Short-Read DNA Sequence Alignment using the Burrows-Wheeler Transform

Comprehensive Report—CS Senior Comprehensive Exercise 2023–24

Shreya Nair, Phi Rapacz, Serafin Patino

Jack Owens, Zev Goldhaber-Gordon, and A.J. Ristino

Advisor: Professor Layla Oesper

Carleton College

INTRODUCTION

Our project involves an in-depth investigation of the Burrows-Wheeler Transform (BWT), its associated structures and optimizations, its practical application to short-read sequence alignment, and the resultant ethical debates in genomics. In addition to this paper, we produced our implementation of a BWT-based short-read aligner, built in C, which is accessible through our GitHub repository. This paper presents our findings surrounding the BWT and its algorithmic function, inclusion, and application in short-read aligners. Next, we will discuss the BWT's inclusion in genomics and document our methodology in developing a BWT-based short-read-aligner. Lastly, we put forward a review of the ethical issues in genomics unearthed by the BWT and provide our own outlook and project conclusion.

ALGORITHMS

A. BWT

The base Burrows-Wheeler Transform (BWT) is a cyclical string-transformation algorithm. Originally intended to ease the process of compressing significant texts, the BWT outputs transformed versions of inputs containing long sequences (or 'runs') of similar characters. When encoding, a sentinel (or terminating) character is first added to the end of the string. The character '$' is commonly used because computers interpret it alphabetically earlier than other letters and because of its comparative rarity in texts. We then take our input string and rotate it to the right by one character n times (where n is the number of characters in the input), generating all cyclical permutations of the string. After generation, we lexicographically sort all our permutations, taking advantage of the '$' character's alphabetical priority. Lastly, we record

the final column of our lexicographically ordered and permuted strings, obtaining our encoded input (Fig. 1).

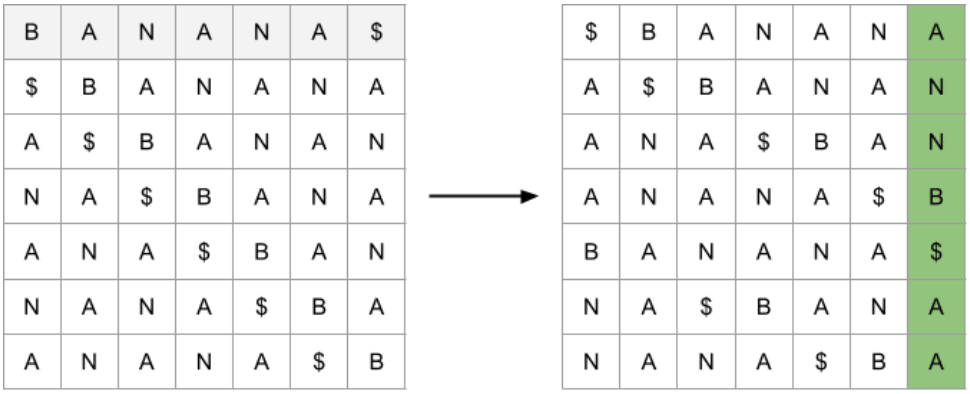
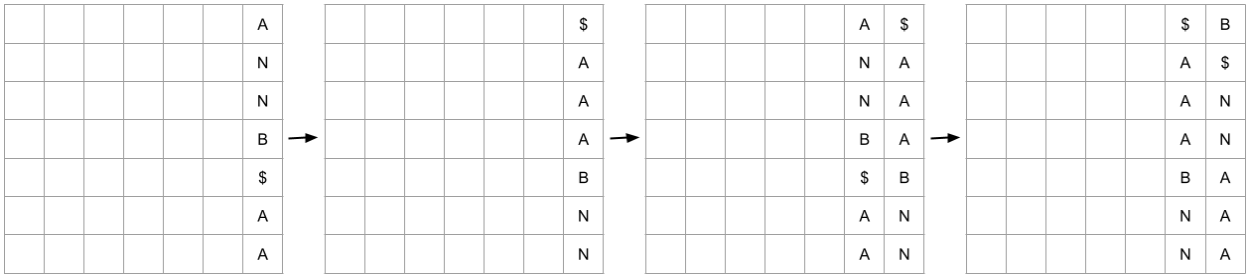


Figure 1. Grid of all rotated string permutations, rows unsorted (left); grid of all permutations sorted lexicographically (right).

One of the advantages of BWT encoding is its reversibility. There are two ways to decode a transformed string. The first is the reconstruction of the encoding grid, as displayed in Fig. 1, column by column from the last to the first. We fill the last column with the transformed string, and the rows are re-sorted into lexicographical order. Then, the second to last column is filled in with the transformed string, and the rows are sorted again. This continues until the table has been filled. At this point, sort the rows one final time. To identify the original string, you return the row ending in '$' (Fig. 2).



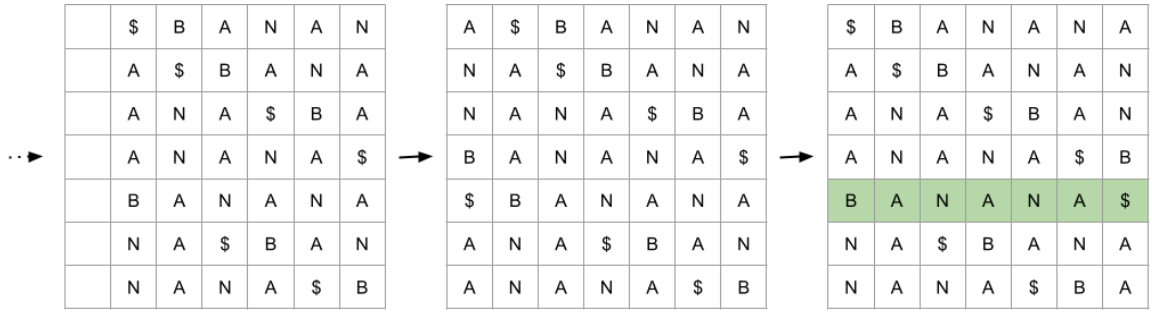


Figure 2. The process of reconstruction of the grid, adding the BWT string column by column and re-sorting rows after each addition. The reconstructed string is shown in green, with the terminator character now at the beginning of the string rather than the end.

The second method of decoding a BWT codeword is using the string's FM Index. Since we sort the string lexicographically, the transformed string can produce two strings immediately with no additional sorting or calculation: the codeword and a string with the character's characters sorted into lexicographical order; the last and first columns of the rightmost grid in Fig. 1. With the FM Index of each character (discussed in the following section), we can tie the unique instance of each character from one column to the other. Starting with the sentinel character, we reconstruct the original string by jumping from the last to the first column (Fig. 3).

Figure 3. The transform grid with only the first and last column information (left); the first and last column characters indexed for tracing (right); the character indexed reconstructed string, backwards and forwards (middle).

The BWT is not a compression algorithm in and of itself but rather a compression-preparation tool. As mentioned earlier, the BWT's outputs tend to have long runs of characters, which means they lend themselves well to compression using Run-Length Encoding (RLE) (Golomb et al., 1966). RLE is a compression method in which a string's repeated characters are combined into a single instance of the character, followed by the number of subsequent duplicates (e.g., 'AAACCTGGG' would become 'A3C2T1G3').

RLE does not save significant space when a string is short or the characters are randomized; however, when used on larger quantities of data, the combination of BWT and RLE significantly reduces the required memory to store a text. The compressibility of a string transformed via BWT can be determined by the number and length of contiguous character sequences it contains. Each contiguous character in a sequence past the second is an additional character compressed by RLE. An excellent example of a practical application of the BWT is on DNA sequences.

The number of characters required to store the information of nucleotide sequences is relatively small, consisting of A, C, T, and G and a single instance of the sentinel character. Due to this, the possible number of character subsequences is also significantly less than a typical alphabet; a given subsequence will generally recur far more frequently. This makes the BWT exceptional for compressing and storing DNA sequences and constructing their FM Index, the data structure upon which modern sequence alignment is built.

B. FM Index

The Ferragina-Manzini (FM) Index, or Full-text in Minute Index, is a data structure originally developed to efficiently store and produce the suffix array of a string (Ferragina &

Manzini, 2002). The FM Index construction relies heavily on the BWT of a string. A traditional suffix array is a list of all possible suffixes of a string that's overlaid on the tables of the BWT (Fig. 4).

B	A	N	A	N	A	$
$	B	A	N	A	N	A
A	$	B	A	N	A	N
N	A	$	B	A	N	A
A	N	A	$	B	A	N
N	A	N	A	$	B	A
A	N	A	N	A	$	B

$	B	A	N	A	N	A
A	$	B	A	N	A	N
A	N	A	$	B	A	N
A	N	A	N	A	$	B
B	A	N	A	N	A	$
N	A	$	B	A	N	A
N	A	N	A	$	B	A

Figure 4. The suffix arrays as represented by the BWT, in length order (left) and lexicographical order (right).

Traditionally, complete suffix arrays take a substantial amount of memory to store and are inefficient to query. The FM Index (Fig. 5) takes advantage of the BWT's last-first-order property to address these shortcomings. An FM Index is made from the first and last columns of the ordered BWT of an input string (the rightmost table in Fig. 4) in addition to an attached character tally matrix and two smaller arrays: one of total count by character and the other of first column index offset—or, index of first appearance—by character.

idx	*og*	*F*	*L*	$	A	B	N
0	*6*	$	A	0	1	0	0
1	*5*	A	N	0	1	0	1
2	*3*	A	N	0	1	0	2
3	*1*	A	B	0	1	1	2
4	*0*	B	$	1	1	1	2
5	*4*	N	A	1	2	1	2
6	*2*	N	A	1	3	1	2

	$	A	B	N
Char count	1	3	1	2
Index offset	0	1	4	5

Figure 5. The FM Index data structure of the word “BANANA$”. ‘idx’ = “index”, ‘og’ = “original string index.”

In addition to being fast to query, the FM Index can be searched character-by-character for matches with an unprocessed substring (Fig. 6). The alignment of a substring to the pre-processed string via the FM Index progresses “backward” for the original string:

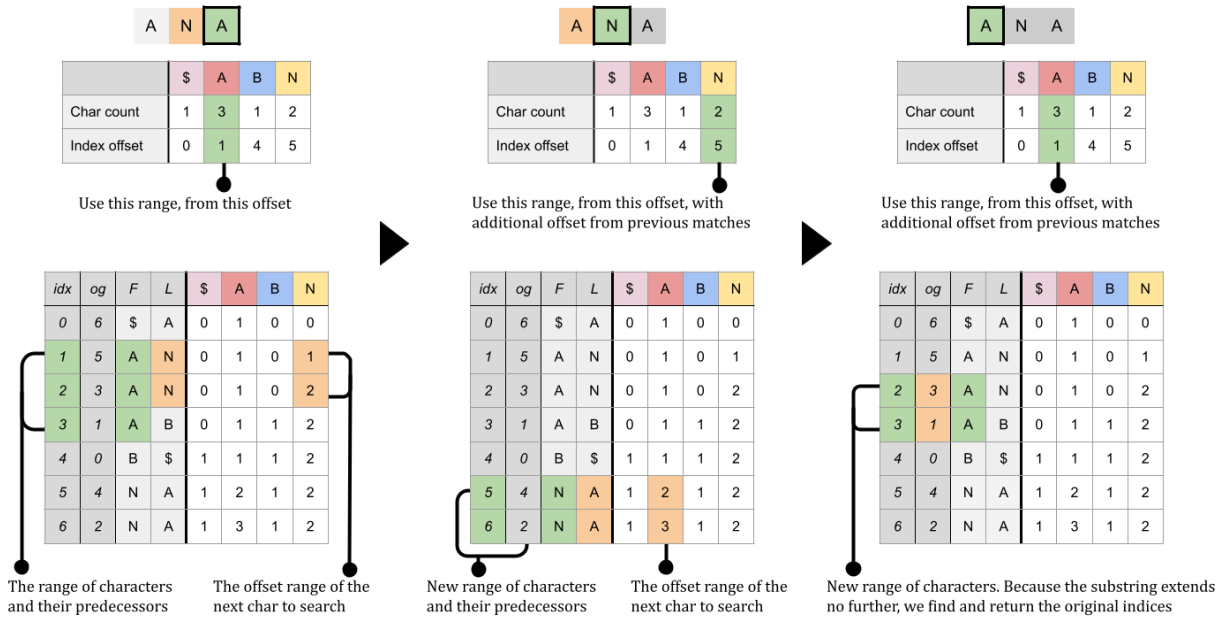


Figure 6. An example search of the FM Index of the string “BANANA$” for the substring “ANA.”

Returns two hits, at indices 1 and 3 of the original string.

Note that this navigation, unaugmented, will find only exact matches between the string and substring. If there are no matches to the next character within the specified range (See Fig. 6), this algorithm will interpret a lack of matches as a failure. This is undoubtedly useful in some applications but is detrimental in DNA sequencing, where the vast majority of valuable alignments will be inexact. As a result, any contemporary aligner that employs the FM Index will employ an augmented search algorithm that allows for inexact matching, often via scoring, such

as the nucleotide-substitution-tolerant backtracking algorithm employed by the BWT-based alignment tool Bowtie (Langmead et al., 2009).

C. Smith-Waterman and Gapped Alignment

While alignment via an FM Index is incredibly efficient, allowing a template string to be searched from multiple locations and returning all matching indices, it has the disadvantage of no gap tolerance. That is to say, it will find only exact matches without backtracking and only matches of the same periodicity with backtracking. This means that, in biological terms, it is tolerant of only one of three possible DNA replication errors: substitution, when a nucleotide swaps out for another. Even backtracking gives preference to results that begin well-matched even if they end poorly matched, as the algorithm only begins to backtrack when it finds a dead end.

The two remaining DNA replication errors, deletion and insertion (when a subsequence of any size is either deleted or inserted), are handled well by dynamic programming matrix scoring like the Needleman-Wunsch algorithm (Needleman & Wunsch, 1970) and its variation of the Smith-Waterman algorithm (Smith & Waterman, 1981). These algorithms function by creating a matrix of dimensions equal to the template length and the query length and scoring every possible alignment between every possible pair of characters, leaving the algorithm with an order of O(n2). Neeldeman-Wunsch allows for negative matrix cell values and is used for global alignment (determining the similarity of two sequences of comparable length). In contrast, Smith-Waterman only allows for cell values zero or greater and is used for local alignment (where a subsequence fits in a more extended sequence).

The score for an alignment is calculated from the top left of the matrix. Scores increase or decrease on diagonals according to a scoring matrix, which is a guide as to the character match bonus and penalties for each possible character mismatch (Fig. 7).

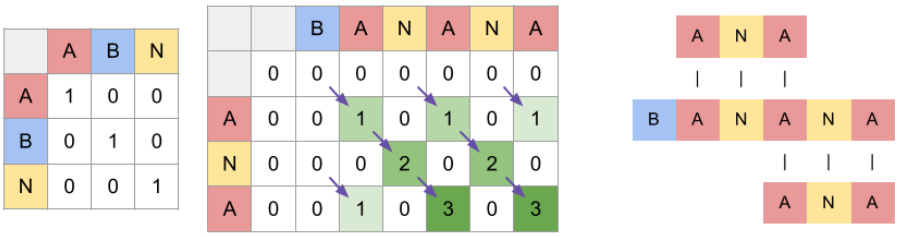


Figure 7. The SW matrix for the alignment of 'ANA' to 'BANANA.' The scoring matrix (left), the alignment matrix (middle), and the possible alignments (right).

The algorithm chooses the highest-scoring cell of the alignment matrix to find the best alignment once all have been computed. It traces the alignment backward, taking the path of the highest-scoring connected cells. Progression is preferentially diagonal up-left, as a perfect match would be an uninterrupted up-left diagonal path. When a mismatch occurs, a path progressing directly up or to the left signifies a gapped alignment (Fig. 8).

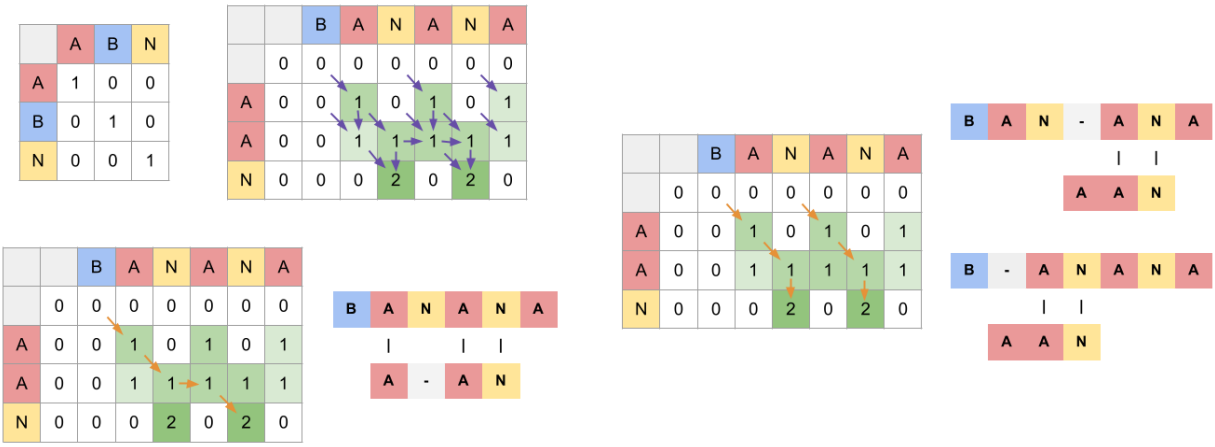
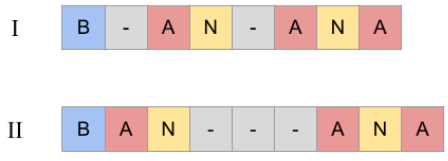


Figure 8. The SW alignment of 'AAN' to 'BANANA.' Hypothetical result paths are highlighted with orange arrows. Due to diagonal traversal preference, the path and result on the right are disincentivized.

Gaps also factor into scoring, and there are several approaches to alignment gap penalties. The calculation of a gap penalty can define which alignment an implementation considers more favorable (Fig. 9). The Constant gap penalty will penalize a constant amount for each gap that occurs (how many times in an alignment you must jump, regardless of distance); a Linear gap penalty will penalize an alignment proportionally to the length of the gap (how many columns or rows an alignment has to jump); the Affine gap penalty is a combination of the two, each gap incurring a flat opening penalty and scaling linearly as the length of the gap increases. Thus, longer gaps incur a more significant penalty but a lower penalty in proportion to their length than shorter gaps.



Penalty Type	equation	Preferred alignment
Constant	P = Constant	II
Linear	P = Coefficient × Length	I
Affine	P = Constant + (Coefficient × Length)	II

Figure 9. Two hypothetical alignments of the same substring (left) and gap penalty calculations that favor each (right).

Given the scoring criteria selected, this approach is guaranteed to find the best possible local alignment; however, it requires a significant amount of time and memory. An area of improvement noted and expanded upon by Lam et al. was that this algorithm, while thorough, will perform the calculations fresh for all identical substrings in the reference string (Lam et al., 2008). For example, in Fig. 7, while two locations of the identical score are found (Fig. 10), the algorithm would have calculated both from scratch. It could have halved its time if it had applied what it had done the first time to the new identical substring.

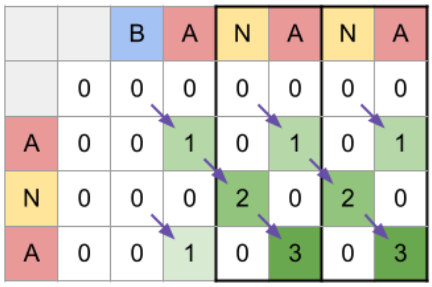


Figure 10. The redundant matrix calculations are outlined.

This is where the BWT and the FM Index once again prove helpful. Due to the FM Index's functionality, it is possible to pinpoint identical substrings of the reference string and avoid computing score calculations for similarity fresh again. The improvements, like the BWT's use in RLE compression, are only substantial on much longer strings when the time saved offsets the overhead—but when DNA sequences can be billions of nucleotides long, the overhead remains minuscule.

D. Optimizations

Many optimizations can be employed in an actual BWT-based alignment implementation. There is, for example, no need to store the entire transformation grid as a matrix; instead, it is a single array of index pointers to represent the starting character of each permutation and, therefore, each row of the grid (Fig. 11).

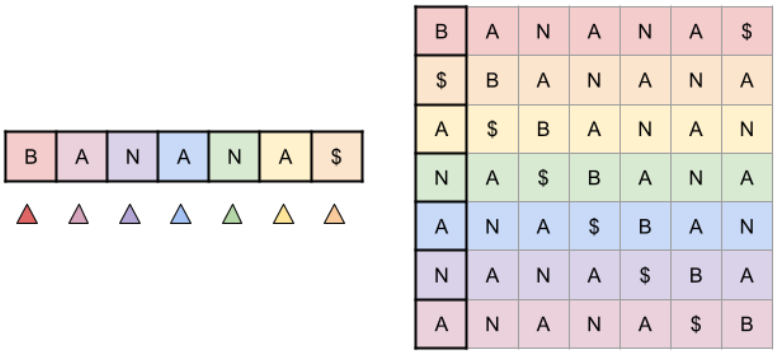


Figure 11. Indices color-coded to the row of the conceptual BWT grid they represent.

Another optimization applies to lexicographical ordering, which, despite being foundational to the BWT, can slow the entire algorithm down, depending on the algorithm chosen for character sorting. Our implementation used a combination of Radix Sort and Quicksort, which, due to index-based optimizations discussed above, was already considerably space-optimized.

Radix sort uses a structure similar to the FM Index, with a character count and index offset structure (Fig. 12).

B	A	N	A	N	A	$
0	1	2	3	4	5	6

	$	A	B	N
Char count	1	3	1	2
Index offset	0	1	4	5

Figure 12. The beginning of the Radix structure. The indexed string (left) and the count and offset table (right).

An array of index pointers the length of the string is constructed and divided into ‘buckets’ (groupings) of appropriate size based on the character count data. The index of the original string is sorted into this array by the next character (Fig. 13).

B	A	N	A	N	A	$
0	1	2	3	4	5	6

	$	A	B	N
Char count	1	3	1	2
Index offset	0	1	4	5

Bucket	$	A			B	N	
Index	5	0	2	4	6	1	3

Figure 13. String indices are sorted into the buckets based not on character, but the character of the next index. For example, while ‘$’ sits at index 6, the next character is ‘B’ and so index 6 is placed in the bucket ‘B’.

After sorting the indices by the next character, they are sorted again by the first character into another array of buckets, proceeding by index as ordered in the first bucket array (Fig. 14). Because we perform two rounds of Radix, this element is O(n).

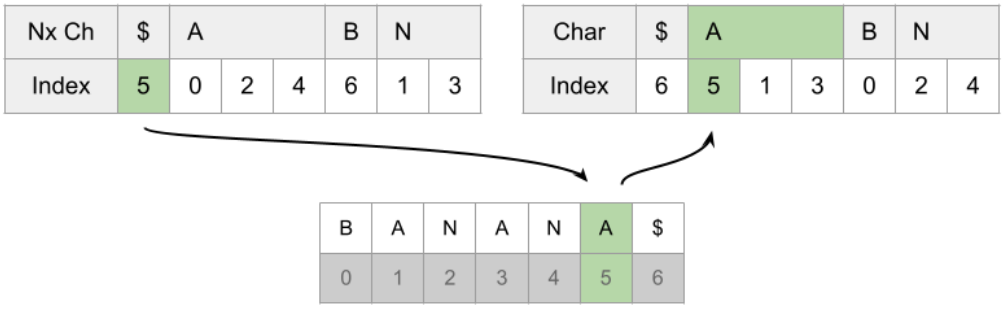


Figure 14. The second index array, with the first placement calculator highlighted in green.

By sorting first by the next character and then by the first character, the indices have been sorted into buckets by the first two characters (Fig. 15).

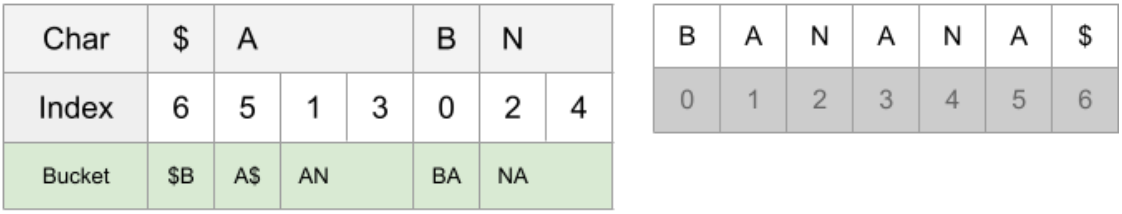


Figure 15. The two character buckets are highlighted in green.

Each bucket containing more than one index must be sorted further by the characters after the first two using Quicksort. This leaves the algorithm with a runtime of O(n log n) like Quicksort itself, though, due to the two rounds of Radix at O(n), it is a comparatively smaller O(n log n).

BIOLOGY

DNA sequencing built the basis of the biological revolution over the last 40 years, which allows us to take biological samples and extract their genetic composition. DNA (deoxyribonucleic acid) is a helix-shaped molecule that stores the genetic information of each organism, composed of two linked strands made up of a sugar-phosphate backbone connected by complementary nitrogenous bases called nucleotides.

These molecules, also known as bases, come in four variants: Adenine (A), Thymine (T), Guanine (G), and Cytosine (C). Each base has a complementary base with A, T, and C bonds with G. These 'base pairs' are held together by hydrogen bonds, forming the 50 – 300 million 'rung' shapes in a human DNA strand.

Understanding these long strands of DNA and how to interpret them is a foundational element of understanding how our world and its living organisms have come to be. The information gleaned from DNA sequencing informs the construction of genomes of many organisms (including previously unknown ones), the diagnosis of genetic diseases, genome resequencing, and much more (Shendure et al., 2017).

The development of Next-Gen sequencing (NGS) was integral to unraveling the mysteries of DNA across life. The NGS process uses short reads, which are short fragments of DNA that come from the genome(s) present in a sample. The short reads get assembled, often using a reference genome.

A famous example of NGS in use, Illumina (an applied genomic technology company), has led the charge, applying sequencing by synthesis. Illumina's process first creates a 'library' consisting of DNA broken into fragments of ~200 - 500 base pairs in length; the double-stranded structure is unraveled into two single strands. Adapters are attached to the strands on either end

so that each strand can bind to a Flow Cell. A Flow Cell is where the central sequencing by synthesis process occurs. The DNA strands use the adapters to bind to the flat surface, and fluorescent-labeled bases (fluorescently-tagged A, C, T & G molecules) are then applied to the single-stranded fragment to identify the input sequence of the DNA. These fluorescent-tagged fragments are called 'short reads.'

When considering genome assembly, we face the challenge of aligning millions, if not billions, of short reads to a given reference model. Alignment refers to identifying the correct position of a sequenced short read. The alignment process for short sequences is both computationally taxing and resource-intensive, a technological hurdle researchers have tried to solve for years. This is where the BWT's mathematical properties and biotechnology intersect. A simple formulation of the BWT's purpose is to manipulate a string and arrange it to be in a better form for compression. An emergent practice in bioinformatics uses the short reads and reference genome as inputs into the BWT, reducing the computational burden that alignment poses.

Short-read alignment algorithms have used a single reference genome for the most part. However, genomes within species can have a high amount of variation. The human genome, for instance, differs by 0.1% between any two individuals (the genome is 3 billion base pairs long). In light of this difference, which can lead to many cases of bias (see ethics section), the field has adopted the concept of a 'pangenome,' a collective "whole-genome sequences of multiple individuals representing the genetic diversity of the species" (Wang et al., 2022). By combining different whole genomes to create a hybrid version that can account for significantly more variation, aligners can more accurately match reads to correct locations within a reference. While this may be computationally slower initially, matching against a pangenome is much more efficient than against multiple references. The pangenome project was primarily headed by the

Human Pangenome Reference Consortium (HPRC), aiming to create a human reference genome that accurately represents genetic diversity across humans.

Conversely, ‘polygenome’ refers to the hybridization of a reference genome to more accurately represent a species’ variation. One or more genomes are put together to try and reconcile or offer more options for the common variation sites or Single Nucleotide Variants (SNVs, sequence locations where different individuals may have different nucleotides; for example, individual 1 has an A at position 10445, but individual 2 has a G at the same position 10445). In the same vein, Polygenomes are another method used to reduce the effect of bias as we try to incorporate different genomes from different ethnicities.

LITERATURE REVIEW

2009 was a landmark year for genomic applications of the BWT in creating short-read alignment tools, including the software package Bowtie and the Burrows-Wheeler Alignment Tool (BWA). These two packages use the BWT to speed up alignment and drastically reduce memory and computational requirements.

Bowtie was developed by Langmead et al. at the University of Maryland. The software can align over 25 million short DNA reads per CPU hour with a memory footprint of approximately 1.3 gigabytes. This is nearly 35 times faster than MAQ (the previously most popular aligner in the field). One of Bowtie’s key features is its ability to deal with mismatches in input sequences using the BWT’s Last-First ordering properties (Langmead et al., 2009), described further in the algorithm section.

Following Bowtie, the Burrows-Wheeler Alignment tool (BWA) was created by Li & Durbin at the Wellcome Trust Sanger Institute. BWA innovates upon previous BWT approaches

by combining the FM Index with backward searching, lowering string-matching query times to sublinear speeds. Much like Bowtie, BWA can handle mismatches but allows for other types of genome mapping beyond allowing for paired-end and gapped alignment mapping. Additionally, BWA's output format is the.SAM file (a text file format containing alignment sequence info) is backward-compatible with commonly used genomics software for added user-friendliness. (Li & Durbin, 2009).

Independent of BWT-based aligner development, GPU `ization and further processor acceleration have sped up BWT processing times even further (Sheng et al., 2021; Kähkönen et al., 2023). BWT's primary time-consuming step is the initial sorting of each string permutation. This can be condensed using a Radix sorting algorithm, which sorts the first two characters before using a conventional quicksort on the remaining characters. This approach is perfect for DNA, as all strings are permutations of the same letters. Given this new method, GPUs can efficiently execute this Radix sort on data above ~25 MByte in size, sorting ~1.61 GBytes/second. An even faster algorithm that treats the permutations as suffixes called SAIS can operate in O(N) time.

A challenge for alignment, however, has been reconciling the differences between individuals and the variance between reference genomes (see biology section). BWT's enhancement of NGS processes has informed proposals to replace standard genome models with poly-or even-pan genomes, which address the issue of cross-genome variation. BWBBLE, a short read aligner created by Lin Huang, Victoria Popic, and Serafim Batzoglou, which aligns the query string to multiple genomes, is an example of circumventing reference genome bias (see ethics section). Although BWBBLE's approach is many orders of magnitude slower than the other versions we have seen (BWA and Bowtie), it is more efficient than attempting to run each

aligner multiple times with a different reference genome each time. The accuracy and confidence will be higher than a single run from BWA or Bowtie, which is extremely important when drawing conclusions that have significant social implications. Due to these advantages, multi-genomes seem to be a possible future for most short-read aligners to turn to once there is a more comprehensive collection of various reference genomes.

IMPLEMENTATION & EXPERIMENTS

A. BWT Benchmarking

We implemented and benchmarked five Burrows-Wheeler Transform programs and three query alignment algorithms in both C and Python. For those written in Python, we analyzed these programs with the benchmarking library timeit, while in C, we manually recorded the computer's timestamps before and after running the program. We performed benchmarking using 20 runs of 50 repetitions each. We took the average runtime over 50 repetitions for each run, and the minimum run average time across all 20 runs was our output because benchmarking attempts to measure the speed of a program in optimal conditions since programs can be slowed down by background activity but never sped up by it. Benchmarks were performed on randomly generated DNA sequences of varying lengths, and runtime was graphed on a log-log scale versus the generated string length.

B. Implementations

Our Base BWT Transform was done in Python. To conduct the transform, it stores all rotations of the reference text and sorts them using Python's built-in sort() method.

Base BWT Querying was our base pattern alignment. We implemented it in Python. It creates two arrays to represent the first and last columns of the rotation array for an encoded

reference text and creates a dictionary to map each character from the last column to the first column. Using the dictionary, it traces a query backward through the columns until it finds all its matches.

NumPy BWT Transform was the second transform implemented in Python and uses the NumPy Python library to perform the rotations and sorting to transform the reference text.

C BWT Transform Quicksort was our first C implementation. It stores the rotations and quicksorts them to execute the transform.

C BWT Transform Radix was our second transform implementation in C. It performs a Radix sort on the first two characters of each rotation and quicksorts the rest.

BWT Transform SAIS was the most efficient transform algorithm we implemented. It was done in Python and uses a suffix array to represent and sort the rotations of the reference text.

Boyer-Moore Querying was a Python implementation of the pattern alignment algorithm used in CTRL+F or Command+F. It preprocesses a query string to skip as many alignments as possible. We implemented it so that we could compare it to BWT alignment.

Naive Querying was the most straightforward pattern alignment algorithm and was implemented to compare it to BWT. It naively checks all possible alignments for a query to the reference text. Like the other alignment programs, we also implemented it in Python.

C. BWT Transform

As seen in the supplementary material, a Radix sort that does not copy memory can reduce the sorting time of the Burrows-Wheeler transform, although it will still be in the same order of time requirement, O(Nlog(N)). To achieve this level of memory control, we implemented our methods in C. The method we eventually used could be just as easily

implemented in Python without copying memory, although it would have been far slower than either C implementation. We chose to use Radix sort on the first two characters of each rotation, then switch to quicksort for the remaining characters. Instead of fully expanding the Burrows-Wheeler Matrix into memory, we represented it as a suffix array, directly indexing it into the original string for the character-to-character comparison function in Radix (See Fig. 16).

Similarly, instead of having separate arrays for each pair of first characters of the string after Radix sort bucketing, we use a counter array to generate an offset array. The offset array splits the more extended suffix array into starting characters (See Fig. 17). We run two iterations of Radix sort: first generating a suffix array sorted by the second character of the string, then using that suffix array as an order to populate a suffix array sorted by both the first and second characters. Once the Radix sort is complete, any remaining unsorted suffixes get quicksorted. A runtime analysis for varying numbers of Radix sorts is included in the supplemental materials. However, we chose to do two iterations since it matches pairs of first and last characters and is thus helpful for later constructing the FM-Index.

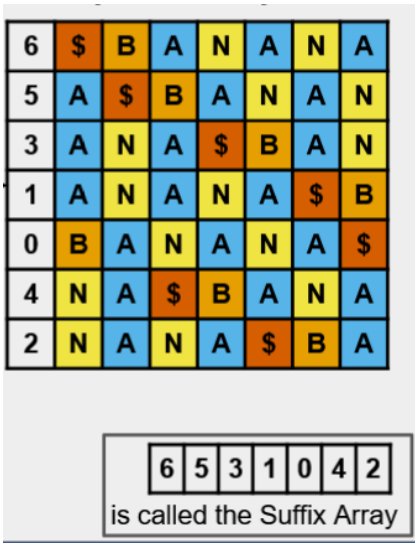


Figure 16. A BWT matrix, and the corresponding suffix array. Same construction as in Fig 4.

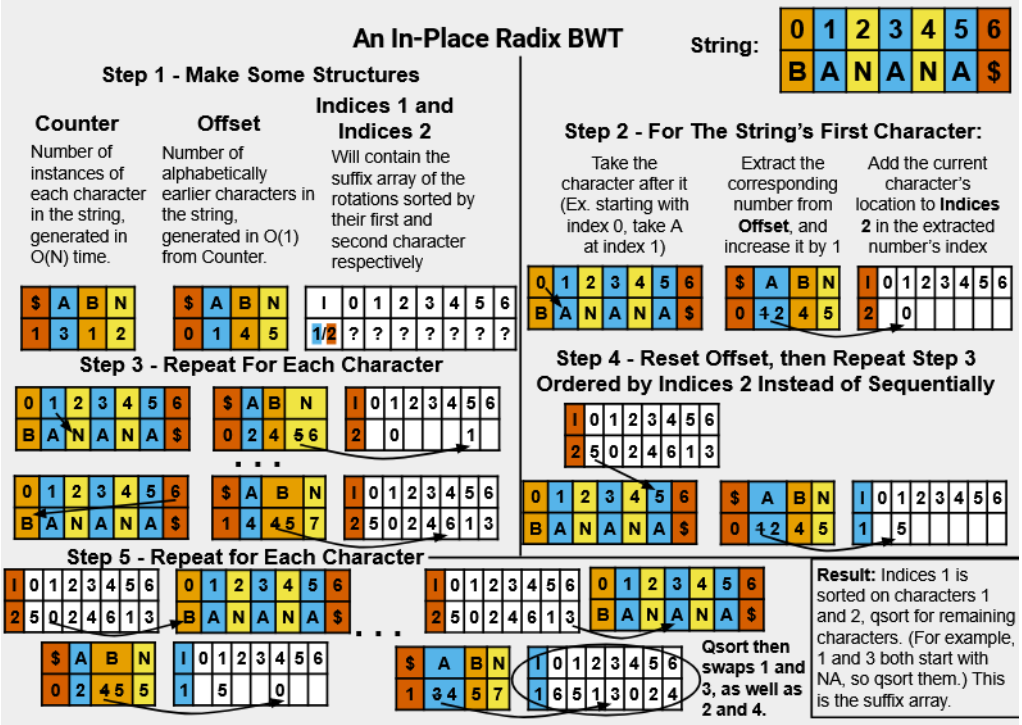


Figure 17. A breakdown of the full Radix+Quicksort process, as coded.

While the Radix sort BWT is our fastest implementation, it is still O(Nlog(N)) time growth, as discussed in Algorithms. Another method for determining the BWT in O(N) is the Suffix Array Induced Sorting (SAIS) algorithm. Much like our Radix implementation, it uses a suffix array instead of a matrix to represent the potential string rotations. We will not describe the entire algorithm, but the basics represent each rotation as S-type or L-type, where S-type rotations are alphabetically earlier than the next rotation, and L-type rotations are later than the next one. S*-type rotations are S-type rotations for which the previous rotation is L-type (See Fig. 18).

M	M	I	S	S	I	S	S	I	I	P	P	I	I	$
L	L	S*	L	L	S*	L	L	S*	S	L	L	L	L	S

Figure 18. A rotation-type array for the string MMISSISSIIPPII

The induced sorting algorithm recursively generates the suffix arrays of the S*-substrings, which start and end with S* characters: in this case, ISSI, ISSI, IIPPII$, and $. We implemented an induced sorter in Python from scratch and benchmarked it alongside the other BWT implementations. While it is slower than C implementations, likely due to the higher overhead inherent to the algorithm and Python being slower than C, it also scales more favorably than the matrix-based Python implementations (See Fig. 19).

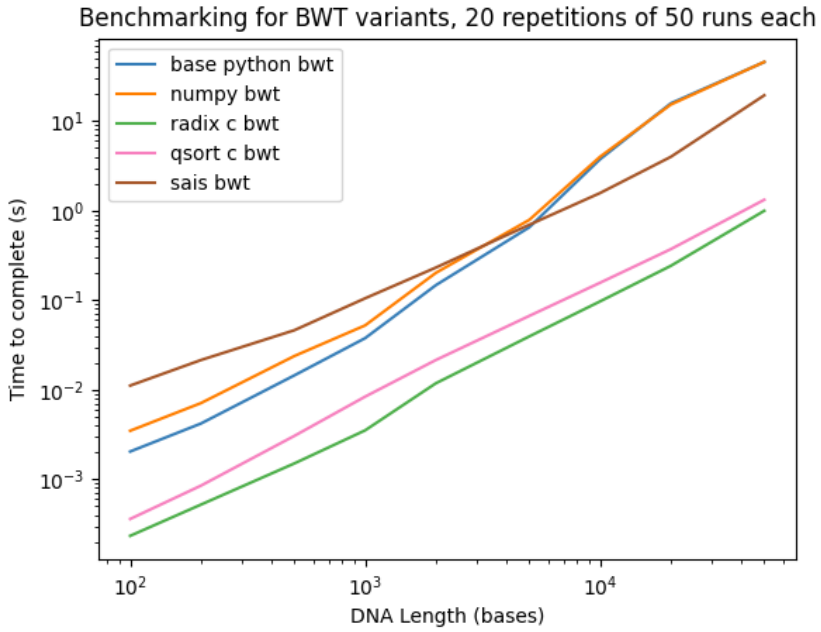


Figure 19. The runtimes of various algorithms on various lengths of input sequences, graphed on a log-log scale. Radix and SAIS appear as described. Qsort is a C quicksorter that also uses a suffix array, base python is a simple Python implementation, and numpy bwt is a Python implementation which utilizes the numpy vector math library to speed up some operations.

D. BWT Alignment

To benchmark alignment using our Python BWT alignment implementation against other alignment methods (See Fig. 20), we implemented two more matching algorithms in Python. The first was naive exact alignment, which checks every possible alignment the pattern could have. In the worst case, naive alignment is O(mn) time, where m is the length of the query and n is the length of the genome. In the best case, it is Ω(n) time. The second algorithm was the

Boyer-Moore alignment. Like naive matching, it traverses a searchable string from start to finish but relies on preprocessing the query to avoid unnecessary comparisons. The preprocessing step takes O(m) time, and the queries take O(mn) time, the same as naive matching. However, the queries are Ω(n/m) in the best case, and the average case is faster as the length of the query m increases.

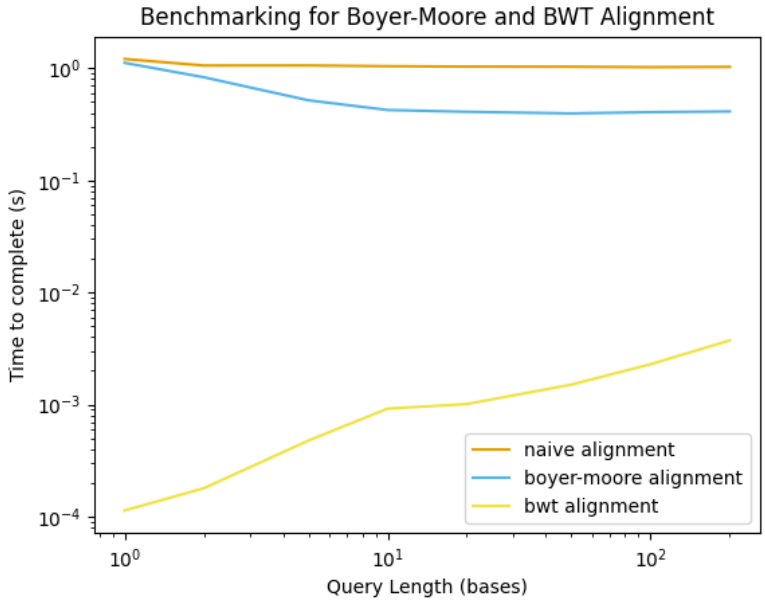


Figure 20. A benchmark comparison of naive alignment, Boyer-Moore, and BWT alignment on query strings

Our C implementation of BWT alignment (not benchmarked) involved three custom structs to store our data. We made an array that stores pairs of numbers to represent the encoded letter of the current character and its number of occurrences so far in the reference text. We also made a 2D array to store the indices where each specific character occurs. Then, we utilized two of these 2D arrays, one for the first column and one for the last column of the rotation matrix. Like the int pair array, each dimension of the 2D array represents the character value and its occurrence. Last, we created a struct to store all these arrays in the same place.

Character	Index		
	1st Occurrence	2nd Occurrence	3rd Occurrence
A	2	4	6
B	1		
N	3	5	

Figure 21. 2D occurrence array for the string BANANA

Once the transform and querying were functional, we expanded our C program to include a user interface in the terminal. The interface prompts the user to enter the name of a text file that contains the reference genome. Once the reference genome gets read, the transform gets performed, and the program prompts the user to enter a query. Typically, entering a query will return the total number of matches it made to the reference genome, and the interface will also display them if the genome is short enough. The program can also output the specific indices in the reference text where matches are. DNA short-read sequencing data gets collected in fastq files, so the interface allows the user to read queries from a .fastq.

Lastly, we produced a simplistic yet effective GUI built in Python. The GUI is a project introduction and overview intended to draw a general audience. Modules used in producing the GUI include Turtles for graphics and OS to sync up the GUI's working and looking directory. The GUI consists of nine slides, including a title screen and table of contents. The GUI contains limited intractability but has enough functionality to allow for repeat viewing of different screens. For more information, visit our project website and click the link on the homepage for a guided video introduction and demonstration.

ETHICS

In this section we seek to address the ongoing debates surrounding information sharing, informed consent, and equity in DNA sampling. We will split the four primary points of discussion into the following subsections:

a. Information sharing
b. Informed consent
c. Equity
d. Outlook

This section offers a discussion of areas of emergent ethical contention in genomics. Our goal with this non-scientific section is to raise awareness about ongoing debates and proposed solutions.

A. Information sharing

BWT-based read-aligners were responsible for the drastic reduction in the data footprint and computational cost required to store the entirety of the human genome, according to Langmead et. Al “For the human genome, Burrows-Wheeler indexing allows Bowtie to align more than 25 million reads per CPU hour with a memory footprint of approximately 1.3 GB.” This reduction allowed the field to prosper, and with the advent of modern computers, sharing the entirety of a human genome over an internet connection is remarkably easy.

This newfound efficiency can be linked to the rise of consumer genetics services, such as 23andMe. The emergence of this new industry has raised concerns regarding how easily someone should be able to access their genetic information or whether they should be able to

access it at all. Multiple projects have suggested ethical frameworks to regulate access to genomic information, including the previously mentioned HPRC.

The potential mishandling of genomic data from improper regulation could expose entire communities to risk. As written by Kaye, Jane, et al. in a research paper titled "Ethical Implications of the Use of Whole Genome Methods in Medical Research." in the European Journal of Human Genetics: "Cheaper sequencing techniques have enabled commercial companies to make genomic information available to the public over the internet… Linking these sources to other information readily available on the internet, such as births, deaths, and marriage records held by government bodies, provides the means to make information previously thought to be non-identifying, potentially identifiable." (Privacy of Research Participants) Consumers' privacy could be breached when someone can web scrape publicly available genomic data from commercial companies to identify individuals. Additional concerns arise from the release of genetic information and the resultant prospect of the public attempting to interpret or use their genomic data. Without proper training or context, this could lead to unnecessary and expensive medical testing that wastes valuable time and resources in medical fields.

B. Informed consent

A core ethical debate in genomics is centered on employing traditional clinical informed consent in studies and research concerning genetics. McGuire et al. emphasize in their paper the challenge of an individual providing consent without fully comprehending the intricate details, especially given the complexities of genomic research. Soon after, in 2009, Kaye, Jane et al. noted the escalating difficulty of ensuring truly informed consent as research techniques advanced. Even before both of these publications, the SACGHS' 2003 framework sought to

address some of these concerns, proposing guidelines to maintain the clinical relevance of genetic data and ensure that genetic information is updated as new knowledge emerges. W With the rise of consumer genomics, heightened risks associated with data misuse have added a new dimension to the informed consent debate, as reported by Kaye, Jane, et al. in a research paper titled “Ethical Implications of the Use of Whole Genome Methods in Medical Research.” in the European Journal of Human Genetics Companies like 23andWe (23andMe’s research division) have been proven to repurpose data for additional studies, without participants’ knowledge. Similarly, McGuire articulated concerns about using participant data for disparate purposes, including studying personality traits or behaviors. In 2022, the HPRC prioritized participant consent in its research framework. Their plans prioritize data de-identification and efforts to increase diversity in genomic studies. Despite the HPRC’s dedication to participant protection, the genomics community still grapples with these ethical debates, indicating a crucial need for consensus as the field evolves.

C. Equity

The equity issues go hand-in-hand with the challenges posed to genomics by consent, specifically regarding representation in genomic models. The most common human genome model currently used by short-read aligners is GRCh38 (and sometimes its older counterpart GRCh37), which is found to be at best, biased and, at worst, inherently incomplete. The researchers involved in the HPRC discussed this in their Embedded ELSI (Ethical, Legal, and Social Implications) Study section: “Most human genomics has been based on individuals of European ancestry, and the datasets available for analysis are thus biased. As a result, current precision medicine is based on genomic variation found in populations with primarily European

ancestry." The HPRC argues that GRCh38 should be dropped in favor of the pangenome as: "...[GRCh38] contains biases and errors within a framework that does not represent global human genomic variation. A high-quality reference with a global representation of common variants, including single-nucleotide variants, structural variants, and functional elements, is needed." (Wang et al., 2022). An additional reason to drop GRCh38 is the impossibility of tracking structural variants: "Both the limitations of short reads and reference biases mean that we may have missed more than 70% of structural variants in traditional whole-genome sequencing studies." (Wang et al., 2022)

The HPRC has dedicated itself to fostering an international community for the pangenome reference alignment to engage the diverse populations represented by the genome actively. As a first step, "The HPRC selected individual genomes for high-quality sequencing among existing cell lines established by the 1000 Genomes Project, which offers a deep catalog of human variation from 26 populations. These cells were originally collected from volunteer donors using consent procedures designed for unrestricted data use, and the cell lines are available in the Nation Genome Research Institute Biorepository…" (Inclusion Criteria) In addition to genomes acquired from external sources such as the BioMe Biobank at Mount Sinai and additional volunteers to aid in tracking down variants, these form a far more diverse baseline for the polygenome model.

In a very literal sense, the HPRC seeks to rectify the ethical shortcomings of genomics, advocating for inclusivity and equity at all levels of development and implementation. As stated at the end of the Embedded ELSI (Ethical, Legal, and Social Implications) Study section: "In developing a state-of-the-art pangenome reference sequence, the HPRC will continue to disseminate standards for accuracy and completeness of sequencing, as well as emphasizing the

importance of ELSI considerations." (Wang et al., 2022). Pursuing the pangenome cements their intent to employ ethics, dedication, justice, and equity in advancing genomics.

D. Outlook

We, as researchers of this subject, and as producers of a functional version of a read-alignment interface, play a role in the expansion of genomics. Our project seeks to understand the wider implications of the technology we're working with, and to make the information we have learned accessible to a general audience. In doing so, we have made a point to take a deep look at the ethical, equitable and social implications applicable to the field to which we contribute.

The indisputable truth is that the human genome is rapidly becoming a subject of intense scrutiny. As it becomes easier to access, store and share genomic information, including across borders, further questions will arise as the field moves in new directions. We, as a group, sincerely hope that the HPRC's framework for an equitable pangenome model will become the foundation for a new era of collaborative and equitable genomic study. It is imperative to genomics that studies, NGS technology, and consumer genetics, be held to high ethical standards in order to mitigate the misuse of genetic information. As best stated in 2008 by McGuire et al. "It is essential that, as we move forward, we do so with some caution and with conscious consideration, critical reflection and careful study of the many ethical and social implications of new developments at each step along the way."

CONCLUSION

The exploration of BWT and its applications in genomics has revealed its profound impact on bioinformatics in short-read sequence alignment. While designed for text compression, BWT is critical in genomic data processing.

Our implementation and analysis of BWT-based aligners in Python and C have revealed the efficiency and potential for optimization of the BWT transform. By implementing Radix Sort and SAIS in the future, we significantly improved runtime and memory usage. These improvements in space and runtime are crucial in genomics, where processing DNA sequences swiftly and accurately is vital.

As we delve deeper into genomics data, the ethical considerations become increasingly significant. The ease of data sharing and storage caused by the BWT brings forth concerns about privacy, consent, and equity. Our project recognizes the importance of these issues, advocating for responsible and ethical use of genomic data. The HPRC's efforts to create a more representative human reference genome is a step in the right direction that we stand by as it addresses biases in our current genomic models.

Had time permitted, our project could have explored several different avenues further to enhance the applicability and efficiency of the BWT. One idea would have been an advanced GUI, making our tool more accessible to a broader audience. This GUI could include advanced animations to represent genomic alignment and show the BWT transform being made to enhance the understanding of our results. Furthermore, an exploration of the application of BWT in long-read sequencing would have been a significant step for us. Long reads allow for more contiguous and accurate genome assemblies, making us see repetitive sequences easier than a short read. This adaptation would address the demands of long-read data and improve genome

assembly and alignment accuracy, allowing for more comprehensive and accurate genomic research. Lastly, alignment algorithms assign scores to sequences to achieve a “best” fit. Various scoring schemes prioritize some alignments over others. Applying this to our current project would have taken it further in accuracy and would be a better representation of currently used aligners.

The BWT’s application in genomics highlights its remarkable efficiency and versatility while bringing to light the ethical considerations of providing our genomic data. As the field of genomics evolves, it is imperative to balance technological advancements with responsible and equitable practices. Through its implementation and ethical deliberations, our project aims to contribute to this ongoing dialogue, aspiring to a future where genomic research is as honest as it is groundbreaking.

WORKS CITED

Allison,L. (1993) Normalization of Affine Gap Costs Used in Optimal Sequence Alignment. *Journal of Theoretical Biology*, **161**, 263–269.

Alser,M. *et al.* (2021) Technology dictates algorithms: recent developments in read alignment. *Genome Biology*, **22**, 249.

Altschul,S.F. and Erickson,B.W. (1986) Optimal sequence alignment using affine gap costs. *Bltn Mathcal Biology*, **48**, 603–616.

Canzar,S. and Salzberg,S.L. (2017) Short Read Mapping: An Algorithmic Tour. *Proc IEEE Inst Electr Electron Eng*, **105**, 436–458.

Fernandez,E. *et al.* (2011) String Matching in Hardware Using the FM-Index. In, *2011 IEEE 19th Annual International Symposium on Field-Programmable Custom Computing Machines*. IEEE, Salt Lake City, UT, USA, pp. 218–225.

Ferragina,P. and Manzini,G. (2005) Indexing compressed text. *J. ACM*, **52**, 552–581.

High-Performance k-mer Membership Queries over Spectral Burrows-Wheeler Transform with GPU computing.

Hu,T. *et al.* (2021) Next-generation sequencing technologies: An overview. *Human Immunology*, **82**, 801–811.

Huang,L. *et al.* (2013) Short read alignment with populations of genomes. *Bioinformatics*, **29**, i361–i370.

Kaya,M. *et al.* (2014) Multiple sequence alignment with affine gap by using multi-objective genetic algorithm. *Computer Methods and Programs in Biomedicine*, **114**, 38–49.

Kaye,J. *et al.* (2010) Ethical implications of the use of whole genome methods in medical research. *Eur J Hum Genet*, **18**, 398–403.

Kempa,D. and Kociumaka,T. (2022) Resolution of the burrows-wheeler transform conjecture. *Commun. ACM*, **65**, 91–98.

Lam,T.W. *et al.* (2008) Compressed indexing and local alignment of DNA. *Bioinformatics*, **24**, 791–797.

Lam,T.W. *et al.* (2009) High Throughput Short Read Alignment via Bi-directional BWT. In, *2009 IEEE International Conference on Bioinformatics and Biomedicine*., pp. 31–36.

Langmead,B. *et al.* (2009) Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biology*, **10**, R25.

Li,H. and Durbin,R. (2009) Fast and accurate short read alignment with Burrows–Wheeler transform. *Bioinformatics*, **25**, 1754–1760.

Manzini,G. and Ferragina,P. (2002) Engineering a Lightweight Suffix Array Construction Algorithm.

McGuire,A.L. *et al.* (2008) Research ethics and the challenge of whole-genome sequencing. *Nat Rev Genet*, **9**, 152–156.

Na,J.C. *et al.* (2018) FM-index of alignment with gaps. *Theoretical Computer Science*, **710**, 148–157.

Navarro,G. (2022) Technical perspective: The compression power of the BWT. *Commun. ACM*, **65**, 90–90.

Needleman,S.B. and Wunsch,C.D. (1970) A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J Mol Biol*, **48**, 443–453.

On the Complexity of Finite Sequences.

Piovesan,A. *et al.* (2019) On the length, weight and GC content of the human genome. *BMC Res Notes*, **12**, 106.

Quan,W. *et al.* (2020) Short Read Alignment Based on Maximal Approximate Match Seeds. *Front Mol Biosci*, **7**, 572934.

Seo,T.-K. *et al.* (2022) Correlations between alignment gaps and nucleotide substitution or amino acid replacement. *Proceedings of the National Academy of Sciences*, **119**, e2204435119.

Shendure,J. *et al.* (2017) DNA sequencing at 40: past, present and future. *Nature*, **550**, 345–353.

Sheng,C. and Dai,F. (2021) Burrows-Wheeler transform acceleration based on CUDA. *Proceedings of International Conference on Artificial Life and Robotics*, **26**, 596–599.

Smith,T.F. and Waterman,M.S. (1981) Identification of common molecular subsequences. *J Mol Biol*, **147**, 195–197.

Strimmer,K. *et al.* (2009) Genetic distances and nucleotide substitution models. In, Lemey,P. *et al.* (eds), *The Phylogenetic Handbook*. Cambridge University Press, pp. 111–141.

Tamames,J. *et al.* (2007) The frontier between cell and organelle: genome analysis of Candidatus Carsonella ruddii. *BMC Evol Biol*, **7**, 181.

Urgese,G. *et al.* (2014) Dynamic Gap Selector: A Smith Waterman Sequence Alignment Algorithm with Affine Gap Model Optimisation.

Wang,T. *et al.* (2022) The Human Pangenome Project: a global resource to map genomic diversity. *Nature*, **604**, 437–446.

Woolfit,M. (2009) Effective population size and the rate and pattern of nucleotide substitutions. *Biol Lett*, **5**, 417–420.

Zhang,Z. and Gerstein,M. (2003) Patterns of nucleotide substitution, insertion and deletion in the human genome inferred from pseudogenes. *Nucleic Acids Res*, **31**, 5338–5348.

Zokaee,F. *et al.* (2018) AligneR: A Process-in-Memory Architecture for Short Read Alignment in ReRAMs. *IEEE Computer Architecture Letters*, **17**, 237–240.