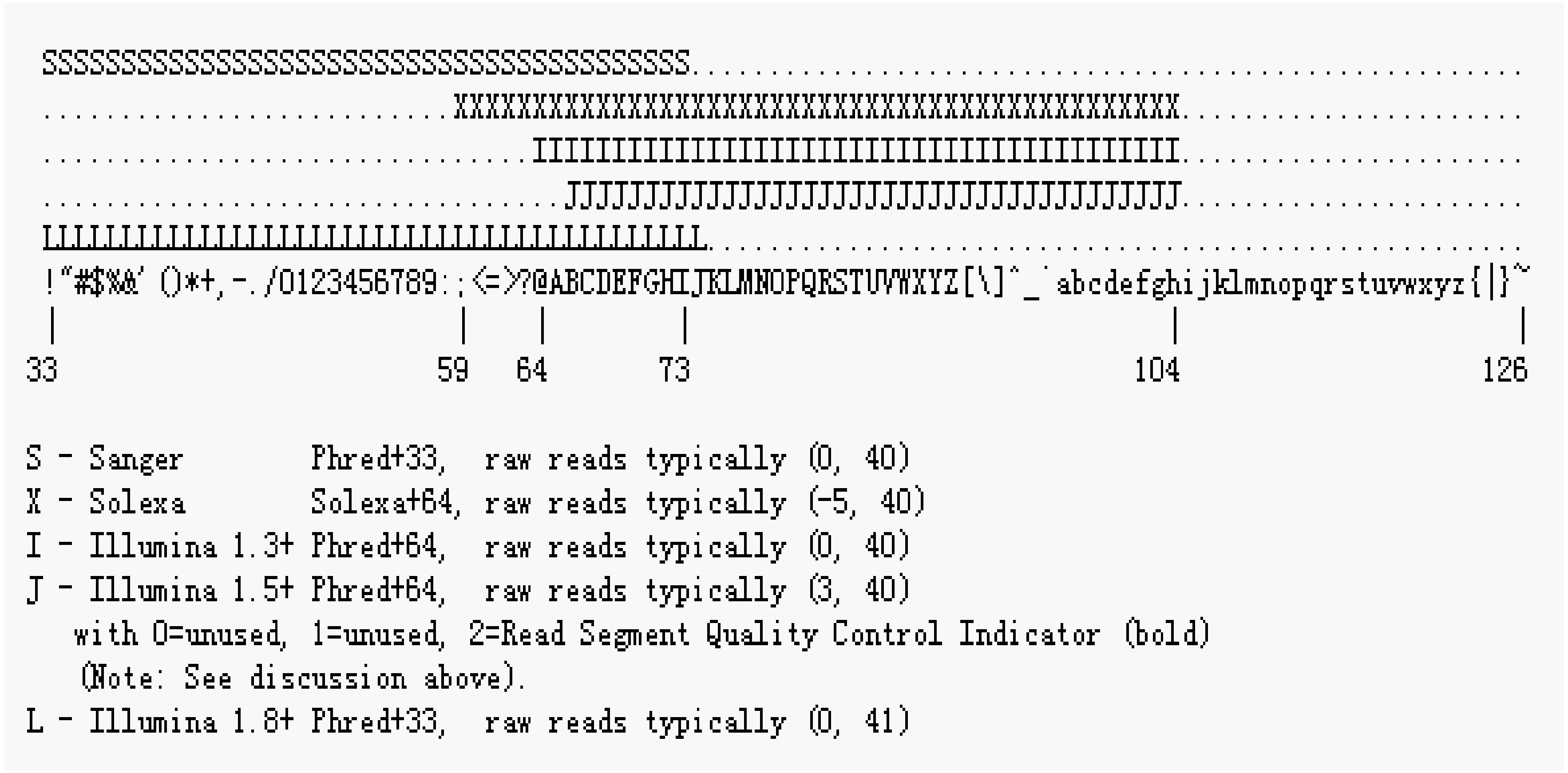
Lab 3: Analysis of Resequencing Data

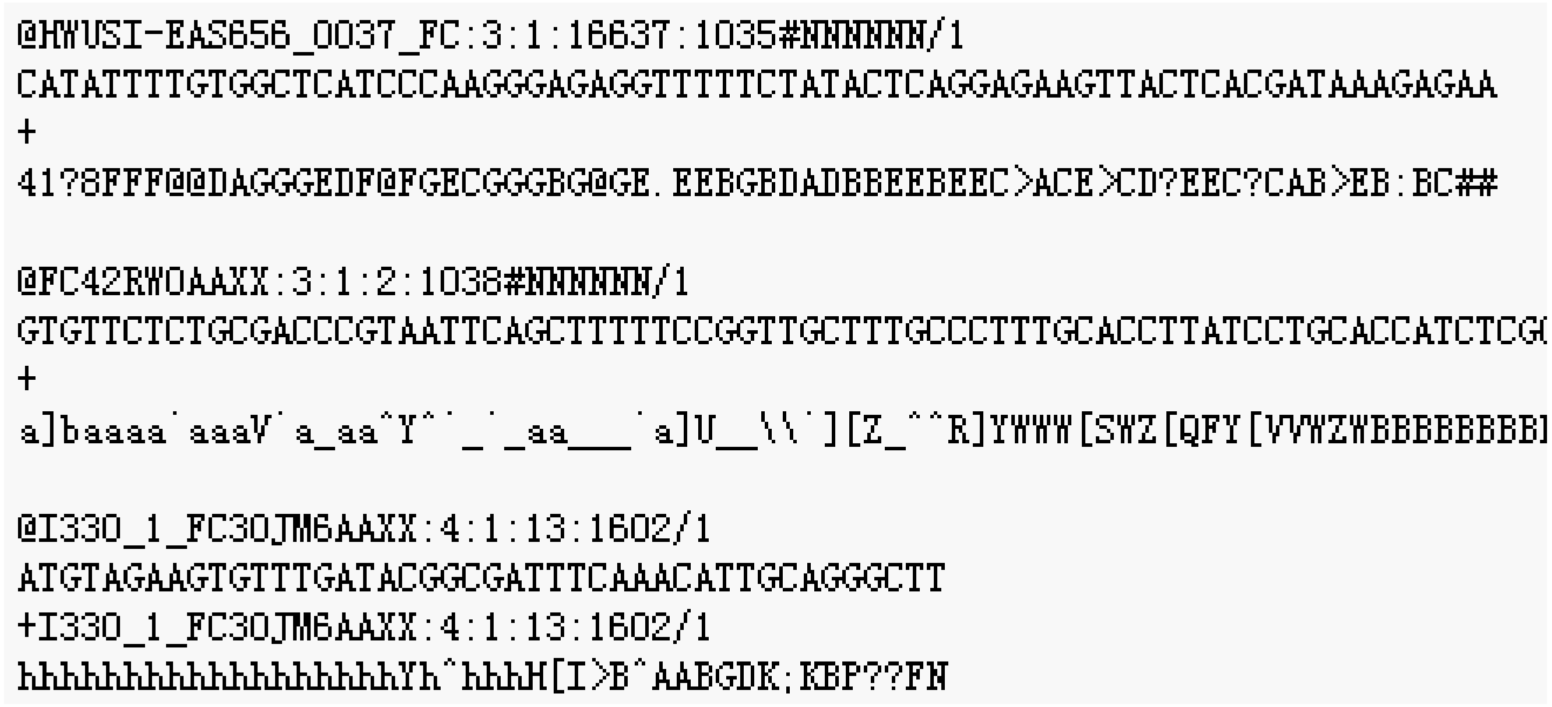
Quality encoding table



```
  SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS.....................................................
  ..........................XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX......................
  ...............................IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII......................
  .................................JJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJ......................
  LLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL....................................................
  !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
  |                         |    |        |                              |                     |
 33                        59   64       73                            104                   126

 S - Sanger        Phred+33,  raw reads typically (0, 40)
 X - Solexa        Solexa+64, raw reads typically (-5, 40)
 I - Illumina 1.3+ Phred+64,  raw reads typically (0, 40)
 J - Illumina 1.5+ Phred+64,  raw reads typically (3, 40)
    with 0=unused, 1=unused, 2=Read Segment Quality Control Indicator (bold)
    (Note: See discussion above).
 L - Illumina 1.8+ Phred+33,  raw reads typically (0, 41)
```

Identify quality encoding

Use the table above encoding table. You only need to differentiate between Sanger (S), Solexa (X) and Illumina (I, J)

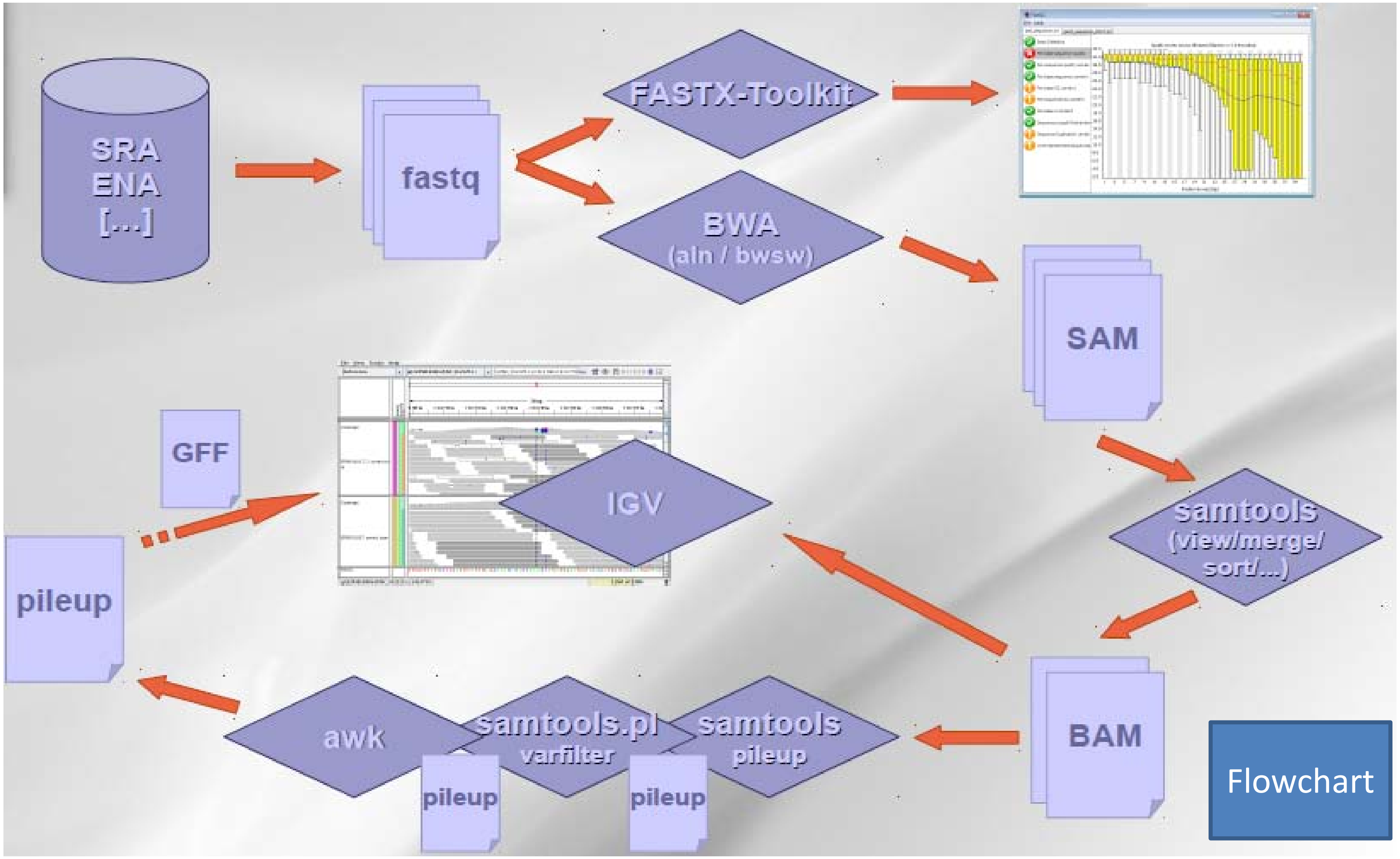


```
@HWUSI-EAS656_0037_FC:3:1:16637:1035#NNNNNN/1
CATATTTTGTGGCTCATCCCAAGGGAGAGGTTTTTCTATACTCAGGAGAAGTTACTCACGATAAAGAGAA
+
41?8FFF@@DAGGGEDF@FGECGGGBG@GE.EEBGBDADBBEEBEEC>ACE>CD?EEC?CAB>EB:BC##

@FC42RW0AAXX:3:1:2:1038#NNNNNN/1
GTGTTCTCTGCGACCCGTAATTCAGCTTTTTCCGGTTGCTTTGCCCTTTGCACCTTATCCTGCACCATCTCG
+
a]baaaa`aaaV`a_aa^Y^`_`_aa___`a]U__\\`][Z_^^R]YWWW[SWZ[QFY[VVWZWBBBBBBBBB

@I330_1_FC30JM6AAXX:4:1:13:1602/1
ATGTAGAAGTGTTTGATACGGCGATTTCAAACATTGCAGGGCTT
+I330_1_FC30JM6AAXX:4:1:13:1602/1
hhhhhhhhhhhhhhhhhhYh^hhhH[I>B^AABGDK;KBP??FN
```

What is resequencing?

- You have a reference genome
  - represents one individual
- You generate sequence from other individuals
  - same species / closely related species
- You want to identify variations
  - map millions of reads to reference genome
  - identify SNPs / indels / structural variations

Flowchart


SRA
ENA
[...]
fastq
FASTX-Toolkit
BWA
(aln / bwsw)
SAM
samtools
(view/merge/
sort/...)
BAM
IGV
GFF
samtools
pileup
pileup
samtools.pl
varfilter
pileup
awk
pileup
Flowchart

Steps in Resequencing

Map reads to a reference genome (.bam)

- finds best placement of reads

recalibrate alignments (.bam)

- realign indels
- remove duplicates
- recalibrate base quality

identify / call variants (.vcf)

- statistical algorithms to detect true variant

Steps in Resequencing

Map reads to a reference genome (.bam)

- finds best placement of reads

recalibrate alignments (.bam)

- realign indels
- remove duplicates
- recalibrate base quality

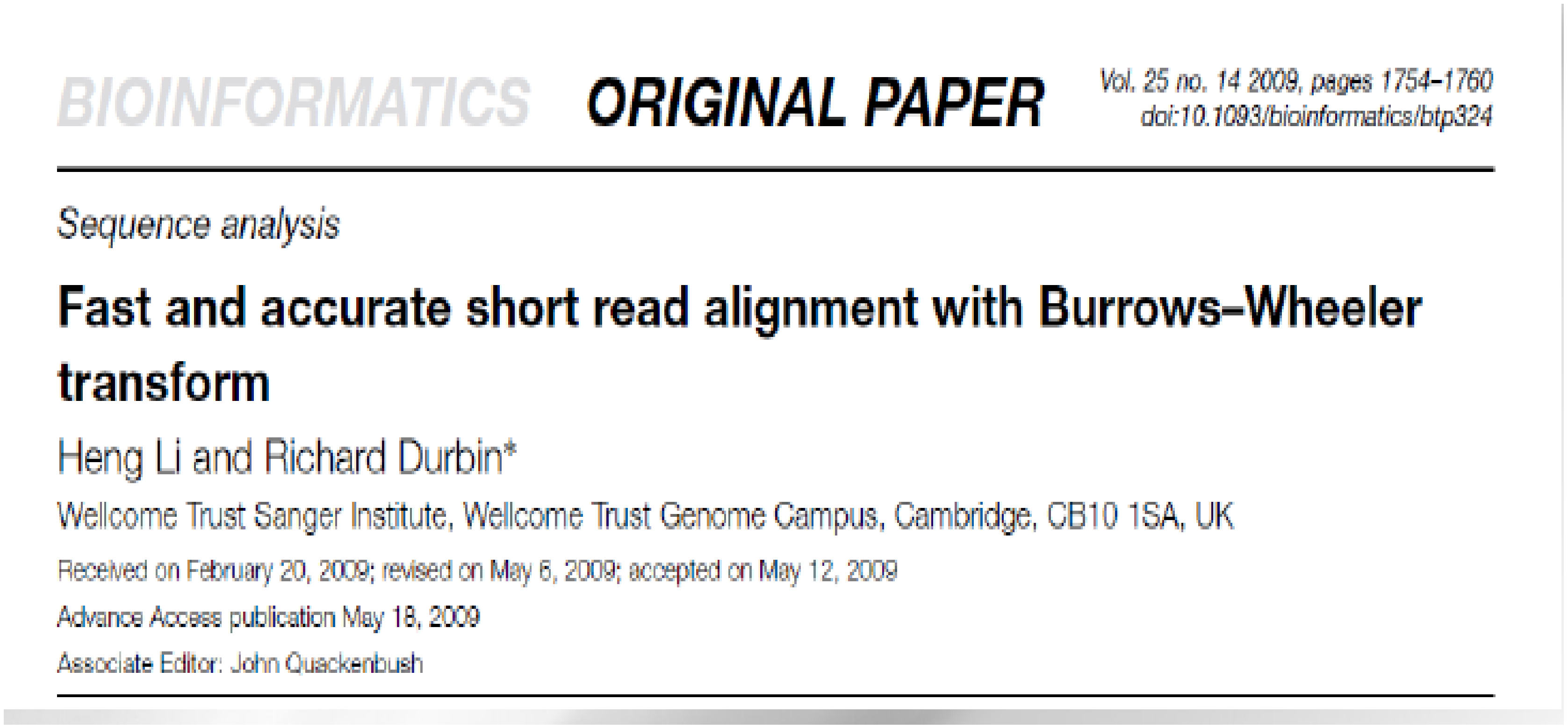
identify / call variants (.vcf)

- statistical algorithms to detect true variant

Step 1: map reads

- MAQ (http://maq.sourceforge.net/)
  - non-gapped
- BWA (http://bio-bwa.sourceforge.net/)
  - Burrows-Wheeler aligner
  - gapped (limited number of errors)
  - successor to Maq, but much faster than MAQ
- Bowtie (http://bowtie-bio.sourceforge.net)
  - fast + memory efficient
- Mosaik (http://bioinformatics.bc.edu/marthlab/)
  - Smith-Waterman

BWA



BIOINFORMATICS **ORIGINAL PAPER**

Vol. 25 no. 14 2009, pages 1754–1760
doi:10.1093/bioinformatics/btp324

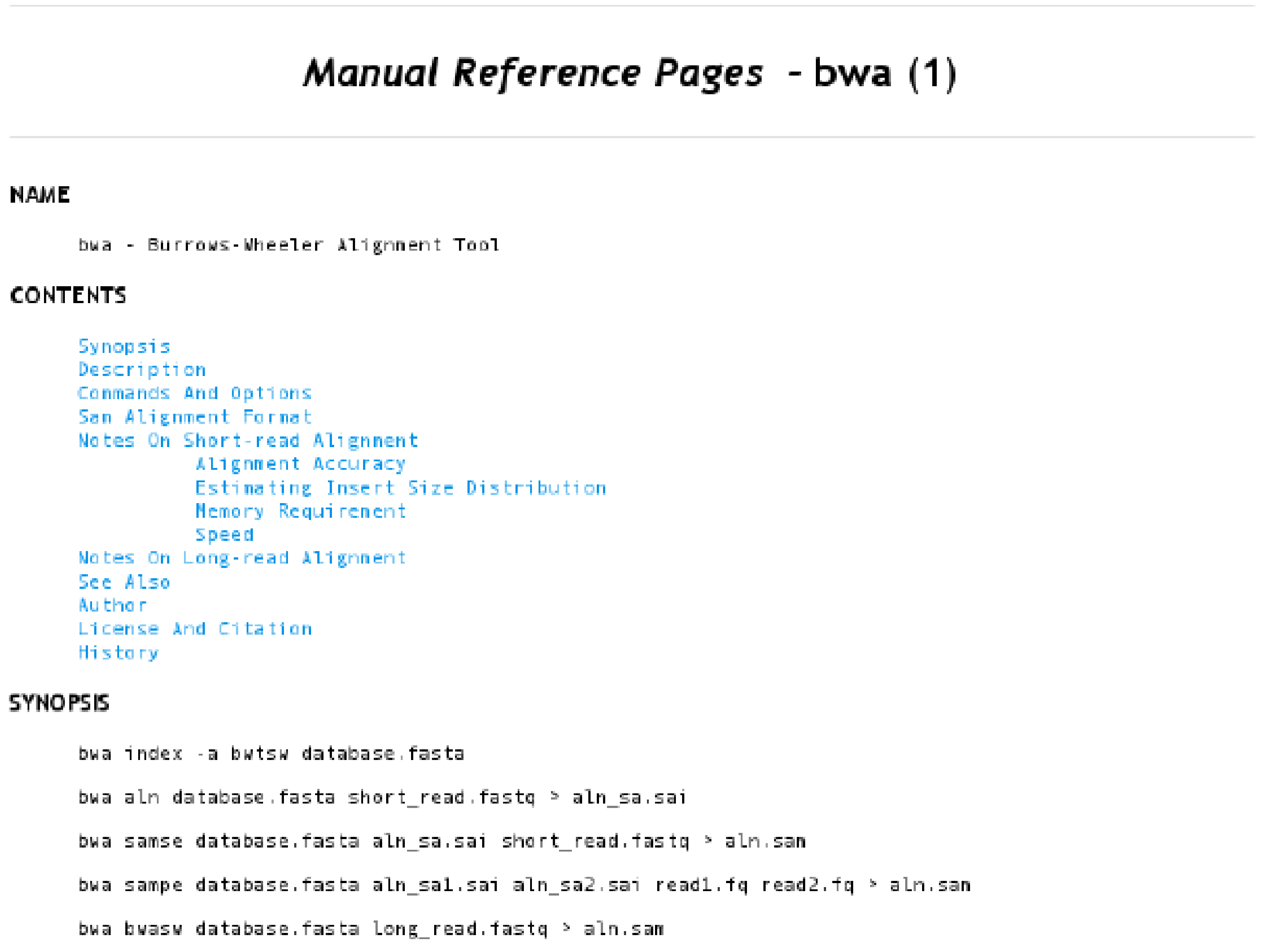
Sequence analysis

Fast and accurate short read alignment with Burrows–Wheeler transform

Heng Li and Richard Durbin*
Wellcome Trust Sanger Institute, Wellcome Trust Genome Campus, Cambridge, CB10 1SA, UK

Received on February 20, 2009; revised on May 6, 2009; accepted on May 12, 2009
Advance Access publication May 18, 2009
Associate Editor: John Quackenbush

bwa (1)



Manual Reference Pages - bwa (1)

NAME

```
bwa - Burrows-Wheeler Alignment Tool
```

CONTENTS

SYNOPSIS

```
bwa index -a bwtsw database.fasta

bwa aln database.fasta short_read.fastq > aln_sa.sai

bwa samse database.fasta aln_sa.sai short_read.fastq > aln.sam

bwa sampe database.fasta aln_sa1.sai aln_sa2.sai read1.fq read2.fq > aln.sam

bwa bwasw database.fasta long_read.fastq > aln.sam
```

bwa command line

- **Reference sequence indexing :**

  bwa index -a bwtsw db.fasta
- **Read Alignment :**

  bwa aln db.fasta short_read.fastq > aln_sa.sai

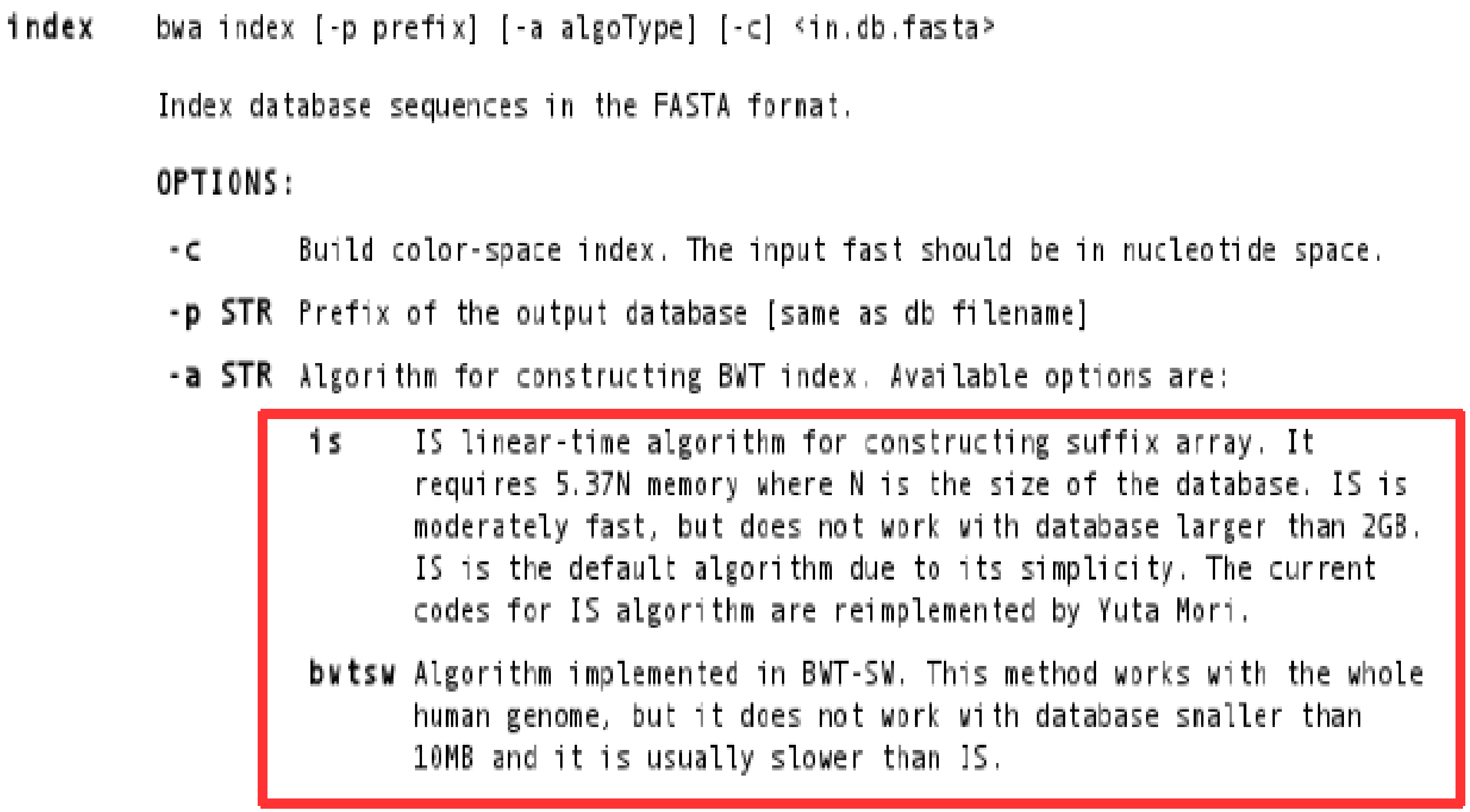
  bwa bwasw database.fasta long_read.fastq > aln.sam
- **Formatting unpaired reads (single-end) :**

  bwa samse db.fasta aln_sa.sai short_read.fastq >
  aln.sam
- **Formatting pair ends (paired-end):**

  bwa sampe database.fasta aln_sa1.sai aln_sa2.sai
  read1.fq

  read2.fq > aln.sam

bwa index



index bwa index [-p prefix] [-a algoType] [-c] <in.db.fasta>

Index database sequences in the FASTA format.

OPTIONS:

-c Build color-space index. The input fast should be in nucleotide space.

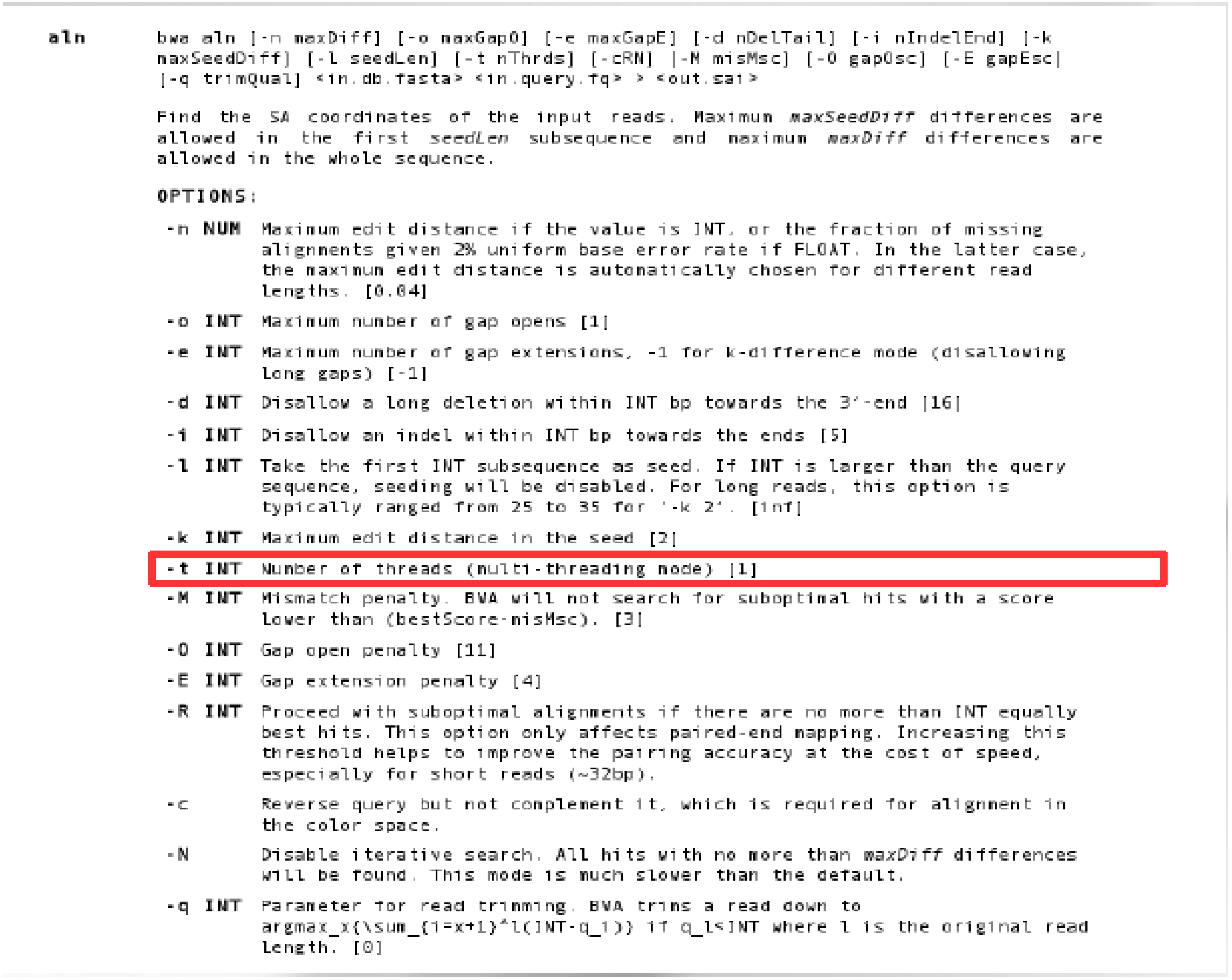
-p STR Prefix of the output database [same as db filename]

-a STR Algorithm for constructing BWT index. Available options are:

is IS linear-time algorithm for constructing suffix array. It requires 5.37N memory where N is the size of the database. IS is moderately fast, but does not work with database larger than 2GB. IS is the default algorithm due to its simplicity. The current codes for IS algorithm are reimplemented by Yuta Mori.

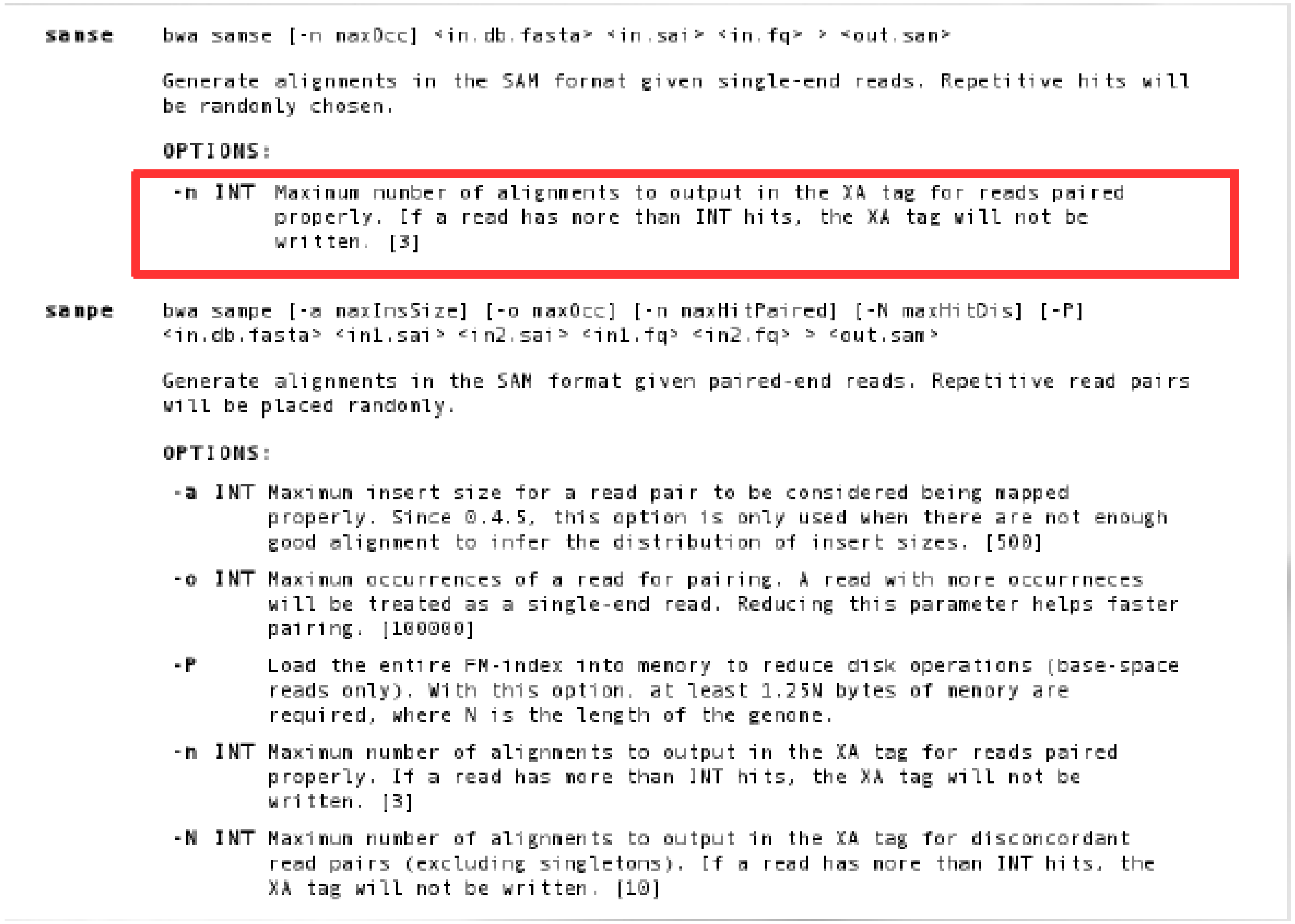
bwtsw Algorithm implemented in BWT-SW. This method works with the whole human genome, but it does not work with database smaller than 10MB and it is usually slower than IS.

bwa aln



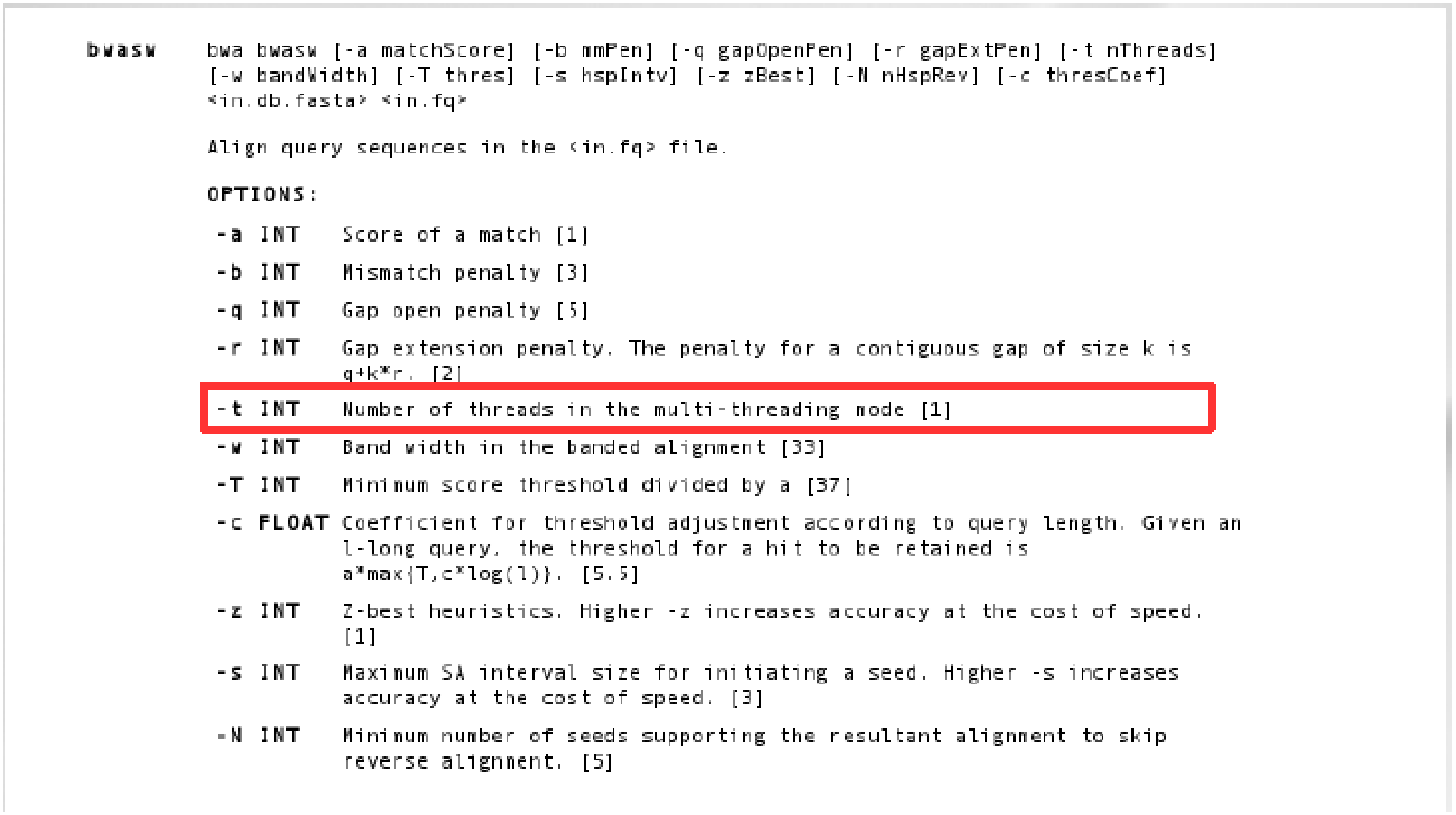
```
aln     bwa aln [-n maxDiff] [-o maxGapO] [-e maxGapE] [-d nDelTail] [-i nIndelEnd] [-k
        maxSeedDiff] [-l seedLen] [-t nThrds] [-cRN] [-M misMsc] [-O gapOsc] [-E gapEsc]
        [-q trimQual] <in.db.fasta> <in.query.fq> > <out.sai>

        Find the SA coordinates of the input reads. Maximum maxSeedDiff differences are
        allowed in the first seedLen subsequence and maximum maxDiff differences are
        allowed in the whole sequence.

        OPTIONS:

         -n NUM  Maximum edit distance if the value is INT, or the fraction of missing
                 alignments given 2% uniform base error rate if FLOAT. In the latter case,
                 the maximum edit distance is automatically chosen for different read
                 lengths. [0.04]

         -o INT  Maximum number of gap opens [1]

         -e INT  Maximum number of gap extensions, -1 for k-difference mode (disallowing
                 long gaps) [-1]

         -d INT  Disallow a long deletion within INT bp towards the 3'-end [16]

         -i INT  Disallow an indel within INT bp towards the ends [5]

         -l INT  Take the first INT subsequence as seed. If INT is larger than the query
                 sequence, seeding will be disabled. For long reads, this option is
                 typically ranged from 25 to 35 for '-k 2'. [inf]

         -k INT  Maximum edit distance in the seed [2]

         -t INT  Number of threads (multi-threading mode) [1]

         -M INT  Mismatch penalty. BWA will not search for suboptimal hits with a score
                 lower than (bestScore-misMsc). [3]

         -O INT  Gap open penalty [11]

         -E INT  Gap extension penalty [4]

         -R INT  Proceed with suboptimal alignments if there are no more than INT equally
                 best hits. This option only affects paired-end mapping. Increasing this
                 threshold helps to improve the pairing accuracy at the cost of speed,
                 especially for short reads (~32bp).

         -c      Reverse query but not complement it, which is required for alignment in
                 the color space.

         -N      Disable iterative search. All hits with no more than maxDiff differences
                 will be found. This mode is much slower than the default.

         -q INT  Parameter for read trimming. BWA trims a read down to
                 argmax_x{\sum_{i=x+1}^l(INT-q_i)} if q_l<INT where l is the original read
                 length. [0]
```

bwa samse & sampe



```
samse    bwa samse [-n maxOcc] <in.db.fasta> <in.sai> <in.fq> > <out.sam>

         Generate alignments in the SAM format given single-end reads. Repetitive hits will
         be randomly chosen.

         OPTIONS:

          -n INT  Maximum number of alignments to output in the XA tag for reads paired
                  properly. If a read has more than INT hits, the XA tag will not be
                  written. [3]

sampe    bwa sampe [-a maxInsSize] [-o maxOcc] [-n maxHitPaired] [-N maxHitDis] [-P]
         <in.db.fasta> <in1.sai> <in2.sai> <in1.fq> <in2.fq> > <out.sam>

         Generate alignments in the SAM format given paired-end reads. Repetitive read pairs
         will be placed randomly.

         OPTIONS:

          -a INT Maximum insert size for a read pair to be considered being mapped
                 properly. Since 0.4.5, this option is only used when there are not enough
                 good alignment to infer the distribution of insert sizes. [500]

          -o INT Maximum occurrences of a read for pairing. A read with more occurrneces
                 will be treated as a single-end read. Reducing this parameter helps faster
                 pairing. [100000]

          -P     Load the entire FM-index into memory to reduce disk operations (base-space
                 reads only). With this option, at least 1.25N bytes of memory are
                 required, where N is the length of the genome.

          -n INT Maximum number of alignments to output in the XA tag for reads paired
                 properly. If a read has more than INT hits, the XA tag will not be
                 written. [3]

          -N INT Maximum number of alignments to output in the XA tag for disconcordant
                 read pairs (excluding singletons). If a read has more than INT hits, the
                 XA tag will not be written. [10]
```

bwasw



```
bwasw    bwa bwasw [-a matchScore] [-b mmPen] [-q gapOpenPen] [-r gapExtPen] [-t nThreads]
         [-w bandWidth] [-T thres] [-s hspIntv] [-z zBest] [-N nHspRev] [-c thresCoef]
         <in.db.fasta> <in.fq>

         Align query sequences in the <in.fq> file.

         OPTIONS:

          -a INT    Score of a match [1]

          -b INT    Mismatch penalty [3]

          -q INT    Gap open penalty [5]

          -r INT    Gap extension penalty. The penalty for a contiguous gap of size k is
                    q+k*r. [2]

          -t INT    Number of threads in the multi-threading mode [1]

          -w INT    Band width in the banded alignment [33]

          -T INT    Minimum score threshold divided by a [37]

          -c FLOAT  Coefficient for threshold adjustment according to query length. Given an
                    l-long query, the threshold for a hit to be retained is
                    a*max{T,c*log(l)}. [5.5]

          -z INT    Z-best heuristics. Higher -z increases accuracy at the cost of speed.
                    [1]

          -s INT    Maximum SA interval size for initiating a seed. Higher -s increases
                    accuracy at the cost of speed. [3]

          -N INT    Minimum number of seeds supporting the resultant alignment to skip
                    reverse alignment. [5]
```

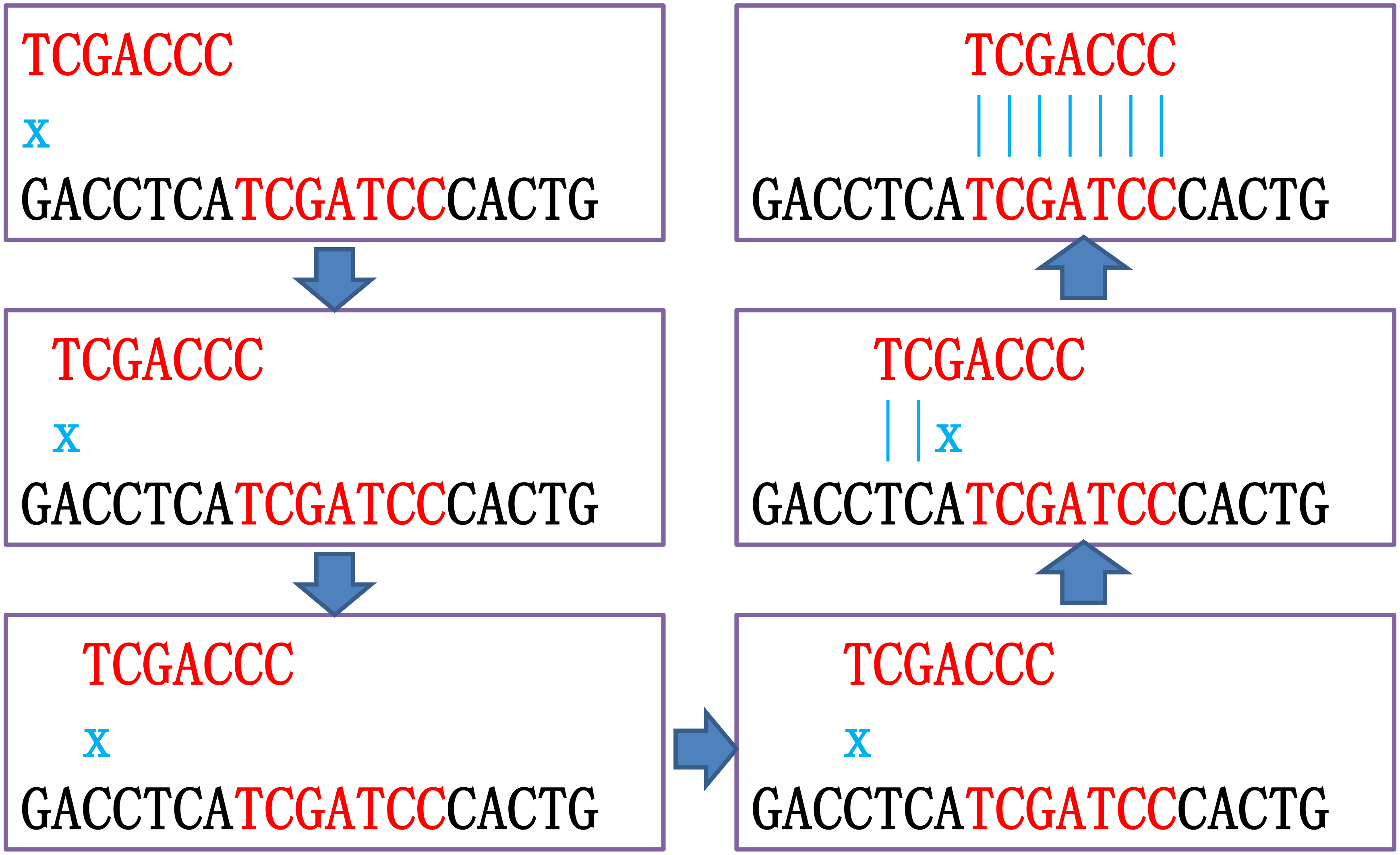
Exercise

- Data sets :
  - SRX002048
  - ERR003037
  - ERR000017
- Retrieving the reference sequence in fasta format :
  - gi|224581838|ref|**NC_012125.1| Salmonella enterica** subsp. enterica serovar Paratyphi C strain RKS4594, complete genome
- Indexing the reference sequence
- Aligning the reads (fastq format)
- Formatting the alignment in SAM

Mapping Algorithm trick

- brute force (simple)
- hash tables
- suffix trees
- Burrows-Wheeler transform (BWT)

Brute force



TCGACCC
x
GACCTCATCGATCCCACTG
TCGACCC
x
GACCTCATCGATCCCACTG
TCGACCC
x
GACCTCATCGATCCCACTG
TCGACCC
x
GACCTCATCGATCCCACTG
TCGACCC
| | x
GACCTCATCGATCCCACTG
TCGACCC
| | | | | | |
GACCTCATCGATCCCACTG

Hash table

Build an index of the reference sequence for fast access

seed length = 7

GACCTCATCGATCCCACTG

GACCTCA	□	chromosome 1, pos 0
ACCTCAT		chromosome 1, pos 1
CCTCATC		chromosome 1, pos 2
CTCATCG	□	chromosome 1, pos 3
TCATCGA	□	chromosome 1, pos 4
CATCGAT		chromosome 1, pos 5
ATCGATC		chromosome 1, pos 6
TCGATCC	□	chromosome 1, pos 7
CGATCCC		chromosome 1, pos 8
GATCCCA		chromosome 1, pos 9

Hash table

Build an index of the reference sequence for fast access

TCGATCC=?

seed length = 7

GACCTCATCGATCCCACTG

GACCTCA	□	chromosome 1, pos 0
ACCTCAT		chromosome 1, pos 1
CCTCATC		chromosome 1, pos 2
CTCATCG	□	chromosome 1, pos 3
TCATCGA	□	chromosome 1, pos 4
CATCGAT		chromosome 1, pos 5
ATCGATC		chromosome 1, pos 6
TCGATCC	□	chromosome 1, pos 7
CGATCCC		chromosome 1, pos 8
GATCCCA		chromosome 1, pos 9

Hash table

Build an index of the reference sequence for fast access

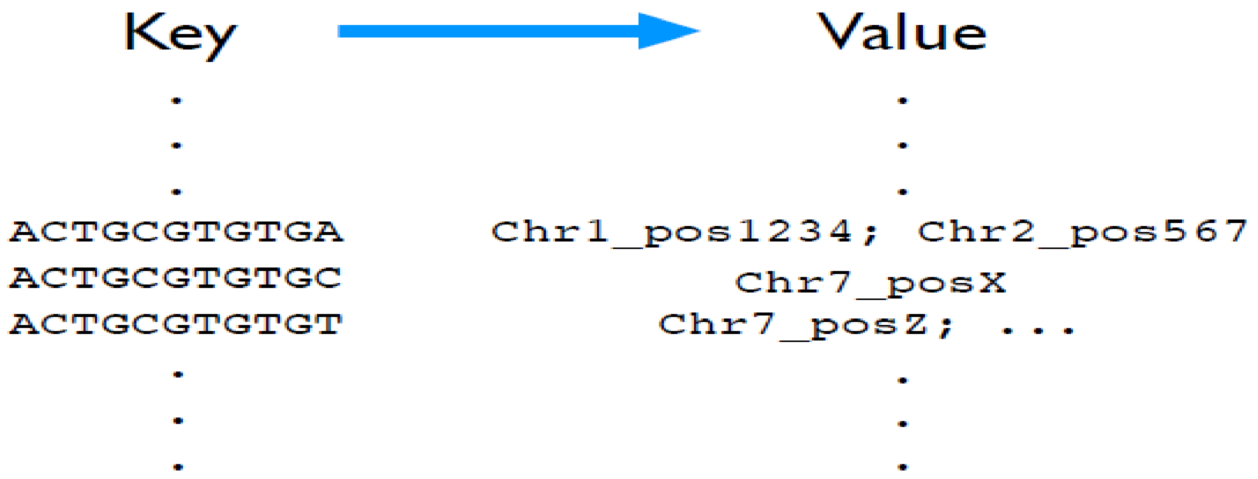
TCGATCC=chromosome 1, pos7

seed length = 7

GACCTCATCGATCCCACTG

GACCTCA	□	chromosome 1, pos 0
ACCTCAT		chromosome 1, pos 1
CCTCATC		chromosome 1, pos 2
CTCATCG	□	chromosome 1, pos 3
TCATCGA	□	chromosome 1, pos 4
CATCGAT		chromosome 1, pos 5
ATCGATC		chromosome 1, pos 6
TCGATCC	□	chromosome 1, pos 7
CGATCCC		chromosome 1, pos 8
GATCCCA		chromosome 1, pos 9

Hash-based algorithms



- Since lookups in hashes are fast!!!

1. Index the reference using *k*-mers
2. Search reads vs. hash *k*-mers
3. Perform alignment of entire read around seed
4. Report best alignment

Hash table: improvement

- **Spaced seed**: increasing sensitivity

- **Multiple seeds**: instead of extending around a single seed, but around positions with multiple seed matches (SHRiMP)

Hash tables

- BLAST
- BLAT, SSAHA (long read aligners)
- MAQ, SOAP (“older” short read aligners)
- Both nucleotide and color-space
  - BFAST
  - SHRiMP
  - Novoalign (commercial)
- partially by Mosaik

- Problem:
  - Memory exhaustive: Indexing big genomes/lists of reads requires lots of memory
  - Poor hashing leads to slow alignment

Burrows-Wheeler Transform (BWT)

- Hash-based aligners require lots of memory and are only reasonable fast
- Can we make it better/faster?
- BWT and suffix arrays
  - originally created for compression (implemented in bgzip2)
  - Aligners: BWA, Bowtie, Bowtie2, SOAPv2, bwa-sw
  - Low memory usage

The concepts

- BWT
  - A reversible transformation of the genome
  - All overlapping regions need only be searched once
- Suffix arrays: quickly find all possible matches
  - “array of integers giving the staring positions of suffixes of a string in lexicographical order”

BWT – creating index

Genome

Marks end-of-string, lexicographically smallest

```
X  =  AGGAGC$
```

BWT – creating index

1. Create all possible transformations of the string
(move first base to end)

```
X = AGGAGC$

  AGGAGC$
```

BWT – creating index

X = AGGAGC$

1. Create all possible transformations of the string
(move first base to end)

```
AGGAGC$
GGAGC$A
```

BWT – creating index

1. Create all possible transformations of the string
(move first base to end)

```
X  =  AGGAGC$

    AGGAGC$
    GGAGC$A
    GAGC$AG
```

BWT – creating index

1. Create all possible transformations of the string
(move first base to end)

```
X = AGGAGC$

    AGGAGC$
    GGAGC$A
    GAGC$AG
    AGC$AGG
    GC$AGGA
    C$AGGAG
    $AGGAGC
```

BWT – creating index

2. Sort the strings lexicographically

```
X = AGGAGC$

0   AGGAGC$
1   GGAGC$A
2   GAGC$AG
3   AGC$AGG
4   GC$AGGA
5   C$AGGAG
6   $AGGAGC
```

BWT – creating index

2. Sort the strings lexicographically

```
X = AGGAGC$

0   AGGAGC$            6   $AGGAG   C
1   GGAGC$A
2   GAGC$AG
3   AGC$AGG
4   GC$AGGA
5   C$AGGAG
6   $AGGAGC
```

BWT – creating index

3. Create the Suffix-Array (SA) and the BWT

X = AGGAGC$

0	AGGAGC$	6	$AGGAG	C
1	GGAGC$A	3	AGC$AG	G
2	GAGC$AG	0	AGGAGC	$
3	AGC$AGG	5	C$AGGA	G
4	GC$AGGA	2	GAGC$A	G
5	C$AGGAG	4	GC$AGG	A
6	$AGGAGC	1	GGAGC$	A

BWT – creating index

3. Create the Suffix-Array (SA) and the BWT

X = AGGAGC$

0	AGGAGC$
1	GGAGC$A
2	GAGC$AG
3	AGC$AGG
4	GC$AGGA
5	C$AGGAG
6	$AGGAGC

i	SA		BWT
0	6	$AGGAG	C
1	3	AGC$AG	G
2	0	AGGAGC	$
3	5	C$AGGA	G
4	2	GAGC$A	G
5	4	GC$AGG	A
6	1	GGAGC$	A

BWT – creating index

3. Create the Suffix-Array (SA) and the BWT

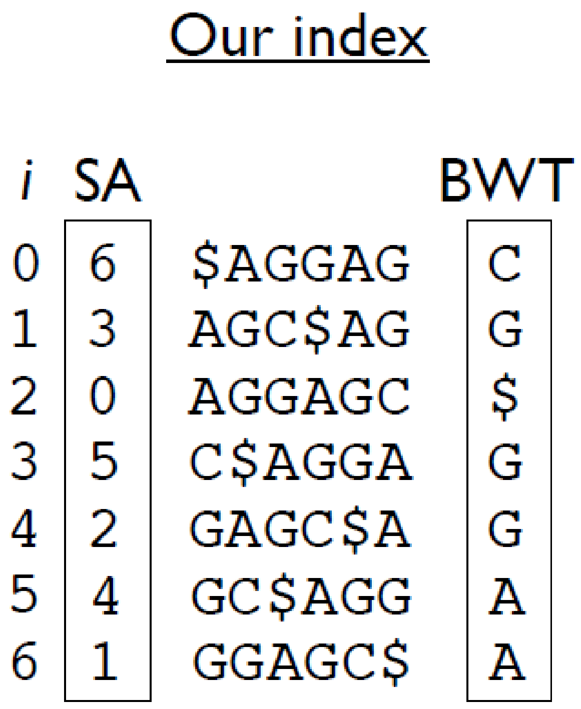
X = AGGAGC$

0	AGGAGC$
1	GGAGC$A
2	GAGC$AG
3	AGC$AGG
4	GC$AGGA
5	C$AGGAG
6	$AGGAGC

i	SA		BWT
0	6	$AGGAG	C
1	3	AGC$AG	G
2	0	AGGAGC	$
3	5	C$AGGA	G
4	2	GAGC$A	G
5	4	GC$AGG	A
6	1	GGAGC$	A

```
i   = (0,1,2,3,4,5,6)
SA  = (6,3,0,5,2,4,1)
BWT = CG$GGAA
```

BWT – align to index



Our index

i	SA		BWT
0	6	\$AGGAG	C
1	3	AGC\$AG	G
2	0	AGGAGC	\$
3	5	C\$AGGA	G
4	2	GAGC\$A	G
5	4	GC\$AGG	A
6	1	GGAGC\$	A

BWT – align to index

<u>Our index</u>

Read = "AG"

i	SA		BWT
0	6	\$AGGAG	C
1	3	AGC\$AG	G
2	0	AGGAGC	\$
3	5	C\$AGGA	G
4	2	GAGC\$A	G
5	4	GC\$AGG	A
6	1	GGAGC\$	A

BWT – align to index

<u>Our index</u>

i	SA		BWT
0	6	$AGGAG	C
1	3	AGC$AG	G
2	0	AGGAGC	$
3	5	C$AGGA	G
4	2	GAGC$A	G
5	4	GC$AGG	A
6	1	GGAGC$	A

Read = "AG"

Which strings starts with "AG"?

BWT – align to index

<u>Our index</u>

	i	SA		BWT
	0	6	$AGGAG	C
→	1	3	$\boxed{\text{AGC\$AG}}$	G
→	2	0	$\boxed{\text{AGGAGC}}$	$
	3	5	C$AGGA	G
	4	2	GAGC$A	G
	5	4	GC$AGG	A
	6	1	GGAGC$	A

Read = "AG"

Which strings starts with "AG"?

Get Suffix Array Indices: *i* = [1,2]

BWT – align to index

Our index

	i	SA		BWT
	0	6	$AGGAG	C
→	1	3	AGC$AG	G
→	2	0	AGGAGC	$
	3	5	C$AGGA	G
	4	2	GAGC$A	G
	5	4	GC$AGG	A
	6	1	GGAGC$	A

Read = "AG"

Which strings starts with "AG"?

Get Suffix Array Indices: *i* = [1,2]

Suffix Array values : SA[*i*] = [3,0]

BWT – align to index

<u>Our index</u>

	i	SA		BWT
	0	6	$AGGAG	C
→	1	3	AGC$AG	G
→	2	0	AGGAGC	$
	3	5	C$AGGA	G
	4	2	GAGC$A	G
	5	4	GC$AGG	A
	6	1	GGAGC$	A

Read = "AG"

Which strings starts with "AG"?

Get Suffix Array Indices: *i* = [1,2]

Suffix Array values : SA[*i*] = [3,0]

= read aligns at pos 0 & 3

BWT – align to index

<u>Our index</u>

	i	SA		BWT
	0	6	$AGGAG	C
→	1	3	$\boxed{\text{AGC\$AG}}$	G
→	2	0	$\boxed{\text{AGGAGC}}$	$
	3	5	C$AGGA	G
	4	2	GAGC$A	G
	5	4	GC$AGG	A
	6	1	GGAGC$	A

Read = "AG"

Which strings starts with "AG"?

Get Suffix Array Indices: *i* = [1,2]

Suffix Array values : SA[*i*] = [3,0]

= read aligns at pos 0 & 3

```
pos  0:  AGGAGC
pos  3:  AGGAGC
```

```
         1 1
0123456789 0 1
S =agcagcagact$
```

	suffix#	BWT(S)	suffix/rotation	
0	11	t	$agcagcagact	$...
1	8	g	act$agcagcag	a...
2	6	c	agact$agcagc	
3	3	c	agcagact$agc	
4	0	$	agcagcagact$	
5	5	g	cagact$agcag	c...
6	2	g	cagcagact$ag	
7	9	a	ct$agcagcaga	
8	7	a	gact$agcagca	g...
9	4	a	gcagact$agca	
10	1	a	gcagcagact$a	
11	10	c	t$agcagcagac	t...

BWT(agcagcagact) = tgcc$ggaaaac

ch	$	a	c	g	t
rank(ch)	0	1	5	8	11

Recovering s from bwt(s)

```
function recover(bwt)
 { // Recover original string from its transform
   var pos = 0,
       ans = endChar;  // $-terminated here
   for( var i = 1; i < bwt.length; i++ )
    { ans = bwt.charAt(pos) + ans;
      pos = inverse(pos, bwt);
    }
   return ans;
 }//recover(BWT)

function inverse(pos, bwt)
 { // one step of the reverse reconstruction
   var ch     = bwt.charAt(pos);
   var chCode = ch.charCodeAt(0);
   return rank[chCode] + occ(ch, bwt, pos);
 }//inverse(pos,bwt)

function occ(ch, bwt, pos)
  // returns the # of occurences of ch in bwt
  // before position pos
```

occ() function

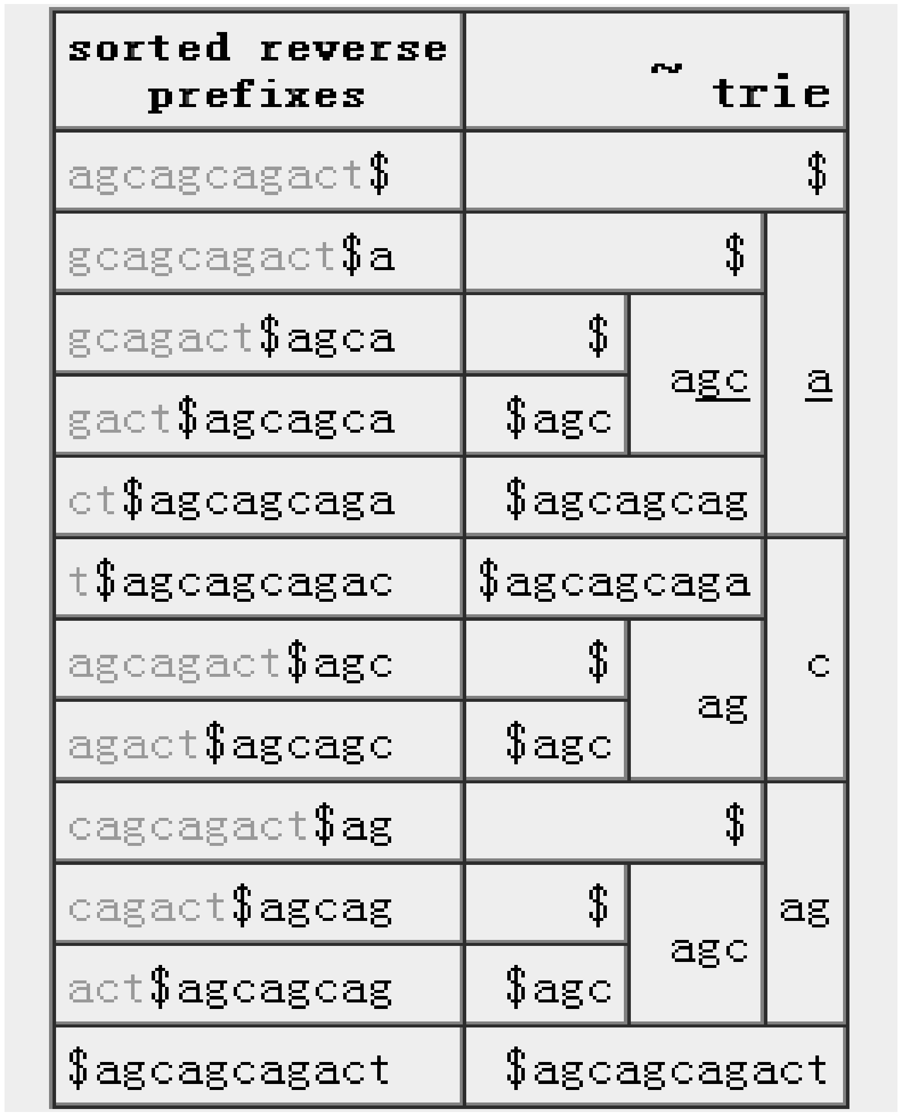
```
function occSlow(ch, bwt, i)  // (SLOW, but see occFast)
 { var count = 0;
   for( var j = 0; j < i; j++ )
      if( bwt.charAt(j) == ch ) count++;
   return count;
 }//occSlow(ch,bwt,i)
```

```
function occFast(ch, bwt, loc)
 { if( loc < 0 ) return 0;
   var bucket = Math.floor(loc/freqBucketSize);
   var lo = bucket * freqBucketSize;
   var count = freqCache[bucket][ch.charCodeAt(0)];
   for(var j = lo; j < loc; j++ )
      if( bwt.charAt(j) == ch ) count ++ ;
   return count;
 }//occFast(ch,bwt,loc)
```

Multiplicity

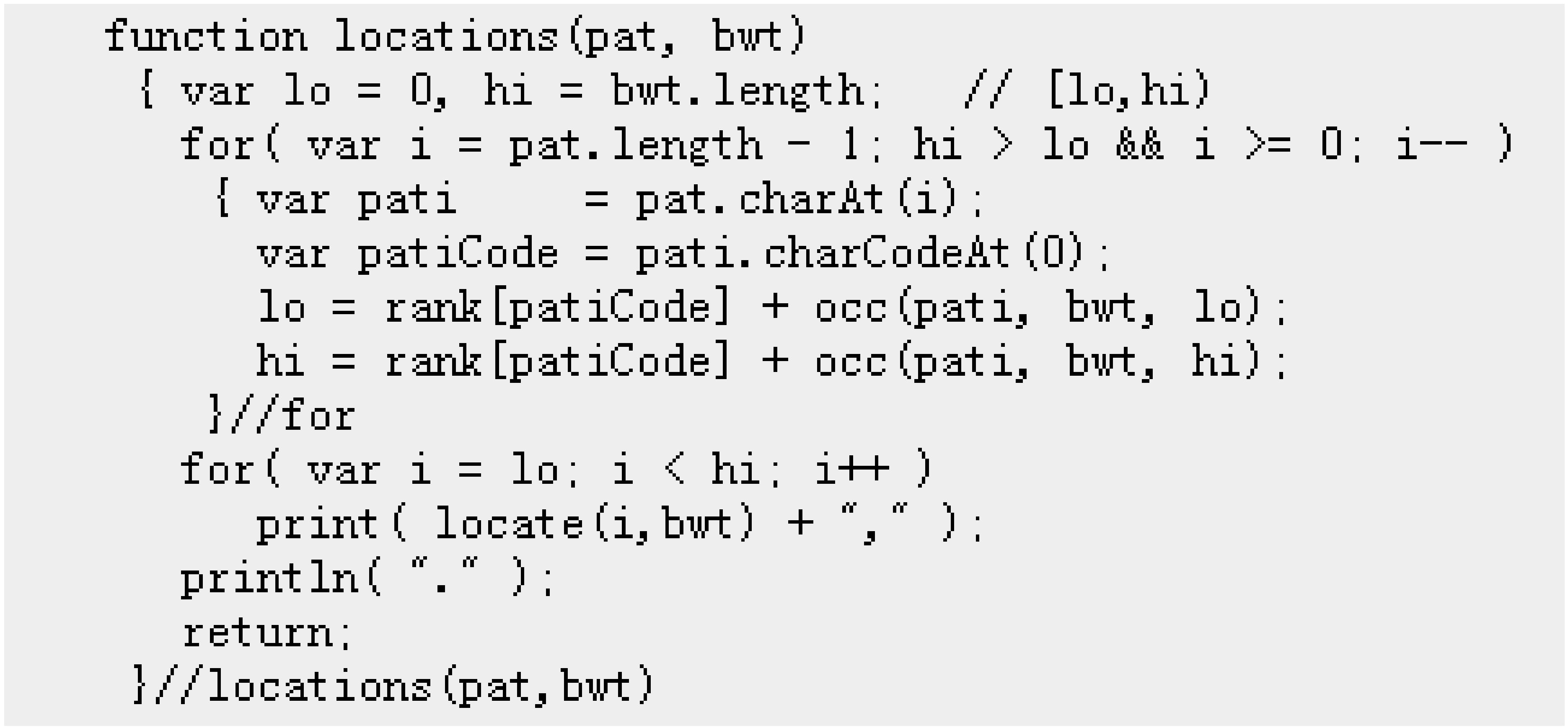
```
function multiplicity(pat, bwt)
 { // Return the number of times, if any, that pat occurs
   // in refStr where bwt is the transform of refStr.
   var lo = 0, hi = bwt.length;  // i.e. [lo,hi)
   for( var i = pat.length - 1; hi = lo && i >= 0; i-- )
    { var pati     = pat.charAt(i);
      var patiCode = pati.charCodeAt(0);
      lo = rank[patiCode] + occ(pati, bwt, lo);
      hi = rank[patiCode] + occ(pati, bwt, hi);
    }//for
   return hi - lo;
 }//multiplicity(pat,bwt)
```

trie



sorted reverse prefixes	~ trie		
agcagcagact$	$		
gcagcagact$a	$		a
gcagact$agca	$	agc	a
gact$agcagca	$agc	agc	a
ct$agcagcaga	$agcagcag		a
t$agcagcagac	$agcagcaga		c
agcagact$agc	$	ag	c
agact$agcagc	$agc	ag	c
cagcagact$ag	$		ag
cagact$agcag	$	agc	ag
act$agcagcag	$agc	agc	ag
$agcagcagact	$agcagcagact		

search for pattern



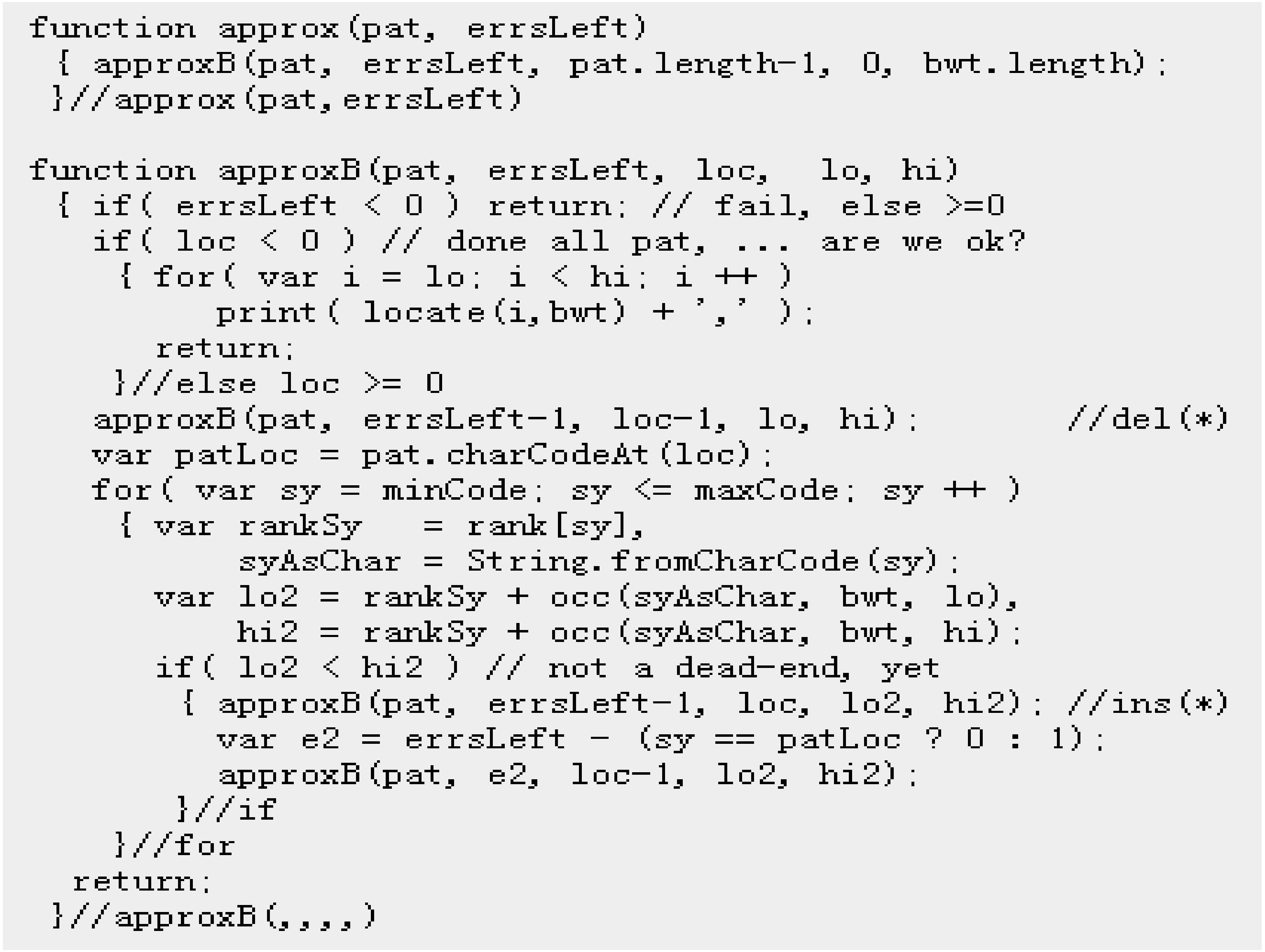
```
   function locations(pat, bwt)
    { var lo = 0, hi = bwt.length;   // [lo,hi)
      for( var i = pat.length - 1; hi > lo && i >= 0; i-- )
       { var pati     = pat.charAt(i);
         var patiCode = pati.charCodeAt(0);
         lo = rank[patiCode] + occ(pati, bwt, lo);
         hi = rank[patiCode] + occ(pati, bwt, hi);
       }//for
      for( var i = lo; i < hi; i++ )
         print( locate(i,bwt) + “,“ );
      println( “.“ );
      return;
    }//locations(pat,bwt)
```

locate() function

```
function locateSlow(pos, bwt)
 { pos = inverse(pos, bwt);
   var count;
   for( count = 0; pos > 0; count ++ )
      pos = inverse(pos, bwt);
   return count;
 }//locateSlow(pos,bwt)
```

```
function locateFast(pos, bwt)   // Fast
 { var count;
   for( count = 0; pos % saBucketSize > 0; count++ )
      pos = inverse(pos, bwt);
   return (count+saCache[pos/saBucketSize]) % bwt.length;
 }//locateFast(pos,bwt)
```

approximate search



```
function approx(pat, errsLeft)
 { approxB(pat, errsLeft, pat.length-1, 0, bwt.length);
 }//approx(pat,errsLeft)

function approxB(pat, errsLeft, loc,  lo, hi)
 { if( errsLeft < 0 ) return; // fail, else >=0
   if( loc < 0 ) // done all pat, ... are we ok?
    { for( var i = lo; i < hi; i ++ )
         print( locate(i,bwt) + ',' );
      return;
    }//else loc >= 0
   approxB(pat, errsLeft-1, loc-1, lo, hi);       //del(*)
   var patLoc = pat.charCodeAt(loc);
   for( var sy = minCode; sy <= maxCode; sy ++ )
    { var rankSy   = rank[sy],
          syAsChar = String.fromCharCode(sy);
      var lo2 = rankSy + occ(syAsChar, bwt, lo),
          hi2 = rankSy + occ(syAsChar, bwt, hi);
      if( lo2 < hi2 ) // not a dead-end, yet
       { approxB(pat, errsLeft-1, loc, lo2, hi2); //ins(*)
         var e2 = errsLeft - (sy == patLoc ? 0 : 1);
         approxB(pat, e2, loc-1, lo2, hi2);
       }//if
    }//for
  return;
 }//approxB(,,,,)
```

Choosing aligners

- Illumina: BWA, Bowtie2
- Solid: SHRiMP
- Ion Torrent: BWA-SW, (TMAP)
- 454: BWA-SW, (gsMapper)
- PacBio: BWA-SW, (BLASR)

Coverage

- Coverage/depth is how many times that your reads covers the genome (on average)

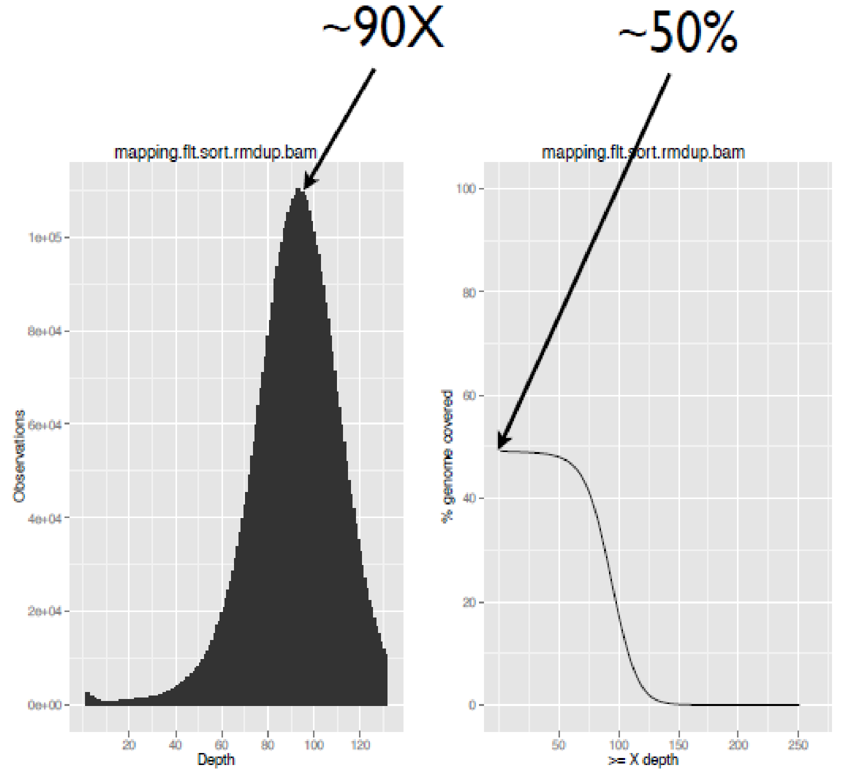
- Example:
  - N: number of reads, 5 M
  - L: read length, 100
  - G: Genome size, 5M
  - C = 5*100/5 = 100X
  - On average there are 100 reads covering each position in the genome

$$\boxed{C = \frac{N \times L}{G}}$$

Actual Depth

- How much do we actually cover?

- Avg.depth ~ 90X
- Range from 0-250X
- Only 50% of the genome was covered with reads



Steps in Resequencing

Map reads to a reference genome (.bam)

- finds best placement of reads

recalibrate alignments (.bam)

- realign indels
- remove duplicates
- recalibrate base quality

identify / call variants (.vcf)

- statistical algorithms to detect true variant

Step 2: recalibration

- realign indels
- remove duplicates
- recalibrate base quality

software

- some very useful programs for manipulation of short reads and alignments
- samtools (http://samtools.sourceforge.net/)
  - provides various utilities for manipulating alignments in the SAM and BAM format, including sorting, merging, indexing and generating alignments in a per-position format
- Picard (http://picard.sourceforge.net/)
  - Java-based command-line that manipulates SAM and BAM files

- Genome Analysis Toolkit (GATK, http://www.broadinstitute.org/gatk)
  - for variant discovery and genotyping as well as strong emphasis on data quality assurance (QA)

- Integrative genomics viewer (IGV, http://www.broadinstitute.org/igv/)
  - visualizing mapped reads

SAM format

header section

```
@HD VN:1.0 SO:coordinate
@SQ SN:1 LN:249250621 AS:NCBI37 UR:file:/data/local/ref/GATK/human_g1k_v37.fasta M5:1b22b98cdeb4a9304cb5d48026a85128
@SQ SN:2 LN:243199373 AS:NCBI37 UR:file:/data/local/ref/GATK/human_g1k_v37.fasta M5:a0d9851da00400dec1098a9255ac712e
@SQ SN:3 LN:198022430 AS:NCBI37 UR:file:/data/local/ref/GATK/human_g1k_v37.fasta M5:fdfd811849cc2fadebc929bb925902e5
@RG ID:UM0098:1 PL:ILLUMINA PU:HWUSI-EAS1707-615LHAAXX-L001 LB:80 DT:2010-05-05T20:00:00-0400 SM:SD37743 CN:UMCORE
@RG ID:UM0098:2 PL:ILLUMINA PU:HWUSI-EAS1707-615LHAAXX-L002 LB:80 DT:2010-05-05T20:00:00-0400 SM:SD37743 CN:UMCORE
@PG ID:bwa VN:0.5.4
```

alignment section

```
8_96_444_1622 73 scaffold00005 155754 255 54M * 0 0 ATGTAAAGTATTTCCATGGTACACAGCTTGGTCGTAATGTGATTGCTGAGCCAG
BC@B5)5CBBCCBCCCBC@@7C>CBCCBCCC;57)8(@B@B>ABBCBC7BCC=> NM:i:0
8_80_1315_464 81 scaffold00005 155760 255 54M = 154948 0 AGTACCTCCCTGGTACACAGCTTGGTAAAAATGTGATTGCTGAGCCAGACCTTC
B?@?BA=>@>>7;ABA?BB@BAA;@BBBBBBAABABBBCABAB?BABA?BBBAB NM:i:0
8_17_1222_1577 73 scaffold00005 155783 255 40M1116N10M * 0 0 GGTAAAAATGTGATTGCTGAGCCAGACCTTCATCATGCAGTGAGAGACGC
BB@BA??>CCBA2AAABBBBBBB8A3@BABA;@A:>B=,;@B=A:BAAAA NM:i:0 XS:A:+ NS:i:0
8_43_1211_347 73 scaffold00005 155800 255 23M1116N27M * 0 0 TGAGCCAGACCTTCATCATGCAGTGAGAGACGCAAACATGCTGGTATTTG
#>8<=<@6/:@9';@7A@@BAAA@BABBBABBB@=<A@BBBBBBBBCCBB NM:i:2 XS:A:+ NS:i:0
8_32_1091_284 161 scaffold00005 156946 255 54M = 157071 0 CGCAAACATGCTGGTAGCTGTGACACCACATCAACAGCTTGACTATGTTTGTAA
BBBBB@AABACBCA8BBBBBABBBB@BBBBBBA@BBBBBBBBBA@:B@AA@=@@ NM:i:0
```

query **reference**

SAM Mandatory fields

- QNAME: Query name of the read or read pair
- FLAG: Bitwise flag (pairing, strand, mate strand, etc.)
- RNAME: Reference sequence name
- POS: 1-based left most position of clipped alignment
- MAPQ: Mapping quality (Phred-scaled)
- CIGAR: Extended CIGAR string (MIDNSHP)
- MRNM: Mate reference name (‘=’ if same as RNAME)
- MPOS: 1-based leftmost mate position
- ISIZE: Inferred insert size
- SEQQuery: Sequence on the same strand as the reference
- QUAL: Query quality (ASCII-33=Phred base quality)

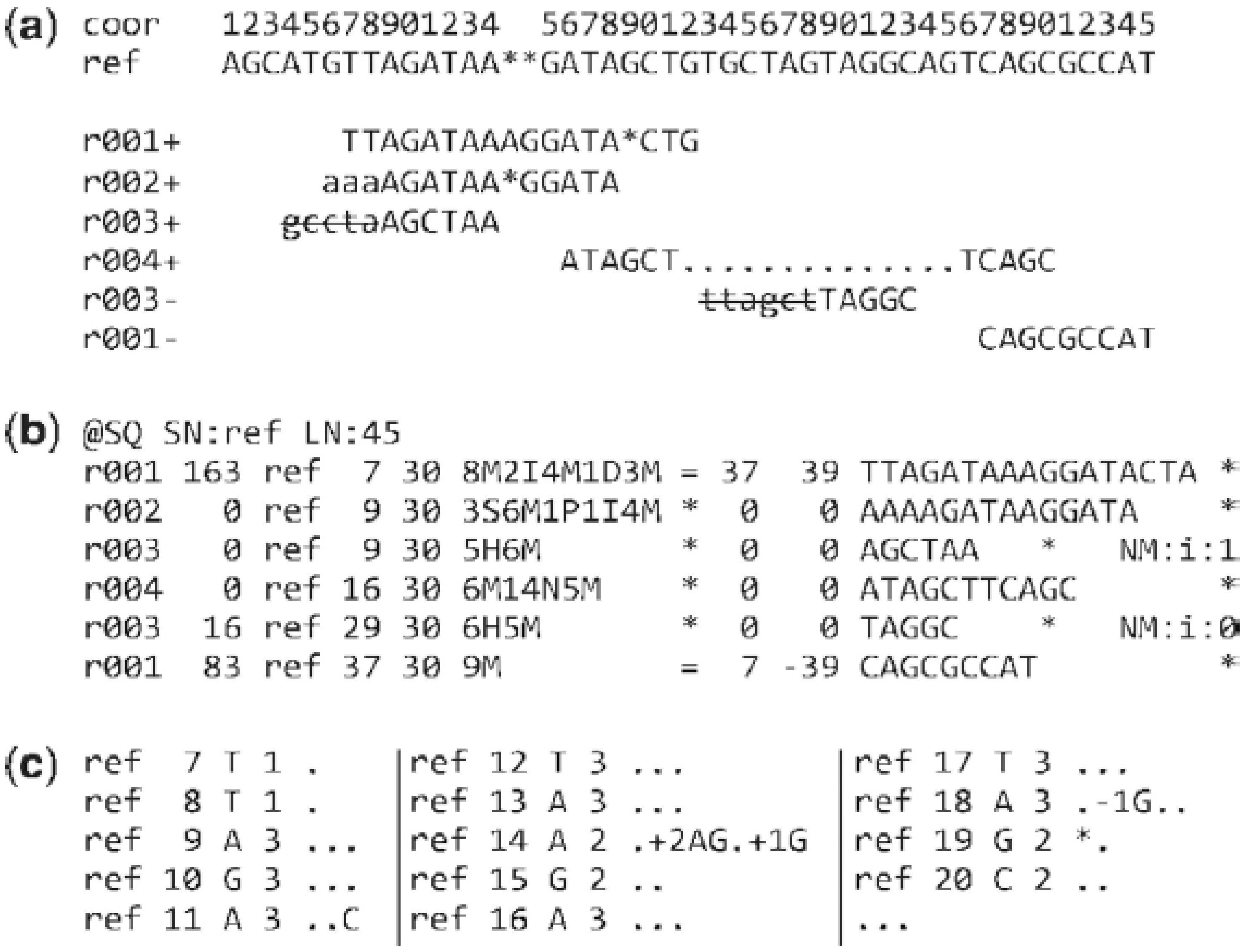
CIGAR operators

- **M: alignment match/mismatch**
- **I: insertion to the reference**
- **D: deletion from the reference**
- **S: softclip on the read (clipped sequence present in <seq>**
- **H: hardclip (clipped sequence NOT present in <seq>)**
- **P: padding (silent deletion from the padded reference)**
- **N: skipped region from the reference**

POS	**CIGAR**
5	2S4M2D6M3S

```
Reference:  GCATTCAGATGCAGTACGC
     Read:    ccTCAG--GCAGTAgtg
```

Example of extended CIGAR and pileup output



(a)
```
coor     12345678901234  5678901234567890123456789012345
ref      AGCATGTTAGATAA**GATAGCTGTGCTAGTAGGCAGTCAGCGCCAT

r001+          TTAGATAAAGGATA*CTG
r002+         aaaAGATAA*GGATA
r003+       ~~gccta~~AGCTAA
r004+                     ATAGCT..............TCAGC
r003-                             ~~ttagct~~TAGGC
r001-                                          CAGCGCCAT
```

(b)
```
@SQ SN:ref LN:45
r001 163 ref  7 30 8M2I4M1D3M = 37  39 TTAGATAAAGGATACTA *
r002   0 ref  9 30 3S6M1P1I4M *  0   0 AAAAGATAAGGATA    *
r003   0 ref  9 30 5H6M       *  0   0 AGCTAA   *   NM:i:1
r004   0 ref 16 30 6M14N5M    *  0   0 ATAGCTTCAGC       *
r003  16 ref 29 30 6H5M       *  0   0 TAGGC    *   NM:i:0
r001  83 ref 37 30 9M         =  7 -39 CAGCGCCAT         *
```

(c)
```
ref  7 T 1 .     |ref 12 T 3 ...          |ref 17 T 3 ...
ref  8 T 1 .     |ref 13 A 3 ...          |ref 18 A 3 .-1G..
ref  9 A 3 ...   |ref 14 A 2 .+2AG.+1G    |ref 19 G 2 *.
ref 10 G 3 ...   |ref 15 G 2 ..           |ref 20 C 2 ..
ref 11 A 3 ..C   |ref 16 A 3 ...          |...
```

Li H et al. Bioinformatics 2009;25:2078-2079

BAM format

- Binary representation of SAM
- Compressed by BGZF library
- Greatly reduces storage space requirement to about 27% of original SAM

samtools examples

- Create BAM from SAM

  samtools view –bS aln.sam -o aln.bam
- Sort BAM file

  samtools sort example.bam sortedExample
- Merge sorted BAM files
- samtools merge sortedMerge.bam sorted1.bam sorted2.bam
- Index BAM file

  samtools index sortedExample.bam
- Visualize BAM file

  samtools tview sortedExample.bam reference.fa

Steps in Resequencing

Map reads to a reference genome (.bam)

- finds best placement of reads

recalibrate alignments (.bam)

- realign indels
- remove duplicates
- recalibrate base quality

identify / call variants (.vcf)

- statistical algorithms to detect true variant

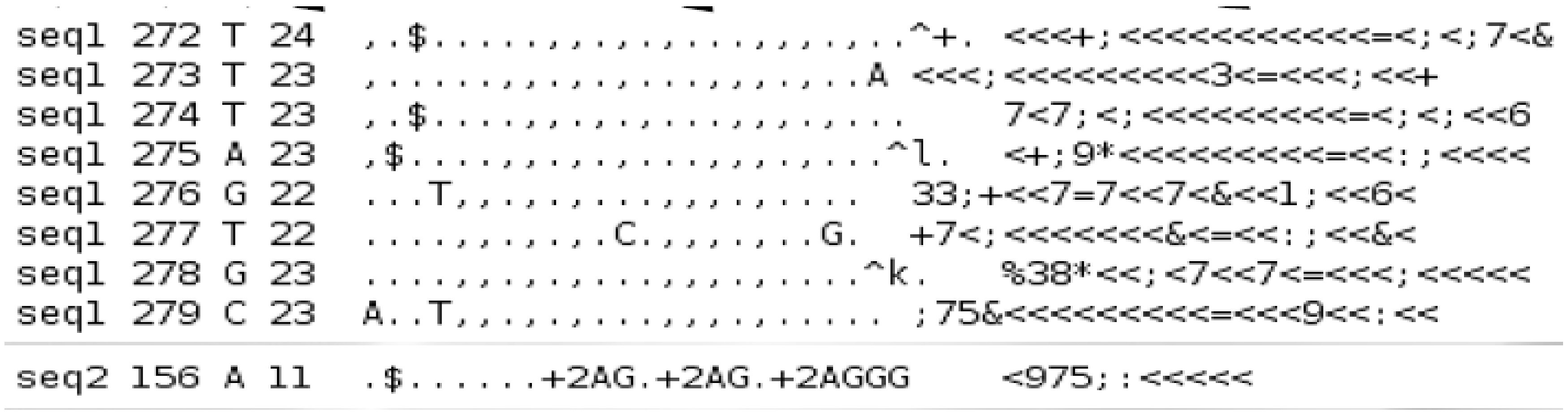
Variant calling with samtools

- Get the raw variant :
  ```
  samtools pileup vcf ref.fa aln.bam > raw.txt
  samtools view -u aln.bam X | samtools pileup vcf ref.fa > rawX.txt
  ```

ref.fa fasta
formatted file of the reference genome
aln.bam
Sorted BAM formatted file, from the alignments
raw[-X].txt
Output pileup formatted, with consensus calls
-c
Calls the consensus base at each position
-v
Show positions that do not agree with ref.fa
-f
Reference sequence, ref.fa (in fasta format)

The pileup format

chr - coord - base (* for indel) – number of reads cover the site - read bases* base quality

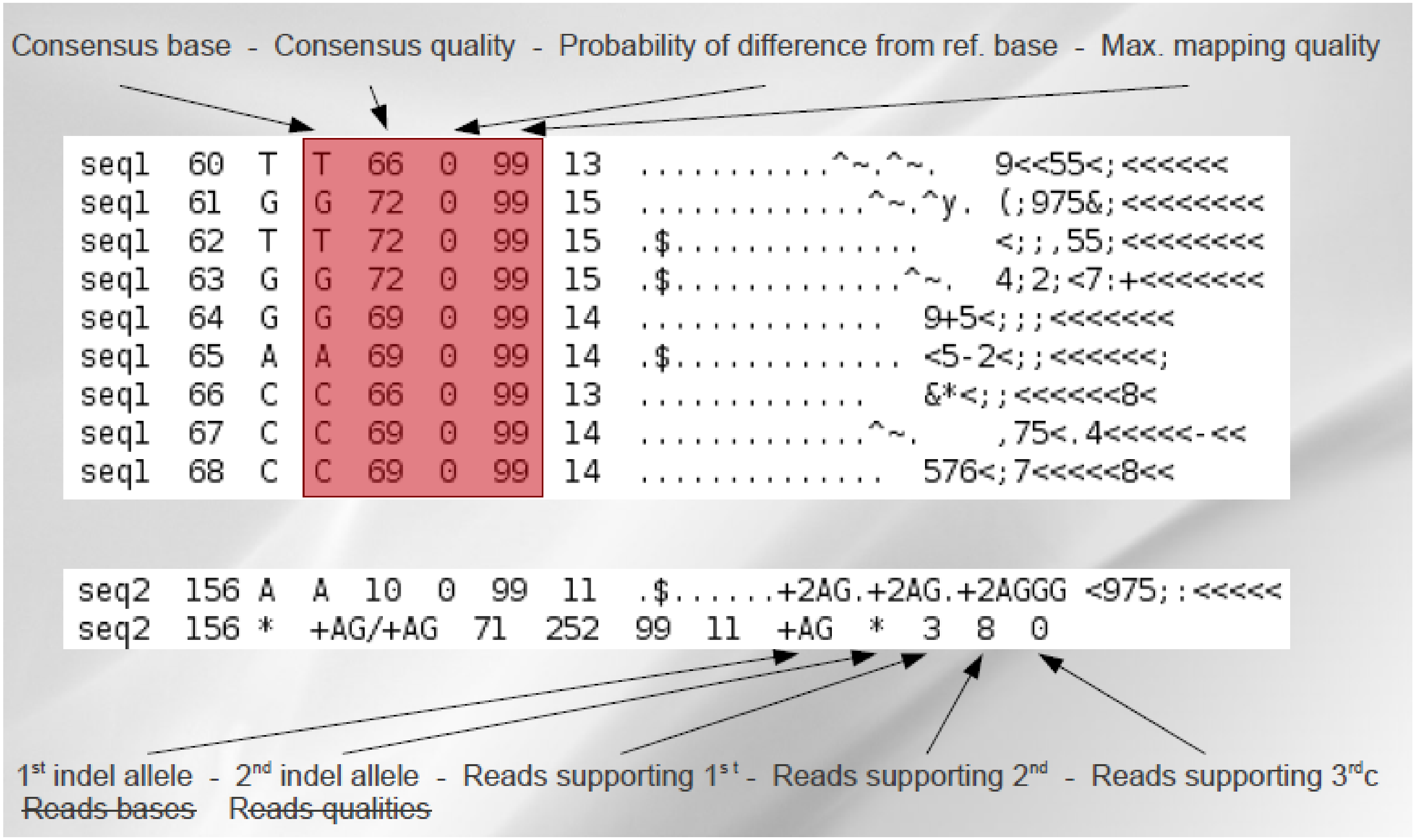


```
seq1 272 T 24  ,.$.....,,.,.,...,,,.,..^+. <<<+;<<<<<<<<<<<=<;<;7<&
seq1 273 T 23  ,.....,,.,.,...,,,.,..A <<<;<<<<<<<<<3<=<<<;<<+
seq1 274 T 23  ,.$....,,.,.,...,,,.,...    7<7;<;<<<<<<<<<=<;<;<<6
seq1 275 A 23  ,$....,,.,.,...,,,.,...^l.  <+;9*<<<<<<<<<=<<:;<<<<
seq1 276 G 22  ...T,,.,.,...,,,.,....  33;+<<7=7<<7<&<<1;<<6<
seq1 277 T 22  ....,,.,.,.C.,,,.,..G.  +7<;<<<<<<<&<=<<:;<<&<
seq1 278 G 23  ....,,.,.,...,,,.,....^k.   %38*<<;<7<<7<=<<<;<<<<<
seq1 279 C 23  A..T,,.,.,...,,,.,..... ;75&<<<<<<<<<=<<<9<<:<<
```

```
seq2 156 A 11  .$......+2AG.+2AG.+2AGGG    <975;:<<<<<
```

- **Read bases :**
- **'.' and ',' : match to the reference base on the forward/reverse strand**
- **'ACTGN' and 'actgn' : for a mismatch on the forward/reverse strand**
- **'^' and '$' : start/end of a read segment**
- **'+[0-9]+[ACGTNacgtn]+' and '-[0-9]+[ACGTNacgtn]+' : insertion/deletion**

The pileup format


Consensus base - Consensus quality - Probability of difference from ref. base - Max. mapping quality

```
seq1  60  T  T  66  0  99  13  ..........^~.^~.    9<<55<;<<<<<<
seq1  61  G  G  72  0  99  15  ............^~.^y.  (;975&;<<<<<<<<
seq1  62  T  T  72  0  99  15  .$..............    <;;,55;<<<<<<<<
seq1  63  G  G  72  0  99  15  .$.............^~.  4;2;<7:+<<<<<<<
seq1  64  G  G  69  0  99  14  ..............  9+5<;;;<<<<<<<
seq1  65  A  A  69  0  99  14  .$............  <5-2<;;<<<<<<;
seq1  66  C  C  66  0  99  13  .............   &*<;;<<<<<<8<
seq1  67  C  C  69  0  99  14  .............^~.    ,75<.4<<<<<-<<
seq1  68  C  C  69  0  99  14  ..............  576<;7<<<<<8<<
```

```
seq2  156 A  A  10  0  99  11  .$......+2AG.+2AG.+2AGGG <975;:<<<<<
seq2  156 *  +AG/+AG  71  252  99  11  +AG  *  3  8  0
```

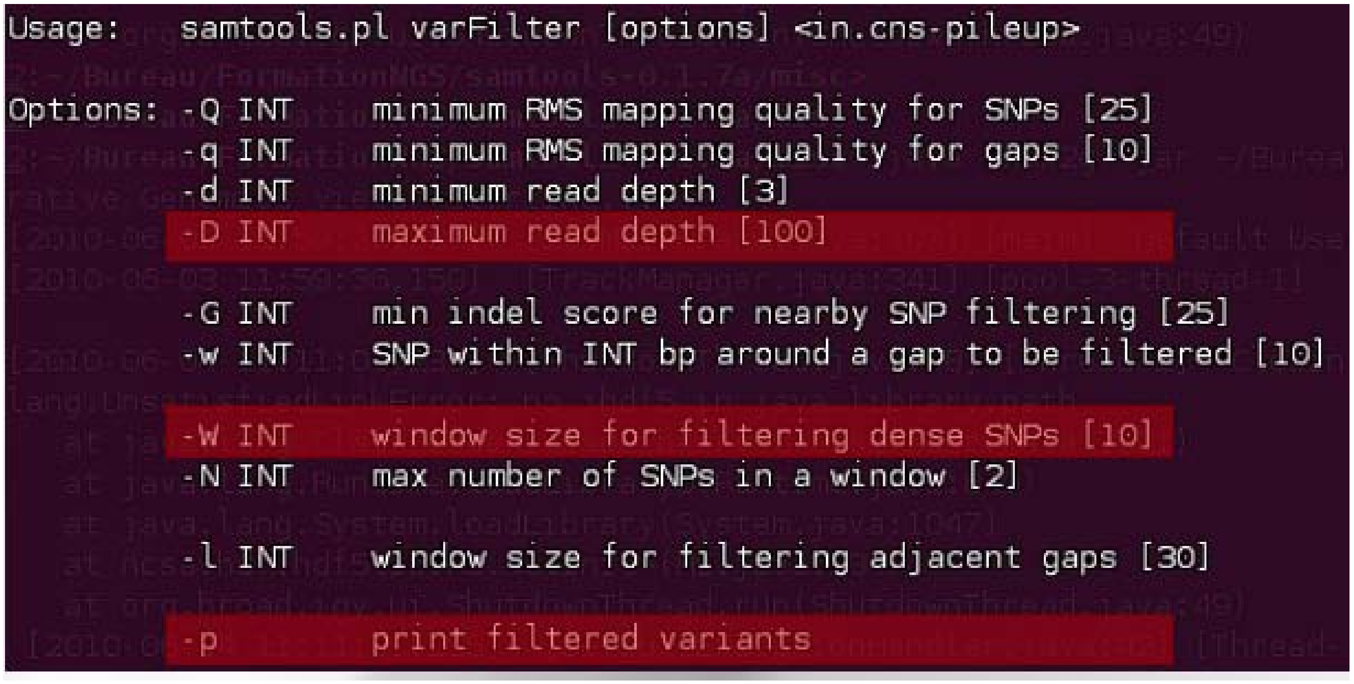
1st indel allele - 2nd indel allele - Reads supporting 1st - Reads supporting 2nd - Reads supporting 3rd c
~~Reads bases~~ ~~Reads qualities~~

samtools.pl: Filter

- Filter the raw variants calls:

  samtools.pl varFilter raw.txt > raw_ok.txt

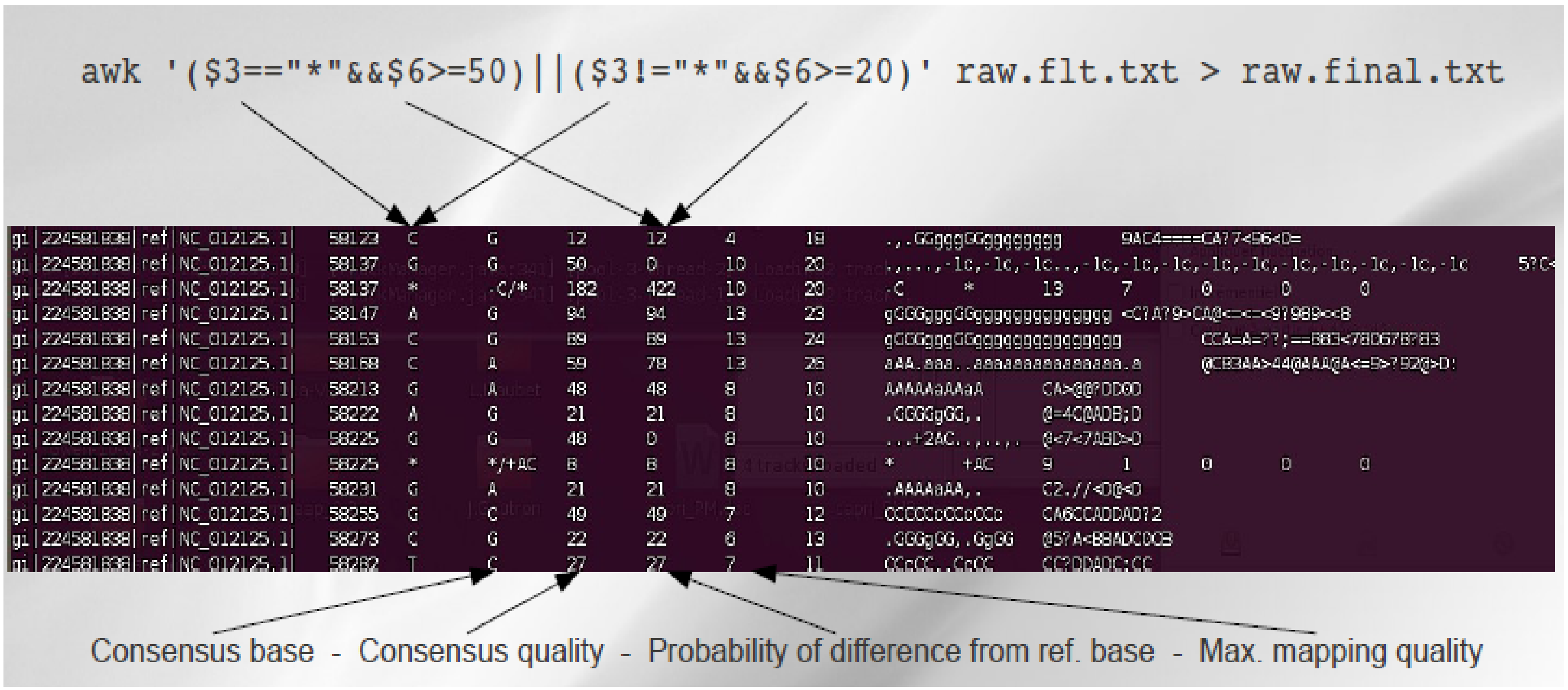
  (samtools.pl varFilter –p raw.txt > raw_ok.txt) >& raw_filtered.txt



```
Usage:   samtools.pl varFilter [options] <in.cns-pileup>

Options: -Q INT    minimum RMS mapping quality for SNPs [25]
         -q INT    minimum RMS mapping quality for gaps [10]
         -d INT    minimum read depth [3]
         -D INT    maximum read depth [100]

         -G INT    min indel score for nearby SNP filtering [25]
         -w INT    SNP within INT bp around a gap to be filtered [10]

         -W INT    window size for filtering dense SNPs [10]
         -N INT    max number of SNPs in a window [2]

         -l INT    window size for filtering adjacent gaps [30]

         -p        print filtered variants
```

awk: Filter

- use quality threshold to filter the final variant calls



```
awk '($3=="*"&&$6>=50)||($3!="*"&&$6>=20)' raw.flt.txt > raw.final.txt
```

```
gi|224581838|ref|NC_012125.1|   58123  C   G      12   12   4   18   .,.GGgggGGgggggggg   9AC4====CA?7<96<D=
gi|224581838|ref|NC_012125.1|   58137  G   G      50   0    10  20   .,...,-1c,-1c,-1c..,-1c,-1c,-1c,-1c,-1c,-1c,-1c,-1c,-1c,-1c   5?C<
gi|224581838|ref|NC_012125.1|   58137  *   -C/*   182  422  10  20   -C   *   13   7   0   0   0
gi|224581838|ref|NC_012125.1|   58147  A   G      94   94   13  23   gGGGgggGGgggggggggggggg <C?A?9>CA@<=<=<9?989<<B
gi|224581838|ref|NC_012125.1|   58153  C   G      89   89   13  24   gGGGgggGGgggggggggggggggg   CCA=A=??;==BB3<78D67B?B3
gi|224581838|ref|NC_012125.1|   58168  C   A      59   78   13  26   aAA.aaa..aaaaaaaaaaaaaaa.a   @CB3AA>44@AAA@A<=9>?92@>D:
gi|224581838|ref|NC_012125.1|   58213  G   A      48   48   8   10   AAAAAaAAaA   CA>@@?DD0D
gi|224581838|ref|NC_012125.1|   58222  A   G      21   21   8   10   .GGGGgGG,.   @=4C@ADB;D
gi|224581838|ref|NC_012125.1|   58225  G   G      48   0    8   10   ...+2AC..,...,.   @<7<7ABD>D
gi|224581838|ref|NC_012125.1|   58225  *   */+AC  8    8    8   10   *   +AC   9   1   0   0   0
gi|224581838|ref|NC_012125.1|   58231  G   A      21   21   8   10   .AAAAaAA,.   C2.//<D@<D
gi|224581838|ref|NC_012125.1|   58255  G   C      49   49   7   12   CCCCCcCCcCCc   CA6CCADDAD?2
gi|224581838|ref|NC_012125.1|   58273  C   G      22   22   6   13   .GGGgGG,.GgGG   @5?A<BBADCDCB
gi|224581838|ref|NC_012125.1|   58282  T   C      27   27   7   11   CCcCC,.CcCC   CC?DDADC;CC
```

Consensus base - Consensus quality - Probability of difference from ref. base - Max. mapping quality

Generating consensus

- **Consensus** : A way of representing the results of a multiple sequence alignment (which residues are most abundant in the alignment at each position).

```
samtools.pl pileup2fq raw.txt > raw.fastq
```