BINF7001 Assessment1
Ssh s4536711@binf-training.biosci.uq.edu.au

PART A

```
#!/bin/bash

# Step 2: Create shortcuts to practice files
#ln is the command used to create links (shortcuts) in Unix/Linux.
#-s create a symbolic link that points to another file or directory.
ln -s /opt/BINF7001/2024/Prac3_2024/Llactis_illuminaData.zip .
ln -s /opt/BINF7001/2024/Prac3_2024/Ll_MG1363.fasta .

# Step 3: Unzip the read data
unzip Llactis_illuminaData.zip

# Step 4: Index the reference genome
#bwa: This is the command-line tool called Burrows-Wheeler Aligner (BWA).
#index: This is the subcommand of BWA that is used to create index files for a reference genome.
bwa index Ll_MG1363.fasta

# Step 5: Perform alignment for both strains
#bwa aln reference.fasta reads.fastq > output.sai
#aln: This is the subcommand of BWA that performs the alignment of single-end reads. It aligns each read in the FASTQ file to the reference genome provided.
bwa aln Ll_MG1363.fasta Llactis_illuminaData/0514c_3_AC0617ACXX_ATCACG_L004.1.fastq > 0514c.1.sai
bwa aln Ll_MG1363.fasta Llactis_illuminaData/0514c_3_AC0617ACXX_ATCACG_L004.2.fastq > 0514c.2.sai
bwa aln Ll_MG1363.fasta Llactis_illuminaData/1768int6_AC0617ACXX_TTAGGC_L004.1.fastq > 1768int6.1.sai
bwa aln Ll_MG1363.fasta Llactis_illuminaData/1768int6_AC0617ACXX_TTAGGC_L004.2.fastq > 1768int6.2.sai

# Step 6: Perform paired-end mapping for both strains
#bwa sampe reference.fasta sai_file_1 sai_file_2 reads_file_1.fastq reads_file_2.fastq > output.sam
#sampe: combines the two .sai files generated from the bwa aln and the original paired-end FASTQ files to create a single SAM file.
bwa sampe Ll_MG1363.fasta 0514c.1.sai 0514c.2.sai Llactis_illuminaData/0514c_3_AC0617ACXX_ATCACG_L004.1.fastq
Llactis_illuminaData/0514c_3_AC0617ACXX_ATCACG_L004.2.fastq > 0514c.Ll_MG1363.sam
bwa sampe Ll_MG1363.fasta 1768int6.1.sai 1768int6.2.sai Llactis_illuminaData/1768int6_AC0617ACXX_TTAGGC_L004.1.fastq
Llactis_illuminaData/1768int6_AC0617ACXX_TTAGGC_L004.2.fastq > 1768int6.Ll_MG1363.sam

# Step 7: Convert SAM files to BAM format and sort them
samtools view -bS 0514c.Ll_MG1363.sam > 0514c.Ll_MG1363.bam
samtools view -bS 1768int6.Ll_MG1363.sam > 1768int6.Ll_MG1363.bam
samtools sort 0514c.Ll_MG1363.bam > 0514c.Ll_MG1363.sorted.bam
samtools sort 1768int6.Ll_MG1363.bam > 1768int6.Ll_MG1363.sorted.bam

# Step 8: Clean up unnecessary files to save storage space
rm 0514c.Ll_MG1363.sam
rm 1768int6.Ll_MG1363.sam

# Step 9: Rename the sorted BAM files
mv 1768int6.Ll_MG1363.sorted.bam 1768int6.Ll_MG1363.bam
mv 0514c.Ll_MG1363.sorted.bam 0514c.Ll_MG1363.bam

# Step 10: Perform variant calling using bcftools
#bcftools mpileup -f reference.fasta input.bam | bcftools call --ploidy 1 -vc > output.vcf
#mpileup: This is a subcommand of bcftools that creates a pileup of aligned reads (from the BAM file) at each position in the reference genome to help identify variants (e.g., SNPs, insertions, deletions).
#The -f option specifies the reference genome file in FASTA format. The reads in the BAM file will be aligned to this reference genome.
#bcftools call: This subcommand performs variant calling from the pileup data generated by bcftools mpileup.
#--ploidy 1: specifies that the organism being studied is haploid (having only one set of chromosomes) like Lactococcus lactis.
#-v: Output variant sites only (i.e., don't include positions where no variation is detected).
#-c: Perform variant calling.
bcftools mpileup -f Ll_MG1363.fasta 0514c.Ll_MG1363.bam | bcftools call --ploidy 1 -vc > 0514c_vc_result.vcf
bcftools mpileup -f Ll_MG1363.fasta 1768int6.Ll_MG1363.bam | bcftools call --ploidy 1 -vc > 1768int6_vc_result.vcf

# Completion message
echo "Read mapping and variant calling completed successfully!
```

Question B1: How much disk space is taken up by the Llactis_illuminaData directory? Roughly how much compression was achieved in the original Llactis_illuminaData.zip file?

- **du -sh Llactis_illuminaData** # du (Disk Usage), -s (Summarize)m -h (Human-readable):
- The output is **249M Llactis_illuminaData**, it means that the Llactis_illuminaData directory, occupies 249 Megabytes of disk space.
- **ls -lh /opt/BINF7001/2024/Prac3_2024/Llactis_illuminaData.zip**
- ls: lists the contents of a directory or details about a specific file.
- -l (Long format), -h (Human-readable)
- The output is **-rwxr-xr-x 1 root root 75M Aug 6 07:25 /opt/BINF7001/2024/Prac3_2024/Llactis_illuminaData.zip**
- **75M: The file size is 75 Megabytes**.
- **The compression ratio = 249 MB / 75 MB ≈ 3.32** meaning the uncompressed data is about 3.32 times larger than the compressed ZIP file. This shows that the ZIP file is effectively compressing the data, reducing the storage space required by more than three times.

Question B2: How many sequence reads are in each of the two samples?

- **grep -c "^@" Llactis_illuminaData/0514c_3_AC0617ACXX_ATCACG_L004.1.fastq**
- **Output: 300089 reads**
- **grep -c "^@" Llactis_illuminaData/1768int6_AC0617ACXX_TTAGGC_L004.1.fastq**
- **Output: 303387 reads**
- grep: A command used to search for specific patterns in a file.
- -c: Tells grep to count the number of lines that match the given pattern rather than printing the matching lines themselves.
- "^@": This is the pattern being searched for.
- ^: This is a regular expression (regex) symbol that matches the start of a line.
- @: This matches the "@" character.
- Together, "^@" ensures that only lines that start with an "@" symbol are counted.

Question B3: What command will add the strain name immediately after the string "HWI-ST212" in each header line of 0514c_3_AC0617ACXX_ATCACG_L004.1.fastq?

- **sed 's/^@HWI-ST212/&_0514c/' Llactis_illuminaData/0514c_3_AC0617ACXX_ATCACG_L004.1.fastq > modified_0514c_3_AC0617ACXX_ATCACG_L004.1.fastq**
- `sed' applies a specified transformation, and then outputs the result.
- `s/`: The substitution command in `sed`. It replaces a specified pattern with a new text string.
- `^`: A regex pattern that matches the start of a line.
- `@HWI-ST212`: This matches the text `@HWI-ST212` at the beginning of each line (
- `&`: refers to the entire matched pattern (`@HWI-ST212` in this case).
- `_0514c`: This is the new text added immediately after `@HWI-ST212`.
- `Llactis_illuminaData/0514c_3_AC0617ACXX_ATCACG_L004.1.fastq`: This is the input FASTQ file. `sed` will read this file line by line and apply the substitution to every line that matches the pattern.
- `> modified_0514c_3_AC0617ACXX_ATCACG_L004.1.fastq`: This redirects the output to a new file called `modified_0514c_3_AC0617ACXX_ATCACG_L004.1.fastq`. This means that the original file remains unchanged, and the modified headers will be written to this new file.
- **head modified_0514c_3_AC0617ACXX_ATCACG_L004.1.fastq**
- **@HWI-ST212_0514c:195:C0617ACXX:4:1101:1196:1988 1:N:0:ATCACG**

Question B4: In every fastq file header there is a numeric suffix in the first string (1988). @HWI-ST212:195:C0617ACXX:4:1101:1196:1988 1:N:0:ATCACG

a) How many unique numbers are there?

- **awk -F: '{print $7}' modified_0514c_3_AC0617ACXX_ATCACG_L004.1.fastq | sort | uniq | wc -l**
- -F:: This tells awk to use : as the field separator.
- {print $7}: Extracts the 7th field (which is the numeric suffix in the header).
- sort | uniq: Sorts the numbers and filters out duplicates.
- wc -l: Counts the number of unique lines, which corresponds to the unique numbers.
- **Output: 103500**

b) Which number is represented the most? How many times?

- **grep -oP '^@HWI-ST212_0514c:[0-9]+:[^:]+:[^:]+:[^:]+:[^:]+:\K[0-9]+' modified_0514c_3_AC0617ACXX_ATCACG_L004.1.fastq | sort | uniq -c | sort -nr | head -n 20**
- grep: This is a command-line utility used to search for patterns in text.
- -o: This option tells grep to print only the part of the line that matches the pattern, not the entire line.
- P: This option enables Perl-compatible regular expressions, which allow for more advanced pattern matching.
- '^@HWI-ST212_0514c:: The ^ indicates the start of the line, ensuring that the match begins at the beginning of the header.
- [0-9]+:: Matches a sequence of digits followed by a colon. This is used to match the various fields in the FASTQ header.
- [^:]+:: Matches characters that are not a colon, followed by a colon to match the non-numeric fields.
- \K: This "forgets" everything matched up to this point. It tells grep to start the final match (output) from this point forward.
- [0-9]+: Matches one or more digits, which corresponds to the numeric suffix that follows the last colon in the header.
- | sort: This is a pipe, which takes the output of the previous command (grep) and uses it as the input for the next command (sort).
- | uniq -c: This command counts the number of occurrences of each unique numeric suffix. The -c option precedes each line with the number of times the suffix appears.
- Result: a list of numeric suffixes along with how many times each one appears.
- | sort -nr:: sorts the list in numeric (-n) and reverse (-r) order, so that the most frequent numeric suffixes appear at the top.
- | head -n 20: displays the 20 most frequent numeric suffixes and their counts.
- **Output: 11 73132**

c) What is a regular expression that will match all of these numbers and not match any other numbers in the header? Explain your choice. Avoid the "." Special character in your regex.

- :[0-9]+:[0-9]+:[0-9]+:[0-9]+:[0-9]+:([0-9]+)
- [0-9]+: Matches one or more digits. This pattern is repeated for each numeric field in the header, separated by colons.
- ([0-9]+): The final numeric suffix is captured in a group, ensuring this specific number is isolated for further processing.
- This regex matches the numeric suffix without including any other numbers.

d) Use sed to add the numeric suffix to the HWI-ST212 prefix in the header to make a new header as follows for every line in 0514c_3_AC0617ACXX_ATCACG_L004.1.fastq: @HWI-ST212:195:C0617ACXX:4:1101:1196:1988 1:N:0:ATCACG @HWI-ST212-1988

- **sed 's/^\(@HWI-ST212_0514c\)\(:[0-9]\+:[^:]\+:[^:]\+:[^:]\+:[^:]\+:\)\([0-9]\+\)/\1-\3\2\3/' modified_0514c_3_AC0617ACXX_ATCACG_L004.1.fastq > B4_modified_0514c_3_AC0617ACXX_ATCACG_L004.1.fastq**
- s/.../.../: This is the substitution command in sed. It searches for a pattern (the part between the first pair of slashes) and replaces it with a new string (the part between the second pair of slashes).
- ^: indicates the start of a line. It's used here to ensure that the pattern match occurs at the beginning of the line.
- \(@HWI-ST212_0514c\): This is the prefix in the FASTQ file headers.
- \(...\): Parentheses are used to capture this portion of the matched text. The captured text can be referred to later using \1.
- :[0-9]\+: Matches a colon followed by one or more digits. This matches the fields in the header that are numeric.
- [^:]\+: Matches one or more characters that are not a colon. This matches fields that are alphanumeric. This entire pattern captures the middle part of the header, which includes the various fields separated by colons. This part is captured as \2.
- \([0-9]\+\): This part captures the final numeric suffix, which is the last field in the header before the space. This captured text can be referred to as \3.
- \1: Refers to the captured prefix @HWI-ST212_0514c.
- -\3: Adds a hyphen and appends the captured numeric suffix (\3).
- \2: Refers to the middle part of the header (all fields up to the final numeric suffix).
- \3: Adds the numeric suffix again to keep the rest of the header intact.
- head B4_modified_0514c_3_AC0617ACXX_ATCACG_L004.1.fastq
- **Output: @HWI-ST212_0514c-1988:195:C0617ACXX:4:1101:1196:1988 1:N:0:ATCACG**

Question B5: How many single nucleotide variants* (SNVs) relative to MG1363 are shared by the two mutant strains? *SNVs include single nucleotide substitutions and indels. SNPs generally refer only to single nucleotide substitutions in the context of bacterial genome data.

- **cut -f1,2,4,5 0514c_vc_result.vcf | sort > 0514c_snvs.txt**
- **cut -f1,2,4,5 1768int6_vc_result.vcf | sort > 1768int6_snvs.txt**
- cut -f1,2,4,5: extracts specific fields from each line of a file.
- Field 1: Chromosome (CHROM) – identifies the chromosome or contig.
- Field 2: Position (POS) – the position of the variant on the chromosome.
- Field 4: Reference Base (REF) – the reference nucleotide at that position.
- Field 5: Alternate Base (ALT) – the alternate nucleotide found in the mutant strain.
- **comm -12 0514c_snvs.txt 1768int6_snvs.txt > shared_snvs.txt**
- comm is a command that compares two sorted files line by line
- The -12 option tells comm to: -1: Suppress lines unique to the first file (0514c_snvs.txt). -2: Suppress lines unique to the second file (1768int6_snvs.txt). Therefore, comm -12 outputs only the lines that are common to both files (i.e., SNVs that are identical in both strains, including chromosome, position, reference base, and alternate base).
- **wc -l shared_snvs.txt → Output 94**

Question B6: Which positions have substitutions unique to each mutant strain? Explain why these substitutions may be relevant for understanding the heat-resistant phenotype.

- **comm -23 0514c_snvs.txt 1768int6_snvs.txt > 0514c_unique_snvs.txt**
- -2: Suppress the output of column 2 (lines unique to the second file).
- -3: Suppress the output of column 3 (lines common to both files).
- 0514c_unique_snvs.txt contain SNVs that are found only in the 0514c strain and not in 1768int6.
- **comm -13 0514c_snvs.txt 1768int6_snvs.txt > 1768int6_unique_snvs.txt**
- 1768int6_unique_snvs.txt contain SNVs that are found only in the 1768int6 strain and not in 0514c.
- **Less 0514c_unique_snvs.txt**
- Position 1487370: Reference: ATTTTTTT, Alternate: ATTTTTT, This indicates a deletion where one T is missing in the 0514c strain compared to the reference genome.
- Position 1538252: Reference: C, Alternate: A, This indicates a C-to-A substitution at this position in the 0514c strain.
- Position 2332872, Reference C, Alternate: T: This indicates a C-to-T substitution at this position in the 0514c strain.
- Deletions (like the one at position 1487370) could lead to frameshifts or loss of function in coding regions, potentially altering protein function.
- Substitutions (like the C-to-A change at 1538252) could lead to non-synonymous mutations if they occur in coding regions, potentially altering protein structure and stability. The functional impact of these SNVs, especially those affecting critical genes or regulatory regions, will gain insights into how these genetic differences might confer a survival advantage under heat stress.

Question B7: We need to tidy up the folders. What command(s) will compress all of the files within the assessment1 folder, delete the original folder and leave you with a single .tgz archive file?

tar czvf assessment1.tgz assessment1

- tar: A utility to create and manipulate archive files.
- c: Create a new archive, z: Compress the archive using gzip, v: Verbose mode, shows the progress of the command, f: Specifies the filename of the archive.
- assessment1.tgz: The name of the resulting archive file.
- assessment1: The directory to be archived
- **rm -r assessment1**

Question B8: We need to keep a record of the practicals. What command(s) will generate a readme file that contains the current date and time, and a list of all the files in each folder week1/ week2/ and week3/ with human readable file sizes, and a list of all the commands that you've executed over the last three weeks?

```
# Step 1: Add the current date and time to README.txt
echo "Practical Log - $(date)" > README.txt

#Step 2: List all files in week1, week2, and week3 with human-readable sizes
for week in week1 week2 week3; do
echo -e "\nFiles in $week:" >> README.txt
ls -lh $week >> README.txt
done

#Step 3: Append the command history to README.txt
echo -e "\nCommand history for the last three weeks:" >> README.txt
history >> README.txt
```

PART C

Introduction (10 marks):

Read mapping is a process in genomic informatics that is essential for aligning short DNA or RNA sequences, called reads, to a reference genome or transcriptome. This alignment step is the basis for a variety of genome and transcriptome analyses, including identification of genetic variants, quantification of gene expression, and detection of structural variants. By mapping reads to a reference genome, researchers can identify single nucleotide polymorphisms (SNPs), insertions, deletions, and other structural variations. In transcriptomics, read mapping quantifies gene expression by counting the number of reads that align to specific genomic regions, such as exons. Three read mappers exemplify different approaches to this process: BWA-MEM, HISAT, and conLSH. BWA-MEM utilize FM indexes, a data structure derived from the Burrows-Wheeler transform (BWT), to align reads to a reference genome. It is known for its accuracy, speed, and memory efficiency, making it great for large data sets such as whole-genome sequencing. HISAT is another FM index-based mapper designed for RNA sequencing data. It introduces hierarchical indexing that combines FM indexing with a graph-based representation of the genome, allowing efficient alignment of reads across exon-exon junctions. HISAT is optimized for splice junction detection and is widely used in RNA-seq analysis workflows. conLSH represents a non-FM index-based approach that uses locality-sensitive hashing (LSH) to map reads and is useful for repetitive sequences or noisy data. This method excels in environments where traditional mappers struggle, such as metagenomics or highly repetitive genomic regions. Three studies demonstrate the practical application of these read mappers. Pham-Quoc et al. (2019) focused on optimizing the BWA-MEM algorithm through FPGA-based IP cores to achieve significant acceleration while reducing power consumption. Kim et al. (2015) introduced HISAT, demonstrating its efficiency and accuracy for RNA-seq read alignment supported by a hierarchical indexing strategy. Chakraborty et al. (2021) proposed S-conLSH, a novel alignment-free mapper that uses gap context-based LSH to map noisy long reads, exhibiting high sensitivity and speed in genome analysis. These studies highlight the diverse approaches and innovations for read mapping, each tailored to specific genomic and transcriptomic challenges.

Common Features (15 marks): 450 words

BWA-MEM and HISAT both utilize the FM-index, derived from the Burrows-Wheeler Transform (BWT), which allows for efficient compression and indexing of the reference genome. This enables rapid searching and alignment, making both tools capable of handling large datasets with minimal computational overhead. S-conLSH, while not based on the FM-index, also emphasizes computational efficiency through its use of Locality-Sensitive Hashing (LSH). By grouping similar sequences into localized "buckets," S-conLSH can quickly identify potential alignment candidates, reducing the need for exhaustive searches across the entire genome (Chakraborty et al., 2021). Both BWA-MEM and HISAT are designed to operate within limited memory environments, which is important for large-scale genome projects. HISAT is noted for its hierarchical indexing approach, which significantly reduces memory requirements to 4.3 gigabytes even when dealing with complex spliced alignments (Kim et al., 2015). S-conLSH also aims to be memory-efficient, especially by focusing on the most relevant portions of the data rather than attempting to align every read against the entire genome. In terms of general applicability, all three mappers are designed to be versatile across different types of genomic data. BWA-MEM is effective for both short and long reads, and its extensions allow for the alignment of reads from technologies like PacBio and Oxford Nanopore. HISAT is optimized for RNA sequencing data, predominantly for aligning reads across spliced regions, but it also performs well with DNA data. S-conLSH, while primarily focused on noisy long reads, is adaptable to various genomic contexts, making it a valuable tool for metagenomics and other applications where traditional aligners may struggle.

Both BWA-MEM and HISAT come with significant computational costs, with time and memory consumption dominated by alignment overhead. Additionally, alignment algorithms often struggle to correctly align distant homologs in the “twilight zone” of 20–35% sequence identity, where weak similarities are difficult to distinguish from random noise (Chakraborty et al., 2021). To address these challenges the concept of ‘context-based’ Locality-Sensitive Hashing (conLSH), hashes close points in the feature space into localized slots of the hash table. This method shows that the context of DNA base plays a key role in determining sequence similarity. Although S-conLSH prioritizes mapping accuracy, leading to a lower mapping ratio (99.9%) compared to other tools like Minimap2 and lordFAST, it provides a higher degree of alignment precision by leaving some reads unaligned if potential target locations are not found. However, S-conLSH has a higher memory footprint, requiring about 13GB for indexing the entire human genome (Chakraborty et al., 2021).

Notable Differences (15 marks): 450 words

Hashing-based methods like S-conLSH are easier to implement compared to suffix tree and BWT-FM based methods like BWA-MEM and HISAT. This simplicity allows for rapid development for specialized applications like alignment-free mapping. Hashing methods are optimized for exact matches, making them fast and efficient for specific sequence alignment tasks. However, FM-index-based methods, like those used in BWA-MEM and HISAT, support both exact and inexact matches, which handle mismatches and indels during the alignment process (Alser et al., 2021). Hashing methods require large index sizes, which can consume significant memory resources, whereas FM-index methods are highly compressed, resulting in much smaller index sizes and making them more memory-efficient, which is important when working with large genomes (Alser et al., 2021). Once the index is built, querying is extremely fast in hashing methods, with operations typically being O(1). This allows tools like S-conLSH to rapidly locate sequences. FM-indexing methods are slower in querying, especially when dealing with variable or large seed lengths, which can affect overall performance. Hashing methods use a fixed seed length for each index, which can limit their adaptability to different sequence types. FM-index-based methods can use either fixed or variable seed lengths, providing more flexibility in aligning diverse sequences with varying levels of complexity (Alser et al., 2021).

BWA-MEM, which is based on the Smith-Waterman algorithm, is effective at handling indels due to its dynamic programming approach. This method allows BWA-MEM to accurately align reads containing small indels by identifying the optimal local alignment, even in complex genomic regions. However, BWA-MEM places computational pressure on current systems, as evidenced by the BWA-MEM/GATK toolkit accounting for 36% of the total execution time in genome analysis processes (Pham-Quoc et al., 2019). This highlights the need for architectural enhancements to support BWA-MEM's intensive demands. HISAT, while capable of handling indels, is optimized more for splicing events and may not be as robust as BWA-MEM in regions with high indel rates. The smaller error rate of short-read sequencing technologies used with BWA-MEM and HISAT facilitates the local pairwise alignment process, making it easier to account for indels compared to long reads. In contrast, S-conLSH handles indels through its use of spaced seeds in locality-sensitive hashing, which allows it to tolerate a certain level of indels by focusing on the overall context of the sequence rather than requiring exact base-to-base matches.

BWA-MEM uses a scoring system that penalizes mismatches while still allowing them within the alignment, ensuring that even reads with a few mismatches can be correctly aligned. The lower error rate of short-read technologies enhances BWA-MEM’s ability to detect single-nucleotide polymorphisms (SNPs), as these technologies provide higher coverage, making SNP detection more reliable. HISAT, with its focus on RNA-seq data, is designed to align reads with multiple mismatches, in regions where RNA editing or sequencing errors are common. HISAT’s ability to use splice site information during alignment also helps align reads

with mismatches that occur near exon boundaries. S-conLSH's approach to mismatches does not rely on a traditional alignment method. Instead, it uses spaced seeds to create a context-based alignment, which can tolerate mismatches within the gaps of the spaced seeds. This allows S-conLSH to maintain alignment accuracy even in noisy data sets where mismatches are frequent, although it may not perform as well in detecting SNPs due to the inherent higher error rate and lower coverage of long-read technologies. Splicing is a critical challenge in RNA-seq data, and HISAT is specifically designed to address this (Alser et al., 2021). HISAT employs a hierarchical index that includes information about known splice sites, enabling it to accurately align reads that span exon-exon junctions. This makes HISAT effective for detecting alternative splicing events. BWA-MEM is not optimized for this task and may struggle with complex splicing patterns, especially in genes with multiple isoforms. S-conLSH does not focus on splicing per se, as it is designed more for alignment-free mapping of long reads.

Methodology Comparison (15 marks):

Pham-Quoc et al. (2019) utilized BWA-MEM in their study to accelerate the Seed extension phase of the algorithm using FPGA (Field-Programmable Gate Array) technology. The researchers aimed to enhance the speed and efficiency of BWA-MEM, particularly focusing on the Seed extension stage, which is computationally intensive. The study involved comparing the execution time of BWA-MEM on an Intel Core i5 processor with their custom FPGA implementation. The dataset used for these experiments was legally provided by NCBI, and BWA-MEM version 0.7.11 was selected for consistency. The primary focus was on performance comparison, with FPGA implementations showing up to 350× speed-up compared to the general-purpose processor. This study highlighted the potential of hardware acceleration in genomic data processing, particularly for tasks like Seed extension in BWA-MEM.

Kim et al. (2015) focused on HISAT, a read-mapper specifically designed for RNA-seq data, and its ability to handle the complexities of splicing. HISAT utilizes a hierarchical indexing strategy that combines global FM indexing with local FM indexes. This method enables HISAT to efficiently align reads across exons, particularly those that span exon-exon junctions. The study categorized reads into three groups to demonstrate HISAT's capabilities: (i) reads mapping within an exon or across two exons with substantial overlap, (ii) reads spanning exons with minimal overlap or across multiple exons, and (iii) reads that might be incorrectly mapped to processed pseudogenes. HISAT's hierarchical indexing is optimized to minimize memory usage, reducing cache misses and improving alignment speed. The study used RNA-seq data, primarily consisting of reads ranging from 75 to 150 bp, with a focus on human genome data, which included 246,208 introns. HISAT effectively handled long introns by utilizing multiple local indexes, with a maximum intron length of 500,000 bp. The study demonstrated HISAT's ability to align reads with mismatches and indels, highlighting its robustness in RNA-seq data analysis, especially for detecting alternative splicing events. The authors emphasized the efficiency of HISAT in managing large-scale RNA-seq data while maintaining high alignment accuracy and minimal memory footprint.

Chakraborty et al. (2021) S-conLSH operates through two main steps: reference genome indexing and read mapping. The reference genome is divided into overlapping windows, which are then hashed into hash tables using specially designed S-conLSH functions. These hash tables, termed 'h_index' and 'Hashtab,' store sequences based on their hash values, allowing S-conLSH to cluster similar sequences together efficiently. For mapping, S-conLSH computes hash values for each noisy long read and retrieves sequences from the reference genome that match these hash values. The final locations of the sequences with the highest hits are chained together, providing an alignment-free mapping of the read. Although the default output of S-conLSH is alignment-free, it offers an option to generate SAM format alignments using the ksw library for cases where base-level alignment is required. The study primarily utilized SMRT sequencing data, focusing on the challenges posed by noisy, long reads, such as high error rates and repetitive sequences. S-conLSH proved effective in these contexts, demonstrating its ability to manage data that might be problematic for traditional alignment methods.

Data and Approach Critique (15 marks):

The use of BWA-MEM in the study by Pham-Quoc et al. demonstrated the algorithm's strengths in efficiently handling the seed extension phase through an FPGA-based implementation. The primary data consisted of reads that were tested for performance speed on different FPGA platforms. The study highlighted BWA-MEM's capability to achieve significant speed-ups, especially in the seed extension phase, which is one of the most computationally intensive steps in the alignment process. The study's results indicated that while BWA-MEM is powerful, its effectiveness is closely tied to the hardware it operates on. The use of FPGA accelerators significantly enhanced its performance, showing that BWA-MEM's inherent computational intensity can be mitigated with specialized hardware. However, this implies that for standard CPU-based systems, BWA-MEM might be less suitable for large-scale analyses without hardware acceleration, potentially influencing the outcomes by requiring longer computation times or greater computational resources.

HISAT's hierarchical indexing strategy proved highly effective for aligning RNA-seq reads in dealing with splicing events. The study's focus on RNA-seq data, specifically with reads ranging from 75 to 150 bp, showed that HISAT could achieve high sensitivity and precision in detecting splice sites, as evidenced by the results in Table 1 and Figure 2. HISAT outperformed other mappers like TopHat2 and STAR in both alignment speed and accuracy. The choice of HISAT in this study was well-suited for the data type (RNA-seq), given its ability to efficiently manage the complexities of splicing. The use of hierarchical indexing reduced memory usage and improved cache performance, making it a robust choice for large-scale RNA-seq datasets. However, the study also revealed that HISAT's effectiveness decreases with reads longer than 150 bp, suggesting that its application is best limited to short-read RNA sequencing rather than longer genomic reads.

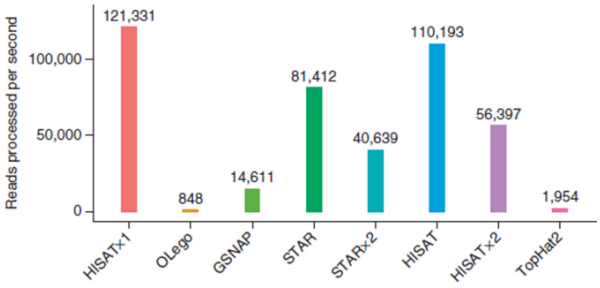

Figure 2.
Alignment speed of spliced alignment software for 20 million simulated 100-bp reads. Alignment speed for all read types (defined in Fig. 1) combined, measured as the number of reads processed per second by the indicated tools. supplementary Figure 2 provides the alignment speed for each type of read separately.

Table 1

Sensitivity and precision of leading spliced aligners

Program	no. of splice sites reported	no. of true splice sites reported	sensitivity (%)	Precision (%)
HISATx1	91,904	85,546	97.3	93.1
HISATx2	90,331	85,603	97.3	94.8
HISAT	90,300	85,587	97.3	94.8
STAR	95,892	84,678	96.3	88.3
STARx2	92,254	84,734	96.3	91.8
GSNAP	92,547	85,598	97.3	92.5
OLego	86,779	82,879	94.2	95.5
TopHat2	96,474	79,705	90.6	82.6

Sensitivity and precision of leading spliced aligners for 87,944 true splice sites contained in 20 million simulated reads from the human genome, with a mismatch rate of 0.5%. Sensitivity is the percentage of true splice sites found out of the total that were present. Precision (or positive predictive value) is the percentage of reported splice sites that are correct.

Chakraborty et al., (2021) demonstrated that S-conLSH is highly effective in mapping reads with a high error rate, as it uses a locality-sensitive hashing (LSH) approach to handle the inherent noise and indels in long reads. The mapping strategy focused on reference genome indexing and alignment-free mapping, which allowed S-conLSH to outperform other mappers like lordFAST and Minimap2 in terms of alignment accuracy, as shown in Table 4. The study suggests that S-conLSH's strength lies in its ability to maintain high accuracy in noisy data environments, making it suitable for long-read technologies like SMRT. However, the high memory usage reported (around 13 GB) for indexing indicates that S-conLSH might not be the best choice for systems with limited memory resources. This trade-off between memory consumption and alignment accuracy implies that S-conLSH is best suited for projects where accuracy is critical and resources are available, rather than for general-purpose genomic studies.

Table 4 Comparative study of running time, percentage of reads aligned and coverage by different aligners for *H.sapiens-real* SMRT dataset of 23,235 reads

Mapper	Indexing time (s)	Mapping time (s)	% of reads aligned	Mean coverage
Minimap	140	30	94.8	NA
Minimap2	138	106	100	0.0473
lordFAST	2286	327	100	0.0566
conLSH	47	404	99	0.0579
S-conLSH	794	99	99.9	0.078

Conclusion (10 marks)

In this comparative analysis of BWA-MEM, HISAT, and S-conLSH, each read-mapper has demonstrated unique strengths and weaknesses in handling various genomic and transcriptomic data types. BWA-MEM, leveraging the Smith-Waterman algorithm, excels in aligning reads with small indels and provides robust performance for short-read sequencing. However, its computational intensity, particularly during the seed extension phase, necessitates hardware acceleration like FPGA to maintain efficiency, making it less practical for general-purpose computing environments without specialized hardware. HISAT, with its innovative hierarchical indexing strategy, is optimized for RNA-seq data, particularly in handling the complexities of splicing events. Its ability to efficiently align reads across exon-exon junctions while minimizing memory usage makes it a preferred choice for large-scale RNA-seq projects. However, its effectiveness diminishes with longer reads, limiting its application primarily to short-read RNA sequencing. S-conLSH, offers a novel approach to managing noisy long reads. Its alignment-free mapping strategy allows it to maintain high accuracy in challenging data environments, making it suitable for metagenomics and repetitive sequence regions. However, the high memory consumption required for indexing poses a limitation for broader use, suggesting that S-conLSH is best employed in projects where accuracy is paramount and sufficient computational resources are available. The implications of these findings for bioinformatics research are significant. The choice of read mapper can impact the outcomes of genomic studies, particularly in terms of computational efficiency, accuracy, and resource requirements In conclusion, the selection of a read-mapper should be carefully aligned with the specific requirements of the project, considering factors such as read length, data type, and available computational resources. As bioinformatics research continues to evolve, the development of more efficient and versatile mapping tools will be critical to addressing the growing complexity and scale of genomic data analysis.

References

Alser, M. *et al.* (2021) 'Technology dictates algorithms: Recent developments in read alignment', *Genome Biology*, 22(1). doi:10.1186/s13059-021-02443-7.

Chakraborty, A., Morgenstern, B. and Bandyopadhyay, S. (2021) 'S-conlsh: Alignment-free gapped mapping of noisy long reads', *BMC Bioinformatics*, 22(1). doi:10.1186/s12859-020-03918-3.

Kim, D., Langmead, B. and Salzberg, S.L. (2015) 'HISAT: A fast spliced aligner with low memory requirements', *Nature Methods*, 12(4), pp. 357–360. doi:10.1038/nmeth.3317.

Pham-Quoc, C., Kieu-Do, B. and Thinh, T.N. (2019) 'A high-performance fpga-based bwa-mem DNA sequence alignment', *Concurrency and Computation: Practice and Experience*, 33(2). doi:10.1002/cpe.5328.