Alignment & BAM files – Answers Part 1

1. List different aligners.

Global alignments, which attempt to align every residue in every sequence, are most useful when the sequences in the query set are similar and of roughly equal size. A general global alignment technique is the Needleman–Wunsch algorithm, which is based on dynamic programming.
Local alignments are more useful for dissimilar sequences that are suspected to contain regions of similarity or similar sequence motifs within their larger sequence context. The Smith–Waterman algorithm is a general local alignment method based on the same dynamic programming scheme but with additional choices to start and end at any place.
Hybrid methods, known as semi-global methods, search for the best possible partial alignment of the two sequences (a combination of one or both starts and one or both ends is stated to be aligned). This can be especially useful when the downstream part of one sequence overlaps with the upstream part of the other sequence. In this case, neither global nor local alignment is entirely appropriate: a global alignment would attempt to force the alignment to extend beyond the region of overlap, while a local alignment might not fully cover the region of overlap. Another case where semi-global alignment is useful is when one sequence is short (for example a gene sequence) and the other is very long (for example a chromosome sequence). In that case, the short sequence should be globally (fully) aligned but only a local (partial) alignment is desired for the long sequence.
Pairwise sequence alignment methods are used to find the best-matching piecewise (local or global) alignments of two query sequences.
Multiple sequence alignment is an extension of pairwise alignment to incorporate more than two sequences at a time. Multiple alignment methods try to align all of the sequences in a given query set. Multiple sequence alignments are computationally difficult to produce and most formulations of the problem lead to NP-complete combinatorial optimization problems.
One way of quantifying the utility of a given pairwise alignment is the 'maximal unique match' (MUM), or the longest subsequence that occurs in both query sequences. Longer MUM sequences typically reflect closer relatedness. Identification of MUMs and other potential anchors, is the first step in larger alignment systems such as MUMmer. Anchors are the areas between two genomes where they are highly similar.
The dot-matrix approach, which implicitly produces a family of alignments for individual sequence regions, is qualitative and conceptually simple, though time-consuming to analyze on a large scale. To construct a dot-matrix plot, the two sequences are written along the top row and leftmost column of a two-dimensional matrix and a dot is placed at any point where the characters in the appropriate columns match. In the absence of noise, it can be easy to visually identify certain sequence features—such as insertions, deletions, repeats, or inverted repeats—from a dot-matrix plot. The dot plots of very closely related sequences will appear as a single line along the matrix's main diagonal. Dot-matrix plots can also be used to assess repetitiveness in a single sequence. A sequence can be plotted against itself and regions that share significant similarities will appear as lines off the main diagonal.
The technique of dynamic programming can be applied to produce global alignments via the Needleman-Wunsch algorithm, and local alignments via the Smith-Waterman algorithm. DNA and RNA alignments may use a scoring matrix, but in practice often simply assign a positive match score, a negative mismatch score, and a negative gap penalty. A common extension to standard linear gap costs, is the usage of two different gap penalties for opening a gap and for extending a gap. The dynamic programming method is guaranteed to find an optimal alignment given a particular scoring function, however, identifying a good scoring function is often an empirical rather than a theoretical matter. Although dynamic programming is extensible to more than two sequences, it is prohibitively slow for large numbers of sequences or extremely long sequences. In practice, the method requires large amounts of computing power or a system whose architecture is specialized for dynamic programming. The technique of dynamic programming is theoretically applicable to

any number of sequences, but, because it is computationally expensive in both time and memory, it is rarely used for more than three or four sequences in its most basic form.
Word methods, also known as k-tuple methods, are heuristic methods that are not guaranteed to find an optimal alignment solution, but are significantly more efficient than dynamic programming. These methods are especially useful in large-scale database searches where it is understood that a large proportion of the candidate sequences will have essentially no significant match with the query sequence. Word methods identify a series of short, non-overlapping subsequences ("words") in the query sequence that are then matched to candidate database sequences. The relative positions of the word in the two sequences being compared are subtracted to obtain an offset; this will indicate a region of alignment if multiple distinct words produce the same offset. Only if this region is detected do these methods apply more sensitive alignment criteria; thus, many unnecessary comparisons with sequences of no appreciable similarity are eliminated.
Progressive, hierarchical, or tree methods generate a multiple sequence alignment by first aligning the most similar sequences and then adding successively less related sequences or groups to the alignment until the entire query set has been incorporated into the solution. The initial tree describing the sequence relatedness is based on pairwise comparisons that may include heuristic pairwise alignment methods similar to FASTA. Progressive alignment results are dependent on the choice of "most related" sequences and thus can be sensitive to inaccuracies in the initial pairwise alignments. Most progressive multiple sequence alignment methods additionally weight the sequences in the query set according to their relatedness, which reduces the likelihood of making a poor choice of initial sequences and thus improves alignment accuracy.
Iterative methods attempt to improve on the heavy dependence on the accuracy of the initial pairwise alignments, which is the weak point of the progressive methods. Iterative methods optimize an objective function based on a selected alignment scoring method by assigning an initial global alignment and then realigning sequence subsets. The realigned subsets are then themselves aligned to produce the next iteration's multiple sequence alignment.
Motif finding, also known as profile analysis, constructs global multiple sequence alignments that attempt to align short conserved sequence motifs among the sequences in the query set. This is usually done by first constructing a general global multiple sequence alignment, after which the highly conserved regions are isolated and used to construct a set of profile matrices. The profile matrix for each conserved region is arranged like a scoring matrix but its frequency counts for each nucleotide at each position are derived from the conserved region's character distribution rather than from a more general empirical distribution. The profile matrices are then used to search other sequences for occurrences of the motif they characterize.
A variety of general optimization algorithms commonly used in computer science have also been applied to the multiple sequence alignment problem. Hidden Markov models have been used to produce probability scores for a family of possible multiple sequence alignments for a given query set. Although early HMM-based methods produced underwhelming performance, later applications have found them especially effective in detecting remotely related sequences because they are less susceptible to noise created by conservative or semiconservative substitutions. Genetic algorithms and simulated annealing have also been used in optimizing multiple sequence alignment scores as judged by a scoring function like the sum-of-pairs method. The Burrows–Wheeler transform has been successfully applied to fast short read alignment.

Short-read sequence aligners:
needle – Needleman–Wunsch algorithm dynamic programming
NW-align – Needleman–Wunsch algorithm dynamic programming
water – Smith–Waterman algorithm dynamic programming
SSEARCH – Smith-Waterman search, slower but more sensitive than FASTA
JAligner – Java open-source implementation of Smith-Waterman
SWIFOLD – Smith-Waterman Acceleration on Intel's FPGA with OpenCL for Long DNA Sequences

UGENE – Smith-Waterman for SSE/CUDA, Suffix array based repeats finder & dotplot, Supports multiple alignment with MUSCLE, KAlign, Clustal and MAFFT plugins, Visual interface both for Bowtie and BWA, and an embedded aligner
MUMmer – suffix tree based, enabling it to be applied to very long sequences
BLAST – Local search with fast k-tuple heuristic (Basic Local Alignment Search Tool)
BLASTN – BLAST's nucleotide alignment program, slow and not accurate for short reads, and uses a sequence database (EST, Sanger sequence) rather than a reference genome.
ScalaBLAST – Highly parallel Scalable BLAST
EMBOSS – European Molecular Biology Open Software Suite, contains a variety of applications for sequence alignment, rapid database searching with sequence patterns, protein motif identification (including domain analysis), and much more.
FASTA – Local search with fast k-tuple heuristic, slower but more sensitive than BLAST
PHI-Blast – Motif search and alignment tool
MSA – Dynamic programming, perform multiple sequence alignment under the sum-of-pairs criterion
Clustal – based on deriving phylogenetic trees from pairwise sequences of amino acids or nucleotides.
ClustalW – Progressive alignment, allowing individual sequences to be weighted down or up according to similarity or divergence respectively in a partial alignment
MAFFT – Progressive-iterative alignment (multiple alignment using fast Fourier transform)
T-Coffee – More sensitive progressive alignment (Tree-based Consistency Objective Function for Alignment Evaluation)
PSAlign – Alignment preserving non-heuristic
MUSCLE – Progressive-iterative alignment (MUltiple Sequence Comparison by Log-Expectation)
POA – Partial order/hidden Markov model
SAM – Local and global search with profile Hidden Markov models, more sensitive than PSI-BLAST
HMMER – Local and global search with profile Hidden Markov models, more sensitive than PSI-BLAST
MegAlign Pro – Software to align DNA, RNA, protein, or DNA + protein sequences via pairwise and multiple sequence alignment algorithms including MUSCLE, Mauve, MAFFT, Clustal Omega, Jotun Hein, Wilbur-Lipman, Martinez Needleman-Wunsch, Lipman-Pearson and Dotplot analysis.
DNADynamo – linked DNA to Protein multiple alignment with MUSCLE, Clustal and Smith-Waterman
BBMap – Uses a short kmers to rapidly index genome; no size or scaffold count limit. Higher sensitivity and specificity than Burrows–Wheeler aligners, with similar or greater speed. Performs affine-transform-optimized global alignment, which is slower but more accurate than Smith-Waterman. Handles Illumina, 454, PacBio, Sanger, and Ion Torrent data. Splice-aware; capable of processing long indels and RNA-seq. Pure Java; runs on any platform. Used by the Joint Genome Institute.
BFAST – Explicit time and accuracy tradeoff with a prior accuracy estimation, supported by indexing the reference sequences. Optimally compresses indexes. Can handle billions of short reads. Performs a full Smith Waterman alignment.
MOSAIK – Fast gapped aligner and reference-guided assembler. Aligns reads using a banded Smith-Waterman algorithm seeded by results from a k-mer hashing scheme. Supports reads ranging in size from very short to very long.
Bowtie – Uses a Burrows–Wheeler transform to create a permanent, reusable index of the genome; 1.3 GB memory footprint for human genome. Aligns more than 25 million Illumina reads in 1 CPU hour. Supports Maq-like and SOAP-like alignment policies.
BWA – Uses a Burrows–Wheeler transform to create an index of the genome. It's a bit slower than Bowtie but allows indels in alignment.

BWA-PSSM – A probabilistic short read aligner based on the use of position specific scoring matrices (PSSM). The aligner is adaptable in the sense that it can take into account the quality scores of the reads and models of data specific biases.
BarraCUDA – A GPGPU accelerated Burrows–Wheeler transform (FM-index) short read alignment program based on BWA, supports alignment of indels with gap openings and extensions.
CUDAlign – DNA sequence alignment of unrestricted size in single or multiple GPUs
CUSHAW – A CUDA compatible short read aligner to large genomes based on Burrows–Wheeler transform.
GEM – High-quality alignment engine (exhaustive mapping with substitutions and indels). More accurate and several times faster than BWA or Bowtie 1/2. Many standalone biological applications (mapper, split mapper, mappability, and other) provided.
LAST – Uses adaptative seeds and copes more efficiently with repeat-rich sequences (e.g. genomes). For example: it can align reads to genomes without repeat-masking, without becoming overwhelmed by repetitive hits.
NextGenMap – Flexible and fast read mapping program (twice as fast as BWA), achieves a mapping sensitivity comparable to Stampy. Internally uses a memory efficient index structure (hash table) to store positions of all 13-mers present in the reference genome. Mapping regions where pairwise alignments are required are dynamically determined for each read.
rNA – A randomized Numerical Aligner for Accurate alignment of NGS reads.
SOAP – robust with a small (1-3) number of gaps and mismatches. Speed improvement over BLAT, uses a 12 letter hash table.
SOAP2 – using bidirectional BWT to build the index of reference, and it is much faster than the first version.
SOAP3 – GPU-accelerated version that could find all 4-mismatch alignments in tens of seconds per one million reads.
SOAP3-dp – also GPU accelerated, supports arbitrary number of mismatches and gaps according to affine gap penalty scores.
Stampy – For Illumina reads. High specificity, and sensitive for reads with indels, structural variants, or many SNPs. Slow, but speed increased dramatically by using BWA for first alignment pass.
HIVE-hexagon – Uses a hash table and bloom matrix to create and filter potential positions on the genome. For higher efficiency uses cross-similarity between short reads and avoids realigning non unique redundant sequences. It is faster than Bowtie and BWA and allows indels and divergent sensitive alignments on viruses, bacteria, and more conservative eukaryotic alignments.
VelociMapper – FPGA-accelerated reference sequence alignment mapping tool from TimeLogic. Faster than Burrows–Wheeler transform-based algorithms like BWA and Bowtie. Supports up to 7 mismatches and/or indels with no performance penalty. Produces sensitive Smith–Waterman gapped alignments.

2. BAM files

2.1. What BAM files contain?

2.2. How we get BAM files?

A BAM file (Binary Alignment Map) is the compressed binary version of a SAM file (Sequence Alignment Map) that is used to represent aligned sequences, a compact and indexable representation of nucleotide sequence alignments. BAM is compressed in the BGZF format. BAM format uses 0-based coordinate system, where as SAM uses 1-based coordinate system.

2.3. List BAM file fields and mark mandatory ones.

The structure of BAM files include a header section and an alignment section:

Header—The sample name, sample length, and alignment method are all included in this section. The alignments section contains alignments that are linked to specific information in the header section.
Alignments—The read name, read sequence, read quality, alignment information, and custom tags are all included in this file. The chromosome, start coordinate, alignment quality, and match descriptor string are all included in the read name.

SAM file fields:
Header:
Each header line begins with the character '@' followed by one of the two-letter header record type codes. In the header, each line is TAB-delimited and, apart from @CO lines, each data field follows a format 'TAG:VALUE' where TAG is a two-character string that defines the format and content of VALUE. No field tag may appear more than once and the order in which the fields appear is not significant. Tags marked with sign '*' are required.
@HD – Optional. If present, there must be only one @HD line and it must be the first line of the file. Tags: VN * – Version number; SO – Sorting order of alignment (Valid values: unknown (default), unsorted, queryname and coordinate); GO – Grouping of alignments, indicating that similar alignment records are grouped together but the file is not necessarily sorted overall (Valid values: none (default), query (alignments are grouped by QNAME), and reference (alignments are grouped by RNAME/POS)); SS – Sub-sorting order of alignments (Valid values are of the form sort-order:sub-sort, where sort-order is the same value stored in the SO tag and sub-sort is an implementation-dependent colon-separated string further describing the sort order, but with some predefined terms.
@SQ – Reference sequence dictionary. The order of @SQ lines defines the alignment sorting order. Tags: RS * – Reference sequence name. The SN tags and all individual AN names in all @SQ lines must be distinct. The value of this field is used in the alignment records in RNAME and RNEXT fields; LN * – Reference sequence length; AH – Indicates that this sequence is an alternate locus. The value is the locus in the primary assembly for which this sequence is an alternative, in the format 'chr :start-end ', 'chr ' (if known), or '*' (if unknown), where 'chr ' is a sequence in the primary assembly. Must not be present on sequences in the primary assembly; AN – Alternative reference sequence names. A comma-separated list of alternative names that tools may use when referring to this reference sequence. These alternative names are not used elsewhere within the SAM file; in particular, they must not appear in alignment records' RNAME or RNEXT fields; AS – Genome assembly identifier; DS – Description; M5 – MD5 checksum of the sequence; SP – Species; TP – Molecule topology (Valid values: linear (default) and circular); UR – URI of the sequence. This value may start with one of the standard protocols, e.g., 'http:' or 'ftp:'. If it does not start with one of these protocols, it is assumed to be a file-system path.
@RG – Read group. Unordered multiple @RG lines are allowed. Tags: ID * – Read group identifier. Each @RG line must have a unique ID. The value of ID is used in the RG tags of alignment records. Must be unique among all read groups in header section. Read group IDs may be modified when merging SAM files in order to handle collisions; BC – Barcode sequence identifying the sample or library. This value is the expected barcode bases as read by the sequencing machine in the absence of errors. If there are several barcodes for the sample/library (e.g., one on each end of the template), the recommended implementation concatenates all the barcodes separating them with hyphens ('-'); SN – Name of sequencing center producing the read; DS – Description; DT – Date the run was produced (ISO8601 date or date/time); FO – Flow order. The array of nucleotide bases that correspond to the nucleotides used for each flow of each read; KS – The array of nucleotide bases that correspond to the key sequence of each read; LB – Library; PG – Programs used for processing the read group; PI – Predicted median insert size; PL – Platform/technology used to produce the reads (Valid values: CAPILLARY, DNBSEQ (MGI/BGI), ELEMENT, HELICOS, ILLUMINA, IONTORRENT, LS454, ONT (Oxford Nanopore), PACBIO (Pacific Bio-sciences), SOLID, and ULTIMA. This field should be omitted when the technology is not in this list (though the PM field

may still be present in this case) or is unknown); PM – Platform model. Free-form text providing further details of the platform/technology used; PU – Platform unit (e.g., flowcell-barcode.lane for Illumina or slide for SOLiD). Unique identifier; SM – Sample. Use pool name where a pool is being sequenced.
@PG – Program. Tags: ID * – Program record identifier. Each @PG line must have a unique ID. The value of ID is used in the alignment PG tag and PP tags of other @PG lines. PG IDs may be modified when merging SAM files in order to handle collisions; PN – Program name; CL – Command line; PP – Previous @PG-ID. Must match another @PG header's ID tag. @PG records may be chained using PP tag, with the last record in the chain having no PP tag. This chain defines the order of programs that have been applied to the alignment; DS – Description; VN – Program version.
@CO – One-line text comment. Unordered multiple @CO lines are allowed.

Alignments:
Each alignment line typically represents the linear alignment of a segment. Each line consists of 11 or more TAB-separated fields. The first eleven fields are always present and in the defined order (marked with sign '*'). If the information represented by any of these fields is unavailable, that field's value will be a placeholder, either '0' or '*' as determined by the field's type. All mapped segments in alignment lines are represented on the forward genomic strand. For segments that have been mapped to the reverse strand, the recorded SEQ is reverse complemented from the original unmapped sequence and CIGAR, QUAL, and strand-sensitive optional fields are reversed and thus recorded consistently with the sequence bases as represented.
QNAME * – Query template name. Reads/segments having identical QNAME are regarded to come from the same template. A QNAME '*' indicates the information is unavailable.
FLAG * – Combination of bitwise FLAGs, decimal (base-10) number is used to represent a binary (base-2) number with digits that represent different true/false statements pertaining to the alignment of the read. A value of zero indicates false while one indicates true: 2^0 – template having multiple segments in sequencing (paired read); 2^1 – each segment properly aligned according to the aligner (read mapped in proper pair); 2^2 – segment unmapped (read is not aligned); 2^3 – next segment in the template unmapped (its mate is not aligned); 2^4 – SEQ being reverse complemented; 2^5 – SEQ of the next segment in the template being reverse complemented (its mate aligned in reverse direction); 2^6 – the first segment in the template (first in pair); 2^7 – the last segment in the template (second in pair); 2^8 – secondary alignment (read had multiple potential alignments, this was one of them, but not the first choice from among them); 2^9 – not passing filters, such as platform/vendor quality controls (read failed quality check); 2^{10} – PCR or optical duplicate (read is flagged as duplicate); 2^{11} – supplementary alignment (exact meaning varies by aligner).
RNAME * – Reference sequence NAME of the alignment. If @SQ header lines are present, RNAME (if not '*') must be present in one of the SQ-SN tag. An unmapped segment without coordinate has a '*' at this field. However, an unmapped segment may also have an ordinary coordinate such that it can be placed at a desired position after sorting.
POS * – 1-based leftmost mapping POSition of the first CIGAR operation that "consumes" a reference base (see table below). The first base in a reference sequence has coordinate 1. POS is set as 0 for an unmapped read without coordinate.
MAPQ * – Mapping quality. It equals $-10 \log_{10} \Pr\{\text{mapping position is wrong}\}$, rounded to the nearest integer. A value 255 indicates that the mapping quality is not available.
CIGAR * – CIGAR string (concise idiosyncratic gapped alignment report string) is a sequence of numbers and letters (in that order) indicating continuities or discontinuities in the alignment caused by inserted or deleted bases (or other causes for discontinuity). M – alignment match (can be a sequence match or mismatch); I – insertion to the reference; D – deletion from the reference; N – skipped region from the reference (for mRNA-to-genome alignment, an N operation represents an intron, for other types of alignments, the interpretation of N is not defined); S – soft clipping (clipped sequences present in SEQ) (S may only have H operations between them and the ends of

the CIGAR string); H – hard clipping (clipped sequences NOT present in SEQ) (H can only be present as the first and/or last operation); P – padding (silent deletion from padded reference); = – sequence match; X – sequence mismatch. Sum of lengths of the M/I/S/=/X operations shall equal the length of SEQ.
RNEXT * – Reference sequence name of the primary alignment of the next read (mate) in the template. For the last read, the next read is the first read in the template. If @SQ header lines are present, RNEXT (if not '*' or '=') must be present in one of the SQ-SN tag. This field is set as '*' when the information is unavailable, and set as '=' if RNEXT is identical RNAME.
PNEXT * – Position of the mate/next read. 1-based Position of the primary alignment of the next read in the template. Set as 0 when the information is unavailable. This field equals POS at the primary line of the next read.
TLEN * – signed observed Template length. For primary reads where the primary alignments of all reads in the template are mapped to the same reference sequence, the absolute value of TLEN equals the distance between the mapped end of the template and the mapped start of the template, inclusively (i.e., end − start + 1). It indicates the length of template sequence to which the read maps (this field is sometimes confused for the read length, which it is not, but will often be equal to in value). A read with multiple insertions may have a smaller template length than the read length, while a read with multiple deletions may have a template length longer than the read length. In the case of RNA or cDNA being aligned to genomic DNA reference, template length may be in the tens of thousands of bases for a short read due to the presence of an intron.
SEQ * – segment sequence. This field can be a '*' when the sequence is not stored. If not a '*', the length of the sequence must equal the sum of lengths of M/I/S/=/X operations in CIGAR. Should generally follow the sequence line from the source FASTQ.
QUAL * – ASCII of Phred-scaled base quality plus 33, same as the quality string in the Sanger FASTQ format. This field can be a '*' when quality is not stored. If not a '*', SEQ must not be a '*' and the length of the quality string ought to equal the length of SEQ.
All optional fields follow the TAG:TYPE:VALUE format where TAG is a two-character string. Within each alignment line, no TAG may appear more than once and the order in which the optional fields appear is not significant. A TAG containing lowercase letters is reserved for end users. In an optional field, TYPE is a single case-sensitive letter which defines the format of VALUE (A – printable character, i – signed integer, f – single-precision float number, Z – printable sting including spaces, H – byte array in HEX format, B – integer or numeric array). For an integer or numeric array (type 'B'), the first letter indicates the type of numbers in the following comma separated array. The letter can be one of 'cCsSiIf', corresponding to int8_t (signed 8-bit integer), uint8_t (unsigned 8-bit integer), int16_t, uint16_t, int32_t, uint32_t and float, respectively. Tags starting with 'X', 'Y' or 'Z' and tags containing lowercase letters in either position are reserved for local use.
AM:i: The smallest template-independent mapping quality of segments in the rest
AS:i: Alignment score generated by aligner.
BC:Z: Barcode sequence, with any quality scores stored in the QT tag.
BQ:Z: Offset to base alignment quality (BAQ), of the same length as the read sequence. At the i-th read base, BAQi = Qi − (BQi − 64) where Qi is the i-th base quality.
CC:Z: Reference name of the next hit; '=' for the same chromosome.
CM:i: Edit distance between the color sequence and the color reference.
CO:Z: Free-text comments.
CP:i: Leftmost coordinate of the next hit.
CQ:Z: Color read quality on the original strand of the read. Same encoding as QUAL; same length as CS.
CS:Z: Color read sequence on the original strand of the read. The primer base must be included.
CT:Z: Complete read annotation tag, used for consensus annotation dummy features.
E2:Z: The 2nd most likely base calls. Same encoding and same length as QUAL.
FI:i: The index of segment in the template.

FS:Z: Segment suffix.
FZ:B:S Flow signal intensities on the original strand of the read, stored as (uint16_t) round(value * 100.0).
LB:Z: Library. Value to be consistent with the header RG-LB tag if @RG is present.
H0:i: Number of perfect hits.
H1:i: Number of 1-difference hits.
H2:i: Number of 2-difference hits.
HI:i: Query hit index, indicating the alignment record is the i-th one stored in SAM.
IH:i: Number of stored alignments in SAM that contains the query in the current record.
MC:Z: CIGAR string for mate/next segment
MD:Z: String for mismatching positions.
MQ:i: Mapping quality of the mate/next segment.
NH:i: Number of reported alignments that contains the query in the current record.
NM:i: Edit distance to the reference, including ambiguous bases but excluding clipping.
OQ:Z: Original base quality (usually before recalibration). Same encoding as QUAL.
OP:i: Original mapping position (usually before realignment).
OC:Z: Original CIGAR (usually before realignment).
PG:Z: Program. Value matches the header PG-ID tag if @PG is present.
PQ:i: Phred likelihood of the template, conditional on both the mapping being correct.
PU:Z: Platform unit. Value to be consistent with the header RG-PU tag if @RG is present.
QT:Z: Phred quality of the barcode sequence in the BC (or RT) tag. Same encoding as QUAL.
Q2:Z: Phred quality of the mate/next segment sequence in the R2 tag. Same encoding as QUAL.
R2:Z: Sequence of the mate/next segment in the template.
RG:Z: Read group. Value matches the header RG-ID tag if @RG is present in the header.
RT:Z: Deprecated alternative to BC tag originally used at Sanger.
SA:Z: Other canonical alignments in a chimeric alignment, formatted as a semicolon-delimited list: (rname, pos, strand, CIGAR, mapQ, NM;)+. Each element in the list represents a part of the chimeric alignment. Conventionally, at a supplementary line, the first element points to the primary line.
SM:i: Template-independent mapping quality.
TC:i: The number of segments in the template.
U2:Z: Phred probility of the 2nd call being wrong conditional on the best being wrong. The same encoding as QUAL.
UQ:i: Phred likelihood of the segment, conditional on the mapping being correct.

BAM format fields:
magic – BAM magic string
l_text – Length of the header text
text – Plain header text in SAM
n_ref – number of reference sequences
List of reference information (n_ref information): l_name – Length of the reference name plus 1; name – Reference sequence name; l_ref – Length of the reference sequence.
List of alignments: block_size – Total length of the alignment record, excluding this field; refID – Reference sequence ID, $-1 \leq$ refID < n ref, -1 for a read without a mapping position; pos – 0-based leftmost coordinate (= POS − 1); l_read_name – Length of read name (= length(QNAME) + 1); mapq – Mapping quality (=MAPQ); bin – BAI index bin; n_cigar_op – Number of operations in CIGAR; flag – Bitwise flags (= FLAG); l_seq – Length of SEQ; next_refID – Ref-ID of the next segment ($-1 \leq$ next refID < n ref); next_pos – 0-based leftmost pos of the next segment (= PNEXT − 1); tlen – Template length (= TLEN); read_name – Read name, NUL-terminated (QNAME with trailing '\0'); cigar – CIGAR: op len<<4|op. 'MIDNSHP=X'→'012345678' ; seq – 4-bit encoded read: '=ACMGRSVTWYHKDBN'→ [0, 15]; qual – Phred-scaled base qualities.

List of auxiliary data: tag – Two-character tag; val_type – Value type: AcCsSiIfZHB; value – Tag value.

2.4. How we know which reads in BAM files are pairs?

By looking in the FLAG filed.
Example input:
samtools flag $(samtools view -f 2 input.bam | cut -f2)
Example output:

```
0x4a3  1187   PAIRED,PROPER_PAIR,MREVERSE,READ2,DUP
0xa3   163    PAIRED,PROPER_PAIR,MREVERSE,READ2
0x93   147    PAIRED,PROPER_PAIR,REVERSE,READ2
0xa3   163    PAIRED,PROPER_PAIR,MREVERSE,READ2
0x93   147    PAIRED,PROPER_PAIR,REVERSE,READ2
0x53   83     PAIRED,PROPER_PAIR,REVERSE,READ1
0x63   99     PAIRED,PROPER_PAIR,MREVERSE,READ1
0x93   147    PAIRED,PROPER_PAIR,REVERSE,READ2
0x53   83     PAIRED,PROPER_PAIR,REVERSE,READ1
0x63   99     PAIRED,PROPER_PAIR,MREVERSE,READ1
```

2.5. What are inserts and how to get them from file?
2.6. What are fragments?
2.7. How different sizes of inserts affect sequencing performance?

In an Illumina sequencing run, either single-end sequencing or paired-end sequencing can be used. In any case, DNA is chopped into small fragments, ligated with adapters at both ends, and sequenced either from one end or from both ends. For paired-end reads, insert size refers to the fragment size (excluding the adapters). It refers to the piece of DNA inserted between the adapters which enable amplification and sequencing of that piece of DNA. Therefore, the insert size includes forward and reverse read as well as the unknown gap between them. The unknown gap can be called the inner mate distance.

```
~~~                                ~~~   Adapters
    ====================               Insert
    Read1
    ------->
~~~====================~~~               Fragment
                        <----------
                           Read2
           <---------->                   Inner distance
    <------------------------------>      Insert size
<---------------------------------------> Fragment size
```

Insert sizes are estimated using TLEN field.
Example for getting insert sizes considering only properly mapped pair:
samtools view -f 2 input.bam | cut -f9
Example output:

```
433
433
-387
419
-370
-376
```

199
-188
-396
423

When the first read is mapped to reverse strand we get the negative value for the insert size. In this case, the insert size of the second read is positive.
The insert size estimation using this simple method has an limitation: if the RNA-seq reads were mapped against genomes (especially eukaryotic genomes), instead of transcriptomes, the reported insert size can be huge, up to thousands or even more bases. It happens because mapping reads to genomes requires that the read aligner splits the reads and map sub-reads to the genome, because in eukaryotic and especially mammalian genomes, two exons can be separated by an intron of hundreds or even thousands nucleotides. This is why estimated insert sizes from genome-mapped SAM/BAM files can be misleading.

3. Samtools
3.1. What is samtools?
3.2. Usage?

Samtools is a set of utilities that manipulate alignments in the SAM, BAM, and CRAM formats. It converts between the formats, does sorting, merging and indexing, and can retrieve reads in any regions swiftly. SAM files are human-readable text files, and BAM files are simply their binary equivalent, whilst CRAM files are a restructured column-oriented binary container format. BAM files are typically compressed and more efficient for software to work with than SAM. SAMtools makes it possible to work directly with a compressed BAM file, without having to uncompress the whole file. SAMtool commands follow a stream model. This allows combining multiple commands into a data processing pipeline. Although the final output can be very complex, only a limited number of simple commands are needed to produce it. If not specified, the standard streams (stdin, stdout, and stderr) are assumed. Data sent to stdout are printed to the screen by default but are easily redirected to another file using the normal Unix redirectors, or to another command via a pipe.

3.3 Check a content of BAM file

Example input – to view content of BAM file including header:
samtools view -h input.bam

3.4. List first 5 lines of BAM

Example – including header (it will be first 5 lines of header):
samtools view -h input.bam | head -5

Example – excluding header:
samtools view input.bam | head -5

3.5. List header of BAM file

Example input:
samtools view -H input.bam

3.6. List different types of sorting BAM files and then sort it by using samtools

SAM or BAM file can be sorted by coordinate, queryname (QNAME), or some other property of the SAM record. The Sort Order of a SAM/BAM file is found in the SAM file header tag @HD in the field labeled SO. FFor coordinate sort, the major sort key is the RNAME field, with order defined by the order of @SQ lines in the header. The minor sort key is the POS field. For alignments with equal RNAME and POS, order is arbitrary. All alignments with '*' in RNAME field follow alignments with some other value but otherwise are in arbitrary order. For queryname sort, no explicit requirement is made regarding the ordering other than that it be applied consistently throughout the entire file.
In GATK tool SortSam for queryname-sorted alignments, the tool orders records deterministically by queryname field followed by record strand orientation flag, primary record flag, and secondary alignment flag.

Example input:
samtools sort input.bam -o sorted.bam

3.7. Check the depth of BAM file

samtools depth – computes the read depth at each position or region. The output of samtools depth has three columns. The first is the name of the reference, the second is the position, and the third is the number of reads aligned at that position.
If we need average depth of each covered base it can be calculated by summing the third column and dividing by number of bases (in this case we need to use option samtools depth -a to output all positions including those with zero depth).

Example input:
samtools depth input.bam
Example output:

Chr3	11699950	1
Chr3	11699951	1
Chr3	11699952	2
Chr3	11699953	3
Chr3	11699954	3
Chr3	11699955	3
Chr3	11699956	3
Chr3	11699957	3

3.8. Check the coverage of BAM file

samtools coverage – produces a histogram or table of coverage per chromosome. The tabulated form uses the following headings: rname - Reference name / chromosome; startpos - Start position; endpos - End position (or sequence length); numreads - Number reads aligned to the region (after filtering); covbases - Number of covered bases with depth >= 1; coverage - Percentage of covered bases [0..100]; meandepth - Mean depth of coverage; meanbaseq - Mean base quality in covered region; meanmapq - Mean map quality of selected reads.

Example input:
Samtools coverage input.bam
Example output:

Chr3	1	23042017	198050	287654	1.24839	0.362466	31	36.9

3.9. Compare depth & coverage of BAM file with view in IGV

By default IGV dynamically calculates and displays the default coverage track for an alignment file. When IGV is zoomed to the alignment read visibility threshold, the coverage track displays the depth of the reads displayed at each locus as a gray bar chart. If a nucleotide differs from the reference sequence in greater than 20% of quality weighted reads, IGV colors the bar in proportion to the read count of each base (A, C, G, T).
The dynamically calculated coverage data can be augmented by loading pre-computed coverage data from a file. When this option is used the track displays coverage at all zoom levels including at the whole genome and chromosome view.

3.10. Index BAM file

Indexing aims to achieve fast retrieval of alignments overlapping a specified region without going through the whole alignments. BAM must be sorted by the reference ID and then the leftmost coordinate before indexing.

samtools index sorted.bam

We get sorted.bam.bai file.

3.11. List only properly paired, mapped reads

samtools view -f 2 input.bam

3.12. List only duplicates

samtools view -f 1024 input.bam

3.13. List only secondary alignments

samtools view -f 256 input.bam

3.14. Import only properly paired and mapped reads in IGV and then only secondary alignments and observe the difference

samtools view -f 2 -h input.bam > paired.bam
samtools sort paired.bam -o paired.sorted.bam
samtools index paired.sorted.bam
samtools view -f 256 -h input.bam > secondary.bam
samtools sort paired.bam -o secondary.sorted.bam
samtools index secondary.sorted.bam

My example.bam does not have secondary alignments. I will have to find another file for example.

3.15. What are secondary alignments?

A read may map ambiguously to multiple locations, e.g. due to repeats. Only one of the multiple read alignments is considered primary, and this decision may be arbitrary. All other alignments have the secondary alignment flag.

3.16. Merge 2 BAM files into one

samtools merge – Merge multiple sorted alignment files, producing a single sorted output file that contains all the input records and maintains the existing sort order.

Samtools merge merged.bam input1.bam input2.bam