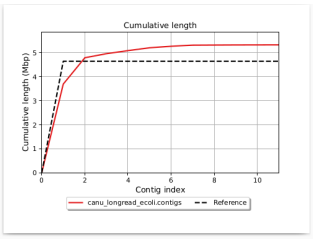
Misassemblies report

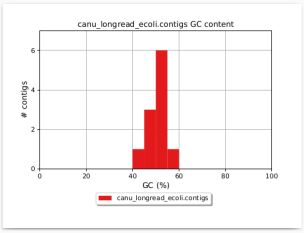
	canu_longread_ecoli.contigs
# misassemblies	96
# contig misassemblies	96
# c. relocations	90
# c. translocations	0
# c. inversions	6
# scaffold misassemblies	0
# s. relocations	0
# s. translocations	0
# s. inversions	0
# misassembled contigs	2
Misassembled contigs length	4787869
# local misassemblies	26
# scaffold gap ext. mis.	0
# scaffold gap loc. mis.	0
# unaligned mis. contigs	2
# mismatches	10071
# indels	378
# indels (<= 5 bp)	312
# indels (> 5 bp)	66
Indels length	4468

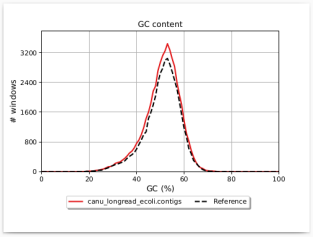
All statistics are based on contigs of size >= 500 bp, unless otherwise noted (e.g., "# contigs (>= 0 bp)" and "Total length (>= 0 bp)" include all contigs).

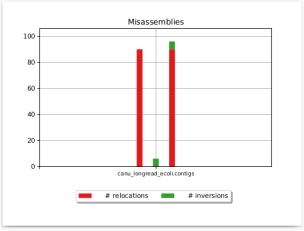












4. Long-read technology can generate notable errors, leading to low accuracy in genome assemblies. To address this challenge, Pilon can be used, a tool that detects and corrects errors in genome assemblies, including single nucleotide polymorphisms (SNPs), insertions, and deletions (indels). In this section, map Illumina short-read data to the PacBio assembly using BWA and then use Pilon to identify and correct assembly errors.

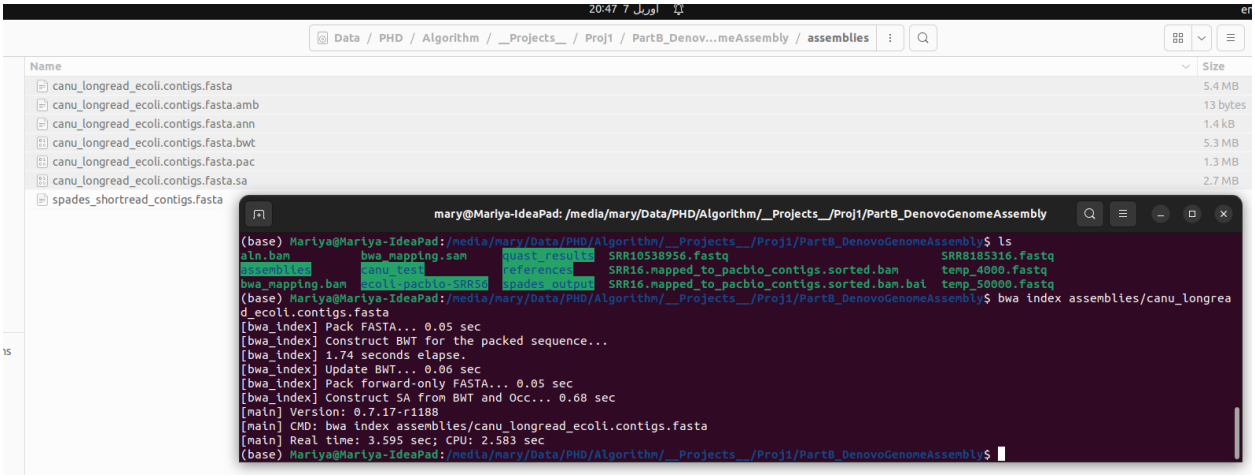
a. index canu contig by bwa

Consider the Canu contig.fasta file as the reference genome and create an index for it.

copying canu contig to this **assembly folder:**

assemblies/canu_longread_ecoli.contigs.fasta

```
bwa index assemblies/canu_longread_ecoli.contigs.fasta
```



The command "bwa index" is used to **create** an **index of a reference genome (canu_longread_ecoli.contigs.fasta)** using the BWA software. BWA stands for **Burrows-Wheeler Aligner** and is a popular software tool used for **aligning sequence reads to a reference genome**.

b. Map Illumina short-read data “SRR8185316.fastq” to the PacBio assembly “canu_longread_ecoli.contigs.fasta” using BWA and create a sorted bam file by samtools “shortread_mapped_to_pacbio_contigs.sorted.bam”

The command "bwa mem -t 4 -p" is used to **align sequence reads to a reference genome** using the BWA software.

commands:

- "bwa mem": This specifies that we want to use the "mem" algorithm within BWA for read alignment.
- "-t 4": This specifies we are using 4 threads to speed up the alignment process.
- "-p": This specifies that we want to **output** the alignments in **the SAM format**, which is a standard file format for storing sequence alignment data.

The command "samtools sort -o shortread_mapped_to_pacbio_contigs.sorted.bam" is used **to sort the alignments in a SAM or BAM file** by their genomic coordinates using the Samtools software.

Here's what each part of the command does:

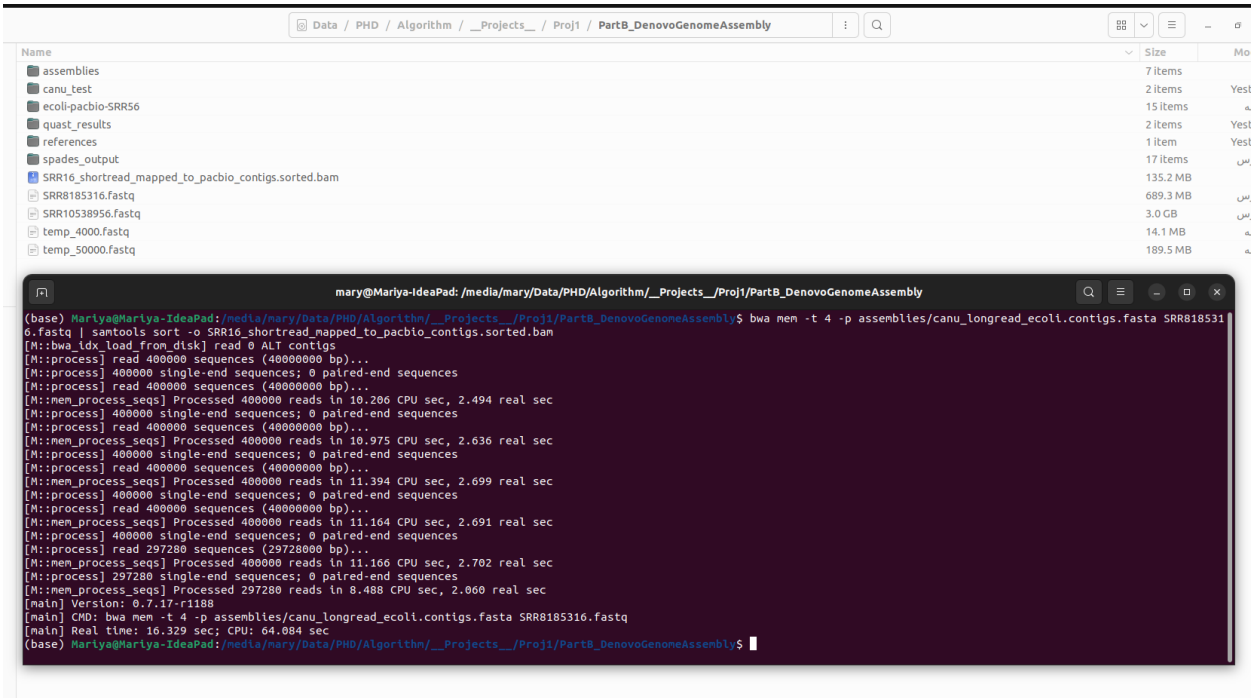
- "samtools sort": This specifies that we want to use the "**sort**" function within Samtools to sort the alignments.
- "-o": This specifies the name of the **output file** that will contain the sorted alignments.
- "shortread_mapped_to_pacbio_contigs.bam": This is the **name of the input file** containing the sequence alignments that we want to sort.

In this example, "shortread_mapped_to_pacbio_contigs.bam" is the name of the input file containing the sequence alignments to be sorted, and "shortread_mapped_to_pacbio_contigs.sorted.bam" is the name of the output file where the sorted alignments will be written.

```
bwa mem -t 4 -p assemblies/canu_longread_ecoli.contigs.fasta SRR8185316.fastq | samtools sort -o shortread_mapped_to_pacbio_contigs.sorte
d.bam
```

- The command "samtools index" is used to create an index file of a sorted BAM file. This index file allows for faster retrieval of alignments from the BAM file.
- The command "pilon --genome assemblies/canu_longread_ecoli.contigs.fasta --bam shortread_mapped_to_pacbio_contigs.sorted.bam --output pilon_output" is used to run Pilon on the sorted BAM file to identify and correct errors in the assembly. The corrected assembly is output to a new FASTA file named "pilon_output.fasta".

- The corrected assembly can be evaluated using Quast to assess the improvement in assembly quality compared to the original PacBio assembly.



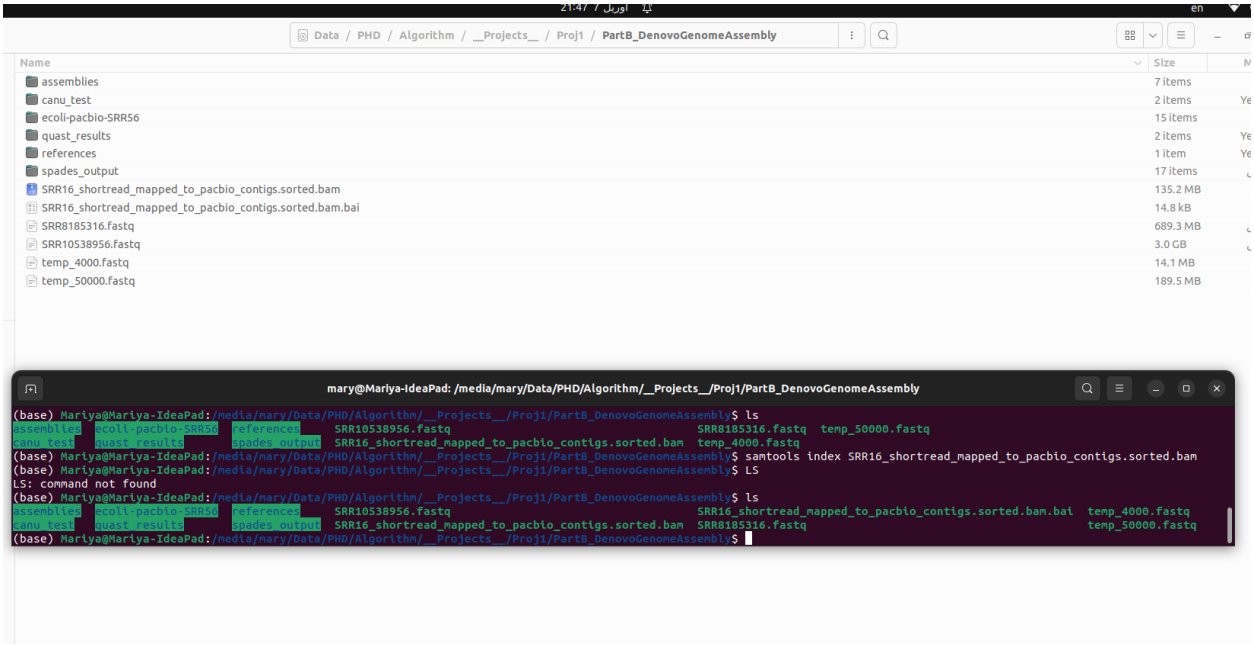
c. index “shortread_mapped_to_pacbio_contigs.sorted.bam” file by samtools

The command "samtools index SRR16_shortread_mapped_to_pacbio_contigs.sorted.bam" is used to create an index file for a sorted BAM file using the Samtools software.

commands:

- "samtools index": This specifies that we want to use the "index" function within Samtools to create an index file.
- "SRR16_shortread_mapped_to_pacbio_contigs.sorted.bam": This is the name of the sorted BAM file that we want to create an index for.

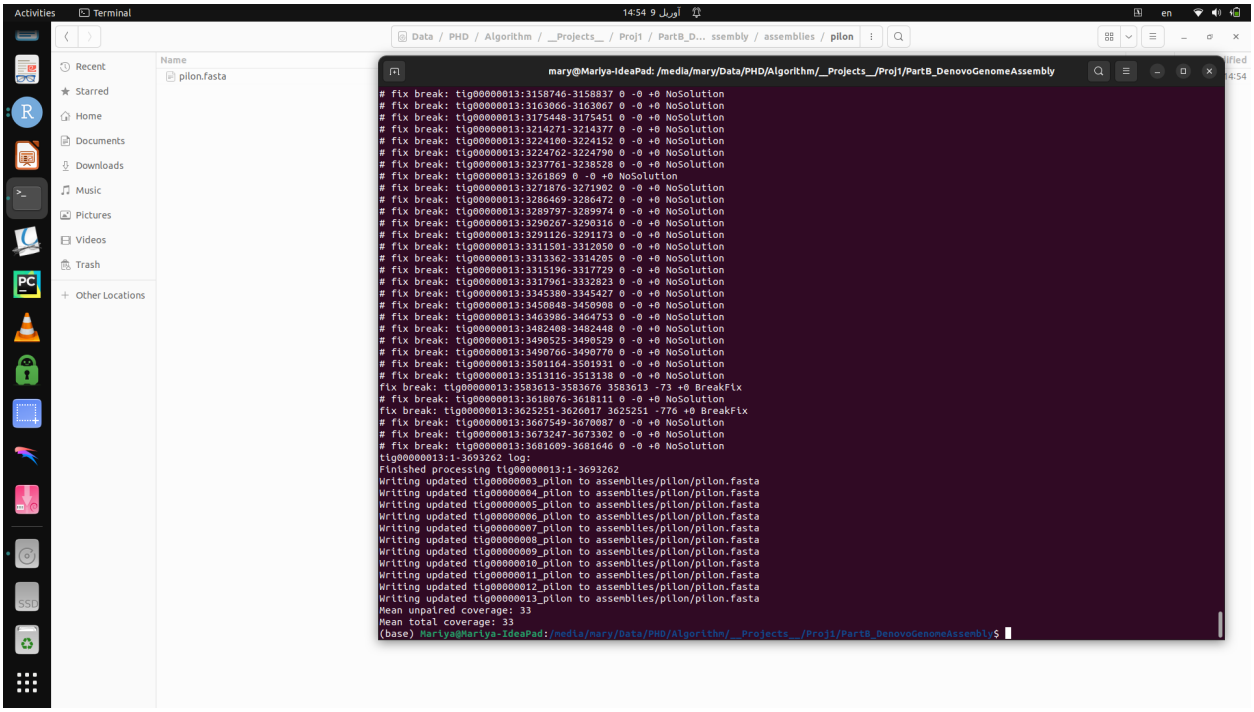
```
samtools index SRR16_shortread_mapped_to_pacbio_contigs.sorted.bam
```



d. Run Pilon to identify and correct assembly errors using short-read sequencing data.

- `Xmx160G`: This specifies the maximum amount of memory to allocate for the Pilon process (160 gigabytes in this case).
- `-genome assemblies/canu_longread_ecoli.contigs.fasta`: This specifies the **input** Canu assembly file in FASTA format (`assemblies/canu_longread_ecoli.contigs.fasta`).
- `-unpaired SRR16_shortread_mapped_to_pacbio_contigs.sorted.bam`: This specifies the input short-read data that aligned to pacbio contigs in sorted BAM format. **Short read is unpaired** then we used **--unpaired.**
- `-outdir assemblies/pilon`: This specifies the **output directory** for the polished assembly and other output files generated by Pilon.

```
pilon -Xmx160G --genome assemblies/canu_longread_ecoli.contigs.fasta --unpaired SRR16_shortread_mapped_to_pacbio_contigs.sorted.bam --outdi
```



Pilon output:

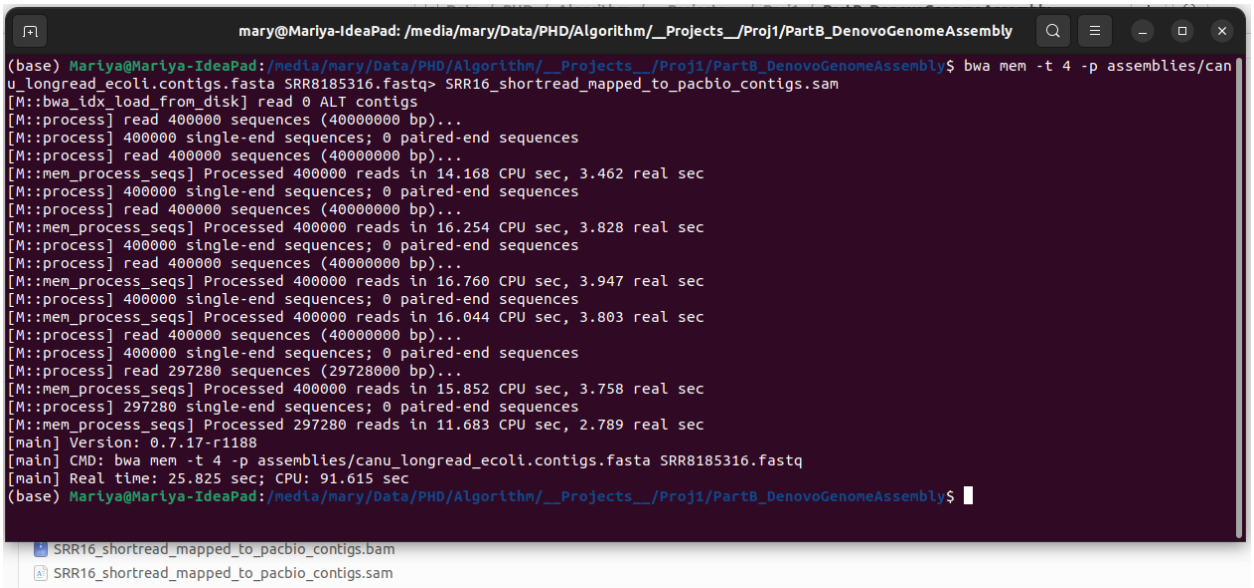
pilon.fasta: This is the **improved genome assembly** in FASTA format. It **contains the corrected contigs or scaffolds**, and may also include additional sequences that were added by Pilon, such as new contigs or gaps.

Part C - Mapping & variant calling

1. Print the head of the SAM file that create ib section5 part B. Explain what you see for the first hit (you can do this step either in Linux or R).

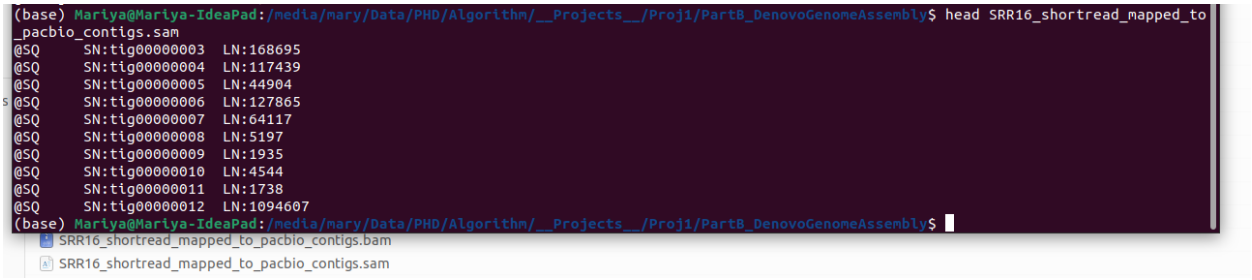
Create .SAM file (SRR16_shortread_mapped_to_pacbio_contigs.sam)

```
bwa mem -t 4 -p assemblies/canu_longread_ecoli.contigs.fasta SRR8185316.fastq> SRR16_shortread_mapped_to_pacbio_contigs.sam
```

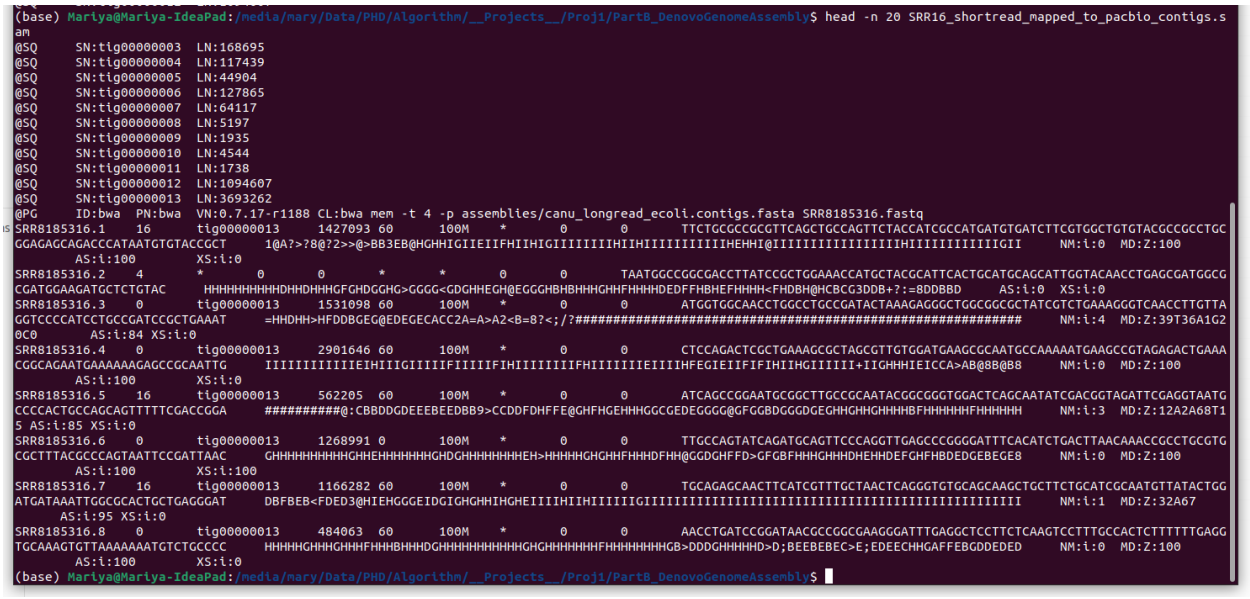


Print the head of the SAM file

```
head SRR16_shortread_mapped_to_pacbio_contigs.sam
```




```
#print 20 line
head -n 20 SRR16_shortread_mapped_to_pacbio_contigs.sam
```



1. Convert the SAM file to a BAM file. Hint: use samtools view, samtools sort, samtools index.

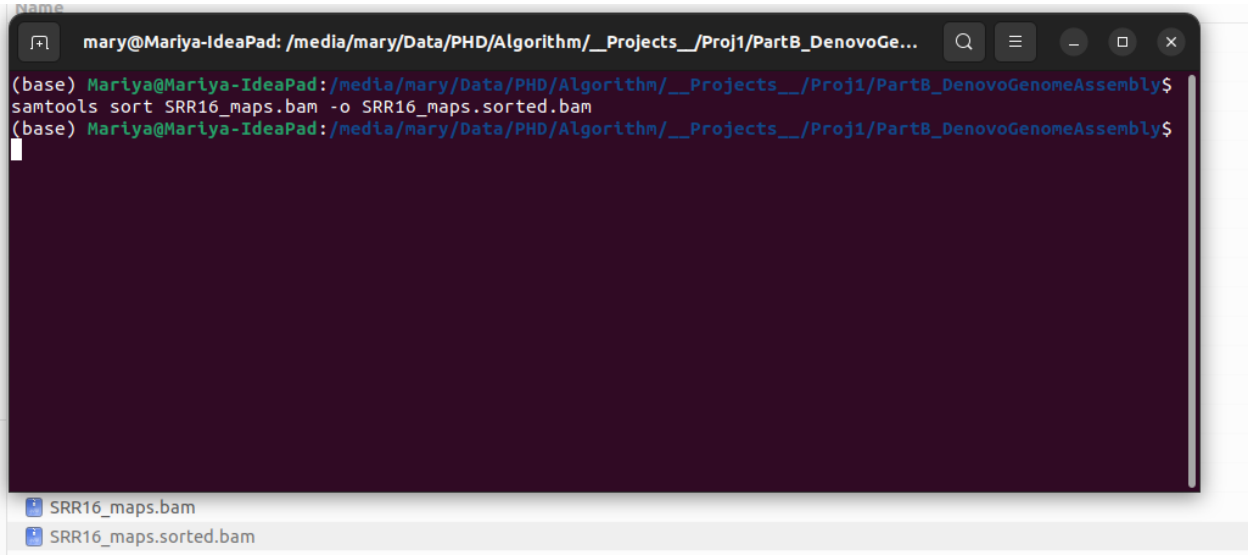
Convert .sam to .bam (samtools view)

```
samtools view -S -b SRR16_shortread_mapped_to_pacbio_contigs.sam > SRR16_maps.bam
```

- "samtools view": This specifies that we want to use the "view" command within SAMtools, which can be used to convert between different sequence alignment file formats.
- "-Sb": These two options specify that we want to convert the input file from **SAM format (-S) to BAM format (-b).**
- "SRR16_shortread_mapped_to_pacbio_contigs.sam": This is the **name of the input SAM** file that we want to convert to the BAM file format.
- "> SRR16_maps.bam": This specifies the **name of the output** file where the **BAM-formatted** alignment data will be written.

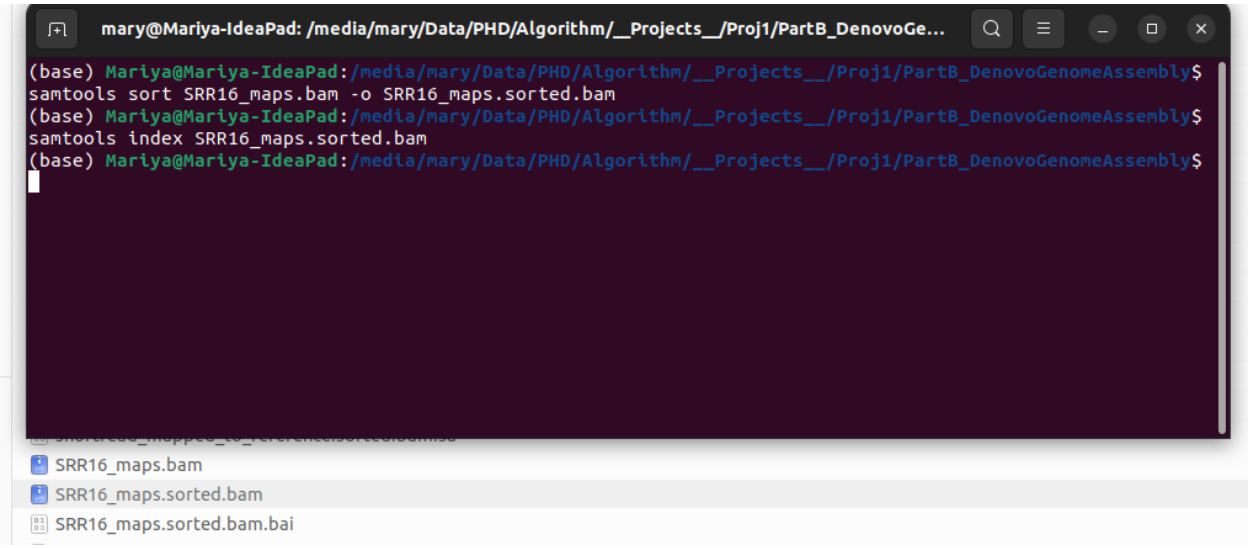
Sort .bam file (samtools sort)

```
samtools sort SRR16_maps.bam -o SRR16_maps.sorted.bam
```



Index .sorted.bam file (samtools index)

```
samtools index SRR16_maps.sorted.bam
```



1. Use the Integrative Genomics Viewer (IGV) to visualize the mapped reads in a 200-b genomic region of your choice. In IGV, firstly, select the reference genome fasta and GTF file (GTF is optional).

Index refference genome(GCF_genomic.fna)

```
bwa index GCF_genomic.fna
```

```
(base) Mariya@Mariya-IdeaPad:/media/mary/Data/PHD/Algorithm/__Projects__/Proj1/PartB_DenovoGenomeAssembly/references$ bwa index GCF_genomic.fna
[bwa_index] Pack FASTA... 0.03 sec
[bwa_index] Construct BWT for the packed sequence...
[bwa_index] 0.91 seconds elapse.
[bwa_index] Update BWT... 0.02 sec
[bwa_index] Pack forward-only FASTA... 0.02 sec
[bwa_index] Construct SA from BWT and Occ... 0.35 sec
[main] Version: 0.7.17-r1188
[main] CMD: bwa index GCF_genomic.fna
[main] Real time: 2.013 sec; CPU: 1.332 sec
```

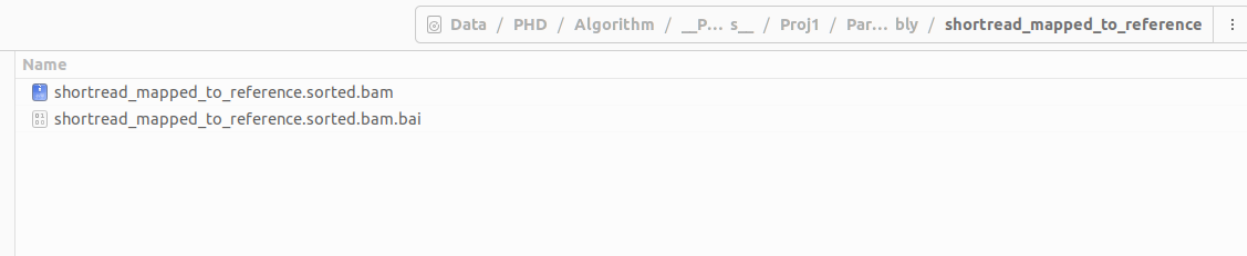
Align short read (SRR8185316.fastq) with references genome and create a sorted.bam file

```
bwa mem -t 4 -p references/GCF_genomic.fna SRR8185316.fastq | samtools sort -o shortread_mapped_to_reference.sorted.bam
```

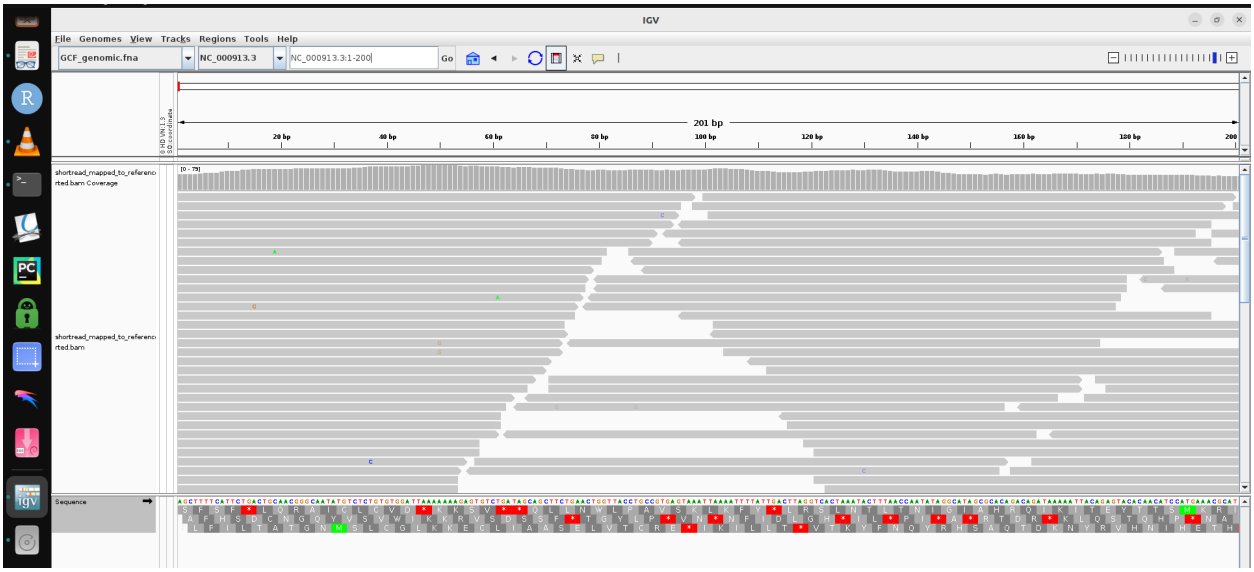
```
(base) Mariya@Mariya-IdeaPad:/media/mary/Data/PHD/Algorithm/__Projects__/Proj1/PartB_DenovoGenomeAssembly/references$ cd ..
(base) Mariya@Mariya-IdeaPad:/media/mary/Data/PHD/Algorithm/__Projects__/Proj1/PartB_DenovoGenomeAssembly$ bwa mem -t 4 -p references/GCF_genomic.fn
a SRR8185316.fastq | samtools sort -o shortread_mapped_to_reference.sorted.bam
[M::bwa_idx_load_from_disk] read 0 ALT contigs
[M::process] read 400000 sequences (40000000 bp)...
[M::process] 400000 single-end sequences; 0 paired-end sequences
[M::process] read 400000 sequences (40000000 bp)...
[M::mem_process_seqs] Processed 400000 reads in 9.005 CPU sec, 2.189 real sec
[M::process] 400000 single-end sequences; 0 paired-end sequences
[M::process] read 400000 sequences (40000000 bp)...
[M::mem_process_seqs] Processed 400000 reads in 9.766 CPU sec, 2.305 real sec
[M::process] 400000 single-end sequences; 0 paired-end sequences
[M::process] read 400000 sequences (40000000 bp)...
[M::mem_process_seqs] Processed 400000 reads in 9.872 CPU sec, 2.337 real sec
[M::process] 400000 single-end sequences; 0 paired-end sequences
[M::process] read 400000 sequences (40000000 bp)...
[M::mem_process_seqs] Processed 400000 reads in 10.076 CPU sec, 2.382 real sec
[M::process] 400000 single-end sequences; 0 paired-end sequences
[M::process] read 297280 sequences (29728000 bp)...
[M::mem_process_seqs] Processed 400000 reads in 9.966 CPU sec, 2.374 real sec
[M::process] 297280 single-end sequences; 0 paired-end sequences
[M::mem_process_seqs] Processed 297280 reads in 7.396 CPU sec, 1.785 real sec
[main] Version: 0.7.17-r1188
[main] CMD: bwa mem -t 4 -p references/GCF_genomic.fna SRR8185316.fastq
[main] Real time: 15.246 sec; CPU: 57.009 sec
```

Index .sorted.bam file by samtools

```
samtools index shortread_mapped_to_reference.sorted.bam
```



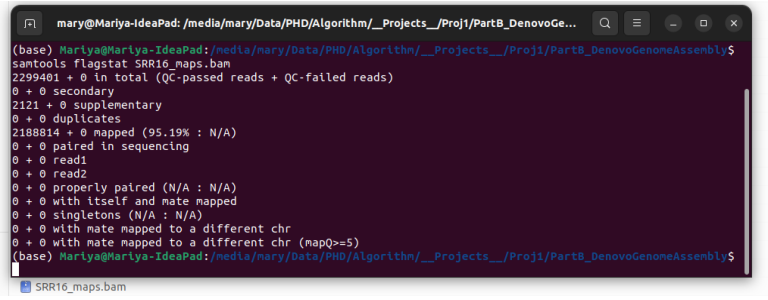
Run IGV with references genome and **shortread_mapped_to_reference.sorted.bam**



4. Determine the percentage of short reads that are mapped to the assembled genome from long reads and reference genome. Hint: use samtools flagstat.

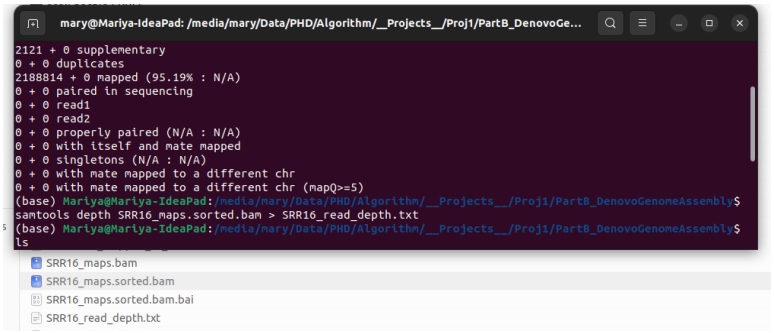
The "samtools flagstat" command generate **a summary of the alignment statistics** from **a BAM file**, including the **total number of reads**, the **number of mapped reads**, and the **number of reads with specific alignment flags.**

```
samtools flagstat SRR16_maps.bam
```

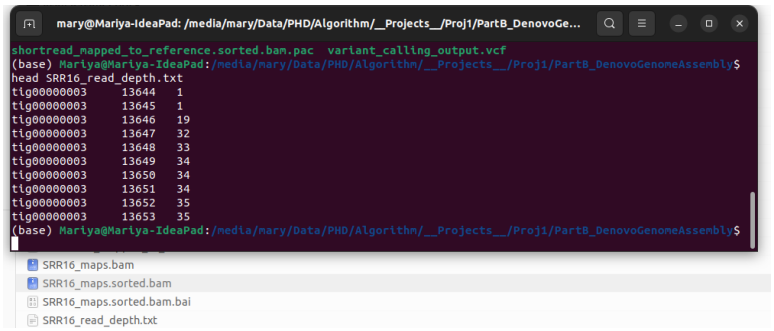


5. Get the read depth for the sorted BAM file at all positions of the assembled genome and report **mean** of all reads. Hint: use samtools depth.

```
samtools depth SRR16_maps.sorted.bam > SRR16_read_depth.txt
```

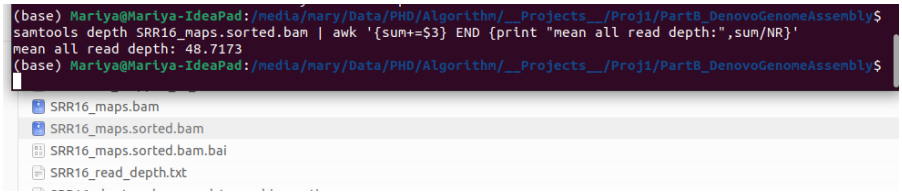



```
#it creates a read_depth_sbam.txt file  with depth, head shown head of file.
head SRR16_read_depth.txt
```



Print mean of depth

```
samtools depth SRR16_maps.sorted.bam | awk '{sum+=$3} END {print "mean all read depth:",sum/NR}'
```



6. Make yourself familiar with the **CIGAR** format. How do you interpret belows?
   “29S21=1X25=”
   “20M2I1M1D10M” “5M10N25M”

The CIGAR (Compact Idiosyncratic Gapped Alignment Report) format is a compact representation of the alignment of a sequence read to a reference genome. The CIGAR string consists of a series of integers and single-letter codes that describe the alignment of the read.

"29S21=1X25="

- "29S": This indicates that the first 29 bases in the read were soft-clipped, meaning they were not matched to the reference but were still present in the read.
- "21=": This indicates that the next 21 bases in the read matched exactly to the reference.

- "1X": This indicates that the next base in the read differed from the reference by a single base substitution.
- "25=": This indicates that the final 25 bases in the read matched exactly to the reference.

CIGAR represents an alignment where the first 29 bases of the read were not aligned to the reference, the next 21 bases matched exactly, the next base differed by a single base substitution, and the final 25 bases matched exactly.

"20M2I1M1D10M"

- "20M": This indicates that the first 20 bases in the read matched exactly to the reference.
- "2I": This indicates that the next two bases in the read were insertions relative to the reference.
- "1M": This indicates that the next base in the read matched exactly to the reference.
- "1D": This indicates that the next base in the reference was deleted relative to the read.
- "10M": This indicates that the final 10 bases in the read matched exactly to the reference.

Therefore, this CIGAR string represents an alignment where the first 20 bases of the read matched exactly to the reference, followed by two insertions, a single base match, a single base deletion, and finally 10 bases matching exactly.

"5M10N25M"

- "5M": This indicates that the first 5 bases in the read matched exactly to the reference.
- "10N": This indicates that the next 10 bases in the read were skipped (i.e., not aligned) relative to the reference.
- "25M": This indicates that the final 25 bases in the read matched exactly to the reference.

Therefore, this CIGAR string represents an alignment where the first 5 bases of the read matched exactly to the reference, followed by a 10-base gap, and finally 25 bases matching exactly.

7. Perform variant calling using the reference genome(or error corrected assembled genome from long reads) and your BAM file. Save the output as a VCF file. Hint: you may need the following commands/options: samtools mpileup, bcftools, multiallelic-caller algorithm

Variant calling is the process by which we **identify variants from sequence data**

Commands:

- "samtools mpileup" is a command in the SAMtools software package that **generates a pileup file from a sorted BAM file** and a reference genome. A pileup file is a **tab-delimited text file that reports the read depth and the base calls at each position** in a reference genome, based on the alignments stored in a sorted BAM file

1. "-u" specifies that the output should be in uncompressed BAM format, which ensures compatibility with "bcftools call".
2. "-f references/GCF_genomic.fna" specifies the name of the reference genome in FASTA format that we want to use for variant calling.
3. "SRR16_maps.sorted.bam" is the name of the sorted BAM file that we want to analyze.
4. "|" pipes the output of "samtools mpileup" to "bcftools call".
5. "bcftools call" is a command in the BCFtools software package that performs variant calling on a pileup file.
6. "-v" specifies that only variant sites should be included in the output VCF file.
7. "-m" specifies the use of a multiallelic-caller algorithm, which is a probabilistic algorithm for calling variants that can handle multiple alternate alleles.
8. "-Ov" specifies that the output should be in VCF format.
9. "-o SRR16_variant_calling.vcf" specifies the name of the output file.

```
samtools mpileup -uf references/GCF_genomic.fna SRR16_maps.sorted.bam | bcftools call -vm -Ov -o SRR16_variant_calling.vcf
```

```
mary@Mariya-IdeaPad: /media/mary/Data/PHD/Algorithm/__Projects__/Proj1/PartB_DenovoGenomeAssembly
[E::faidx_adjust_position] The sequence "tig00000013" was not found
[E::faidx_adjust_position] The sequence "tig00000013" was not found
[E::faidx_adjust_position] The sequence "tig00000013" was not found
[E::faidx_adjust_position] The sequence "tig00000013" was not found
[E::faidx_adjust_position] The sequence "tig00000013" was not found
[E::faidx_adjust_position] The sequence "tig00000013" was not found
[E::faidx_adjust_position] The sequence "tig00000013" was not found
[E::faidx_adjust_position] The sequence "tig00000013" was not found
[E::faidx_adjust_position] The sequence "tig00000013" was not found
[E::faidx_adjust_position] The sequence "tig00000013" was not found
[E::faidx_adjust_position] The sequence "tig00000013" was not found
[E::faidx_adjust_position] The sequence "tig00000013" was not found
[E::faidx_adjust_position] The sequence "tig00000013" was not found
[E::faidx_adjust_position] The sequence "tig00000013" was not found
[E::faidx_adjust_position] The sequence "tig00000013" was not found
[E::faidx_adjust_position] The sequence "tig00000013" was not found
[E::faidx_adjust_position] The sequence "tig00000013" was not found
[E::faidx_adjust_position] The sequence "tig00000013" was not found
(base) Mariya@Mariya-IdeaPad:/media/mary/Data/PHD/Algorithm/__Projects__/Proj1/PartB_DenovoGenomeAssembly$
```

If you're interested, I have implementation files that I can share with you. Let me know if you'd like me to send them to you.

Implementation directory