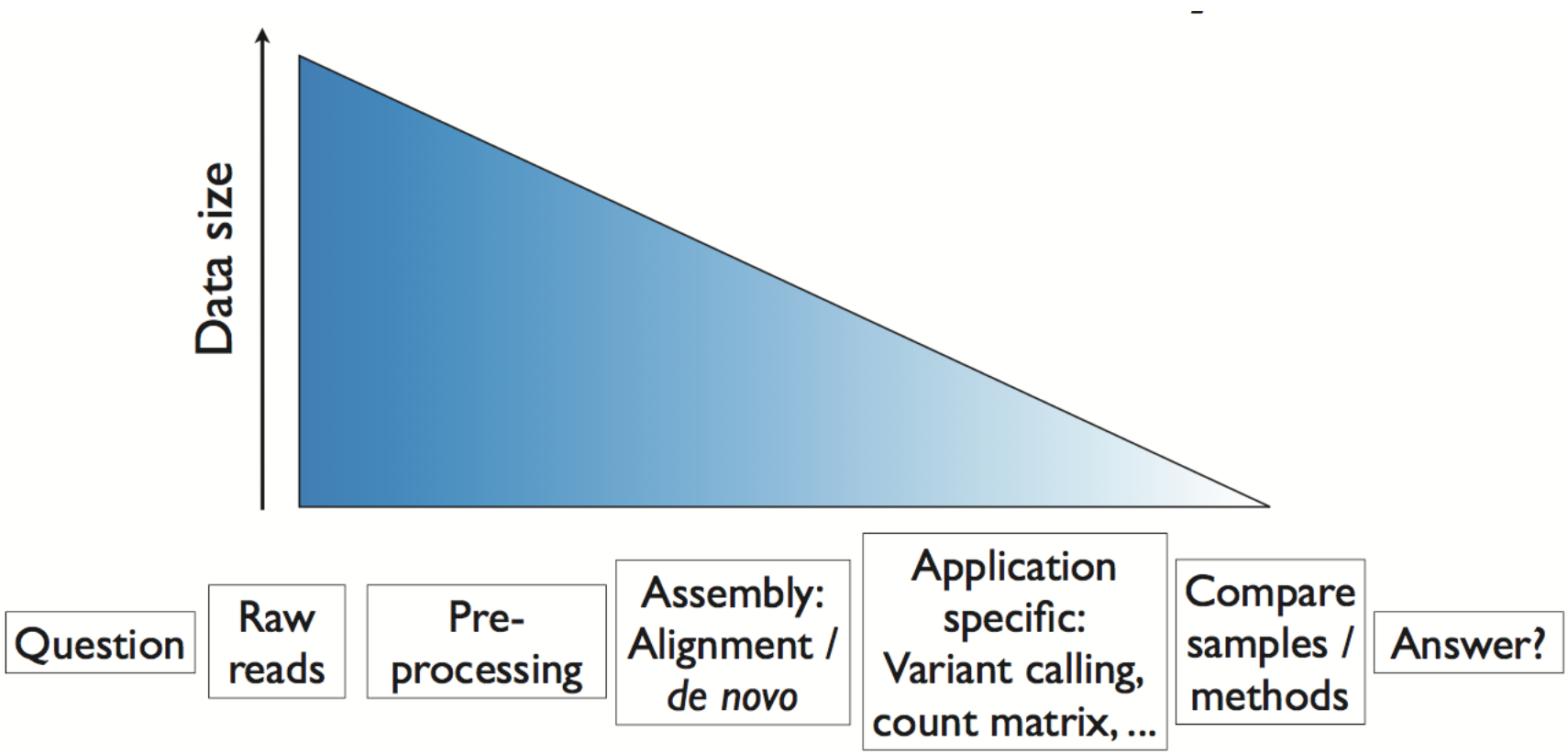
BIO306: Bioinformatics

Lecture 5

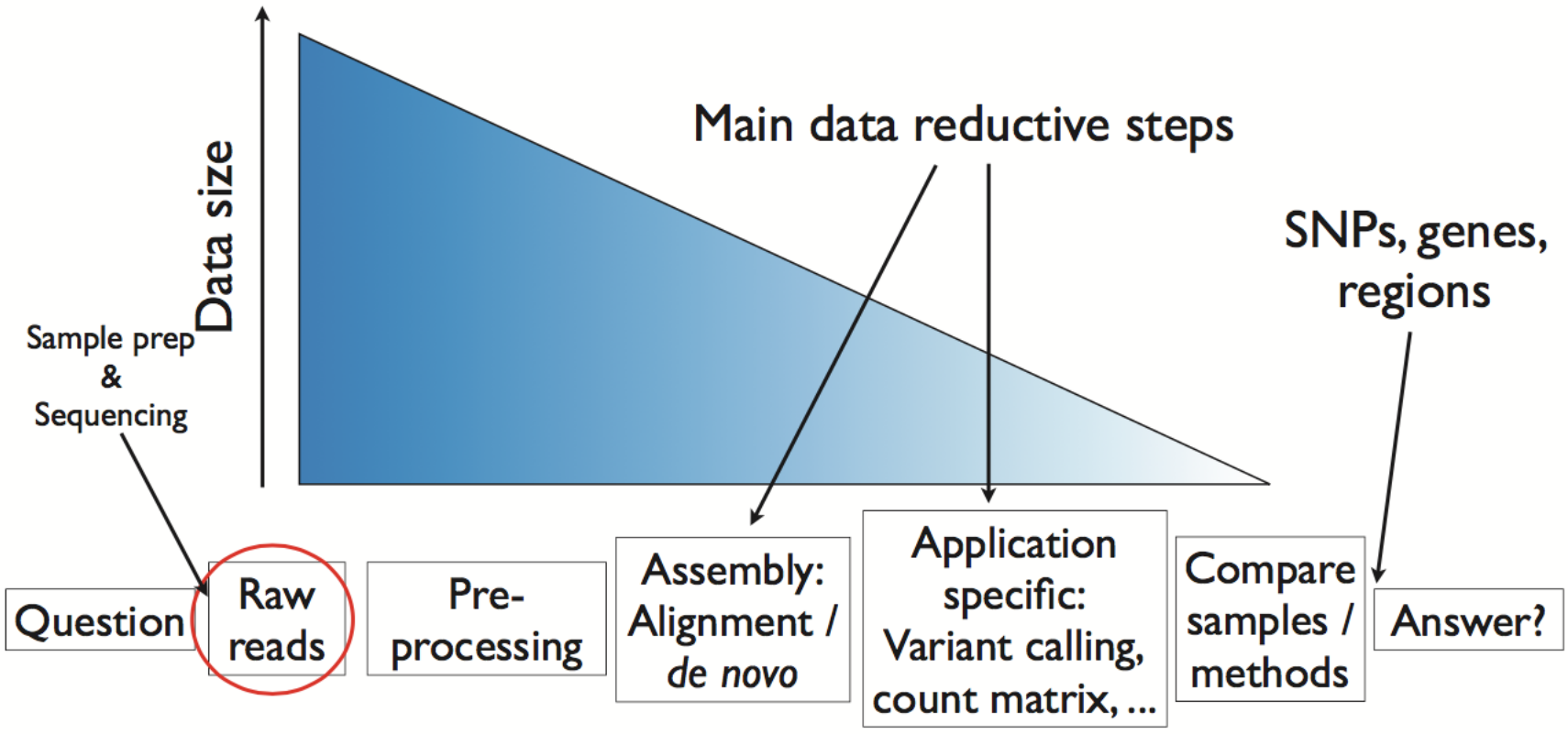
Data Basics and Data Structure

Wenfei JIN PhD
jinwf@sustc.edu.cn
Department of Biology, SUSTech

NGS analysis



NGS analysis



Sequence data

Sequences are stored in fasta-files

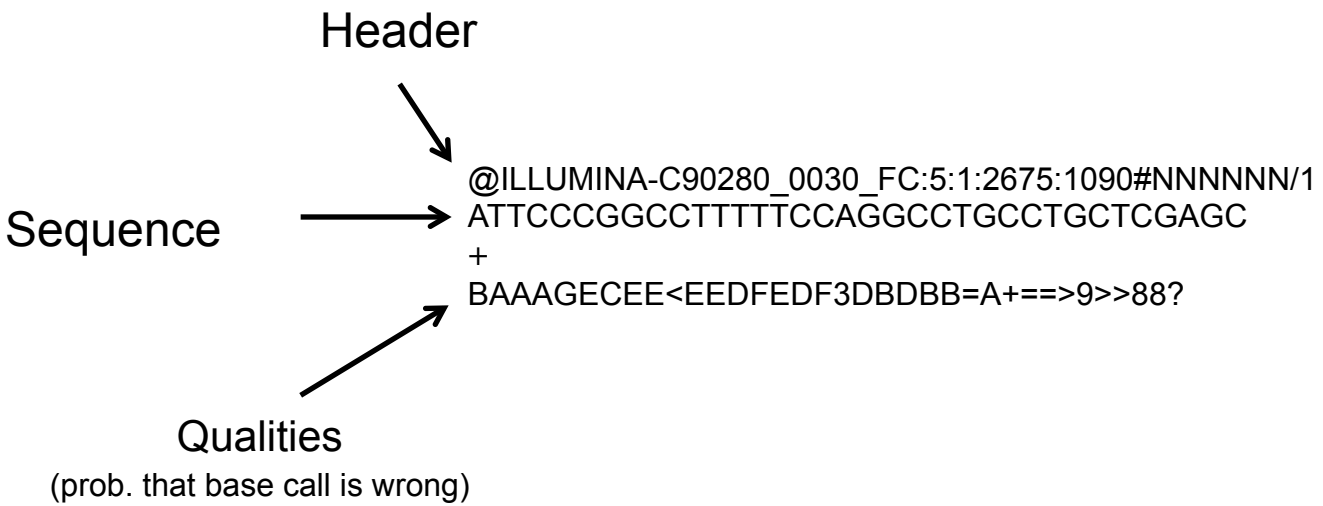
>gi|218693476|ref|NC_011748.1| Escherichia coli 55989 chromosome, complete genome
GTAAGTATTTTTCAGCTTTTCATTCTGACTGCAACGGGCAATATGTCTCTGTGTGGATTAA
AAAAAGAGTGTCTGATAGCAGCTTCTGAACTGGTTACCTGCCGTGAGTAAATTAAAATTTT
ATTGACTTAGGTCACTAAATACTTTAACCAATATAGGCATAGCGCACAGACAGATAAAAATT
ACAGAGTACACAACATCCATGAAACGCATTAGCACCACCATTACCACCACCATCACCATT
ACCACAGGTAACGGTGCGGGCTGACGCGTACAGGAAACACAGAAAAAAGCCCGCACCT
GACAGTGCGGGCTTTTTTTTCGACCAAAGGTAACGAGGTAACAACCATGCGAGTGTTGA
AGTTCGGCGGTACATCAGTGGCAAATGCAGAACGTTTTCTGCGTGTTGCCGATATTCTG
GAAAGCAATGCCAGGCAGGGGCAGGTGGCCACCGTCCTCTCTGCCCCCGCCAAAATCA
CCAACCACCTGGTGGCGATGATTGAAAAAACCATTAGCGGCCAGGATGCTTTACCCAATA
TCAGCGATGCCGAACGTATTTTTGCCGAACTTTTGACGGGACTCGCCGCCGCCCAGCC
GGGGTTCCCGCTGGCGCAATTGAAAACTTTCGTCGATCAGGAATTTGCCCAAATAAAAC
AT

E.coli ~ 4.5 - 6 Mbases

Human ~ 3.2 Gbases

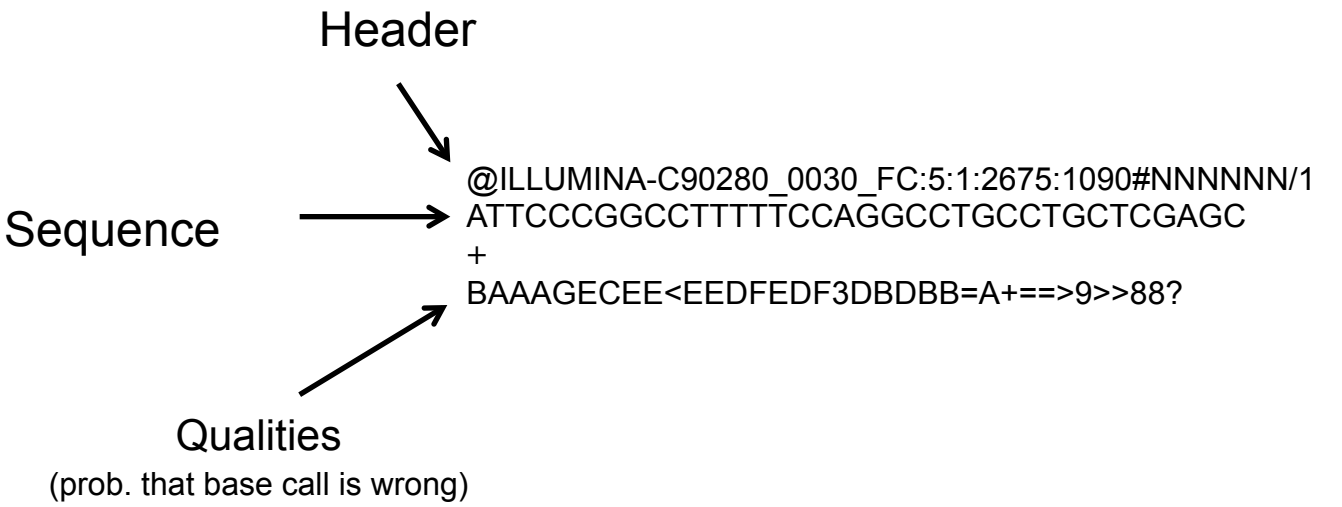
NGS data

fastq



Millions to billions of these

NGS data



One character encodes a number using ascii table (0-255)

This number (*Q*) can be converted to *P*

Phred-scale

$Q = -10 * \log_{10} P$

P=10^(-Q/10)

Phred Quality Score

- base calling program that does detailed statistical analysis on Sanger chromatogram ("trace") files
  - Concept of "PHRED" quality score for base pairs

$$Q = -10 \log_{10}(P_e)$$

e.g. 1 error in 1000 = $-10 \log_{10} (10^{-3}) = 30$

Ewing et al. Genome Research 1998, Vol. 8, Issue 3 for a good overview.

Phred quality scores are logarithmically linked to error probabilities

Phred Quality Score	Probability of incorrect base call	Base call accuracy
10	1 in 10	90%
20	1 in 100	99%
30	1 in 1000	99.9%
40	1 in 10000	99.99%
50	1 in 100000	99.999%
60	1 in 1000000	99.99999%

Phred scale

```
@ILLUMINA-C90280_00
ATTCCCGGCCTTTTTCCAG
+
BAAAGECEE<EEDFEDF3D
↑↑↑

66 65 65
```

Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH	(start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX	(start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX	(end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT	(end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ	(enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK	(acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL	(bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS	(backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB	(horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT	(vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR	(carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO	(shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI	(shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE	(data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1	(device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2	(device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3	(device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4	(device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK	(negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN	(synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB	(end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN	(cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM	(end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB	(substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC	(escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS	(file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS	(group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS	(record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US	(unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Phred scale

```
@ILLUMINA-C90280_00
ATTCCCGGCCTTTTTCCAG
+
BAAAGECEE<EEDFEDF3D
```

66 65 65 ~1e-6

Q ~ Prob

10 ~ 0.1

20 ~ 0.01

30 ~ 0.001

40 ~ 0.0001

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Phred-scaled probabilities

- Base qualities, read mapping qualities, variant qualities, ...
- Straight-forward, except for when they are used in reads!
- Offset: Sanger = 33, Illumina = 64

```
@ILLUMINA-C90280_0030_FC:5:1:2675:1090#NNNNNN/1
ATTCCCGGCCTTTTTCCAGGCCTGCCTGCTCGAGC
+
BAAAGECEE<EEDFEDF3DBDBB=A+==>9>>88?
```

Phred: 66 65 65	~1e-6
Sanger: 33 32 32	~0.001
Illumina: 2 1 1	~1

HUGE difference!

Q ~ Prob
10 ~ 0.1
20 ~ 0.01
30 ~ 0.001
40 ~ 0.0001

Read types

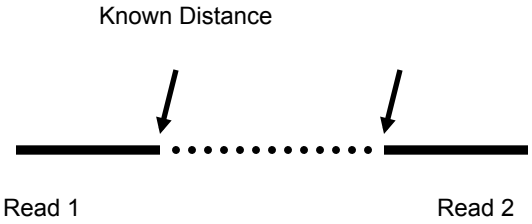
Single end

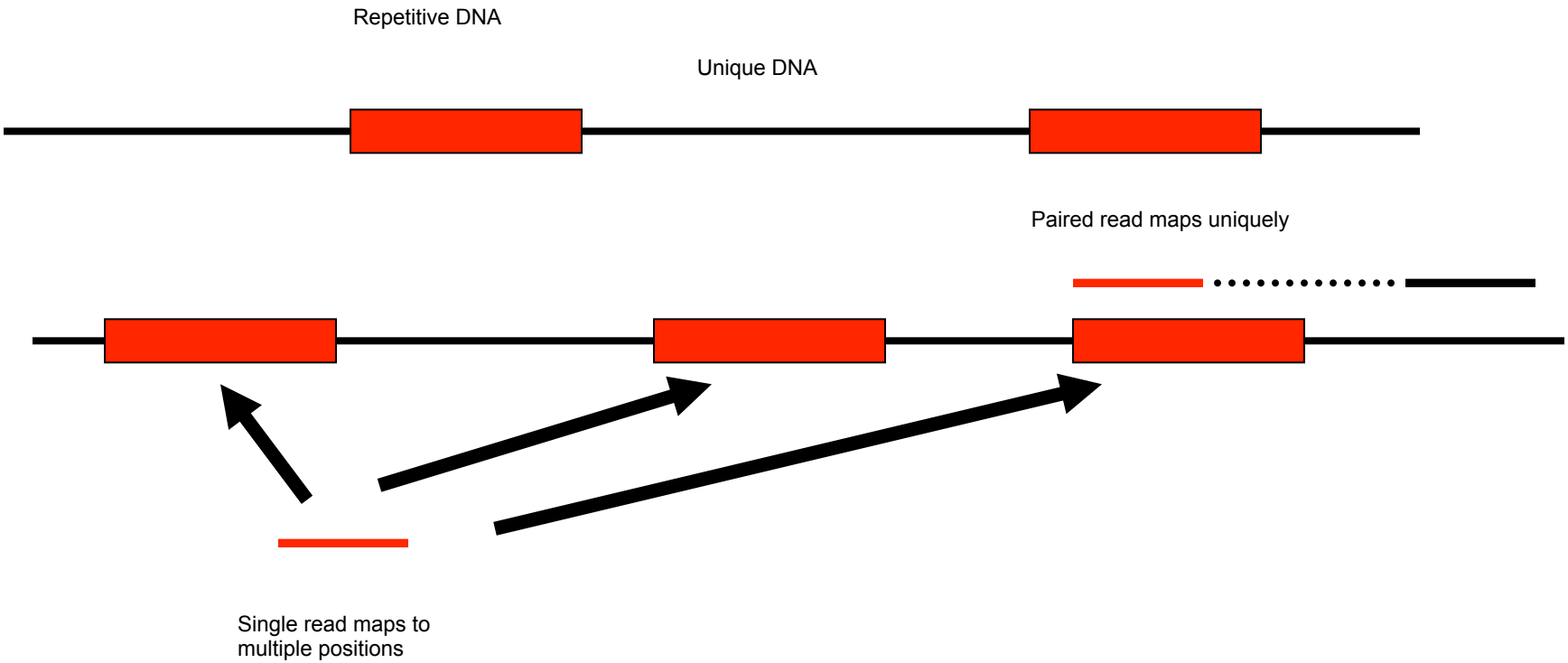
Paired end
Ins 200-800

Mate pair
Ins: 1kb - 40kb (~5kb)

Paired end sequencing

Paired End Reads are Important!



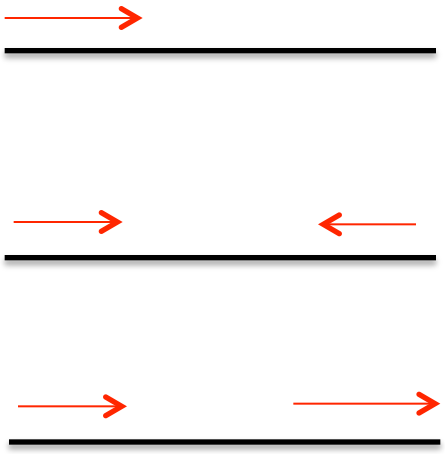


Read orienation

Single end

Paired end

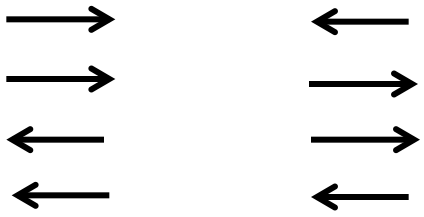
Mate pair



Forward

Reverse

All the possibilioties



Special applications

- Single end reads:
  - Sometimes the only possibility (small DNA fragments / ancient DNA)
- Paired end reads:
  - More precise mapping/alignment/variation calls
  - Medium/Large indels (insertion/deletion)
  - Structural variations
  - Scaffolding in *de novo* assembly
- Mate pairs:
  - Scaffolding in *de novo* assembly
  - Structural variations
  - Chromatin structutre/long distance interaction

Quality control assessment with FASTQC

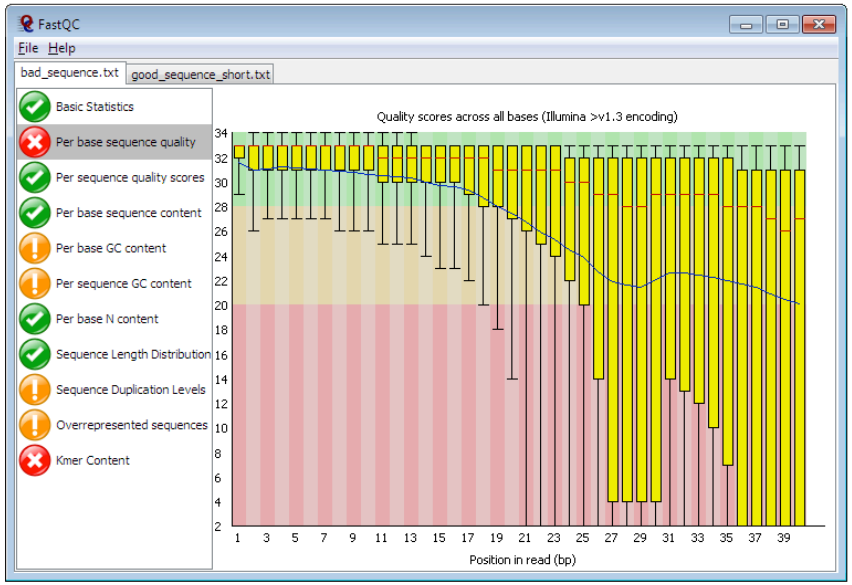
Babraham Bioinformatics

About | People | Services | Projects | Training | Publications

FastQC

Function	A quality control tool for high throughput sequence data.
Language	Java
Requirements	A suitable Java Runtime Environment The Picard BAM/SAM Libraries (included in download)
Code Maturity	Stable. Mature code, but feedback is appreciated.
Code Released	Yes, under GPL v3 or later.
Initial Contact	Simon Andrews

Download Now

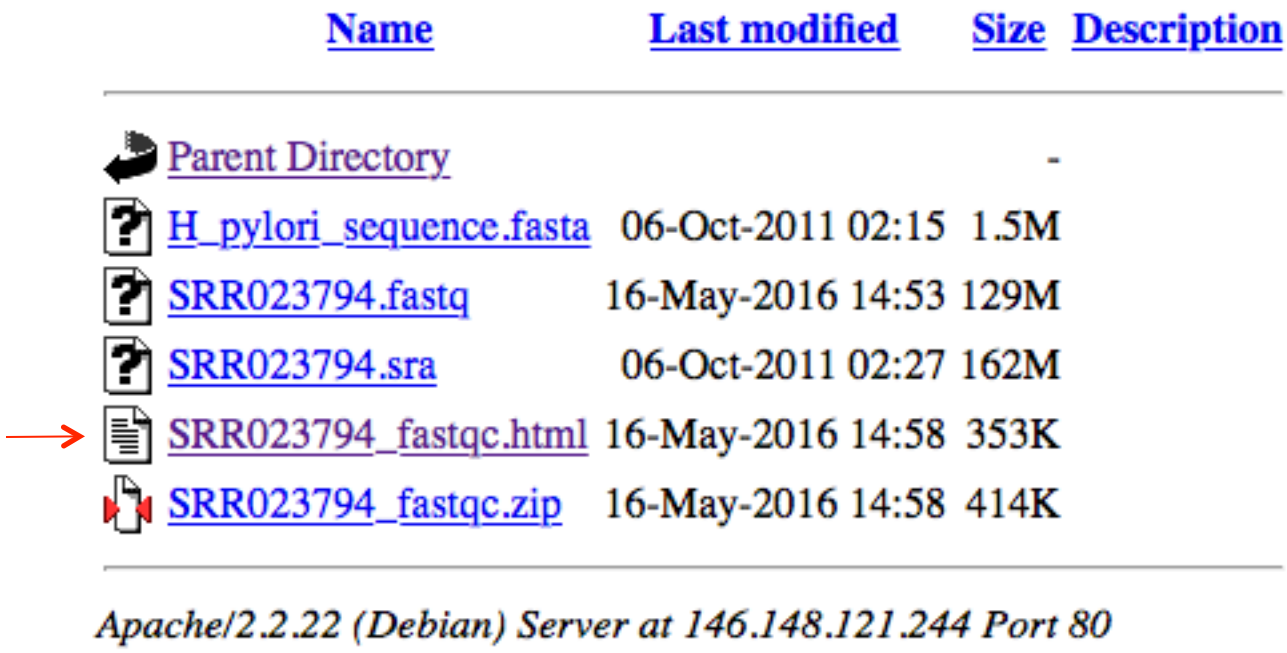


http://www.bioinformatics.bbsrc.ac.uk/projects/fastqc/

Running FASTQC

```
user0@head:~/test$ fastqc SRR023794.fastq
Started analysis of SRR023794.fastq
Approx 5% complete for SRR023794.fastq
Approx 10% complete for SRR023794.fastq
Approx 15% complete for SRR023794.fastq
Approx 20% complete for SRR023794.fastq
Approx 25% complete for SRR023794.fastq
Approx 30% complete for SRR023794.fastq
Approx 35% complete for SRR023794.fastq
Approx 40% complete for SRR023794.fastq
Approx 45% complete for SRR023794.fastq
Approx 50% complete for SRR023794.fastq
Approx 55% complete for SRR023794.fastq
Approx 60% complete for SRR023794.fastq
Approx 65% complete for SRR023794.fastq
Approx 70% complete for SRR023794.fastq
Approx 75% complete for SRR023794.fastq
Approx 80% complete for SRR023794.fastq
Approx 85% complete for SRR023794.fastq
Approx 90% complete for SRR023794.fastq
Approx 95% complete for SRR023794.fastq
Analysis complete for SRR023794.fastq
user0@head :~/test$
```

Running FASTQC



Name	Last modified	Size	Description
Parent Directory		-	
H_pylori_sequence.fasta	06-Oct-2011 02:15	1.5M	
SRR023794.fastq	16-May-2016 14:53	129M	
SRR023794.sra	06-Oct-2011 02:27	162M	
SRR023794_fastqc.html	16-May-2016 14:58	353K	
SRR023794_fastqc.zip	16-May-2016 14:58	414K	

Apache/2.2.22 (Debian) Server at 146.148.121.244 Port 80

Running FASTQC

FastQC Report

Mon 16 May 201
SRR023794.fast

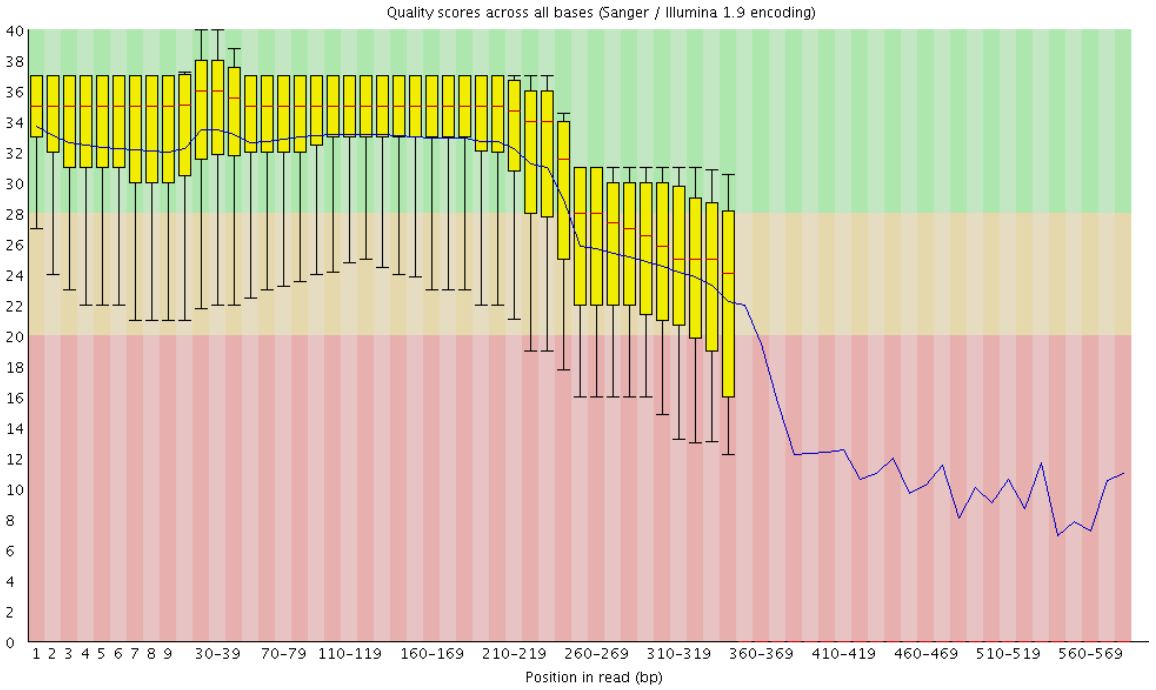
Summary

- Basic Statistics
- Per base sequence quality
- Per sequence quality scores
- Per base sequence content
- Per sequence GC content
- Per base N content
- Sequence Length Distribution
- Sequence Duplication Levels
- Overrepresented sequences
- Adapter Content
- Kmer Content

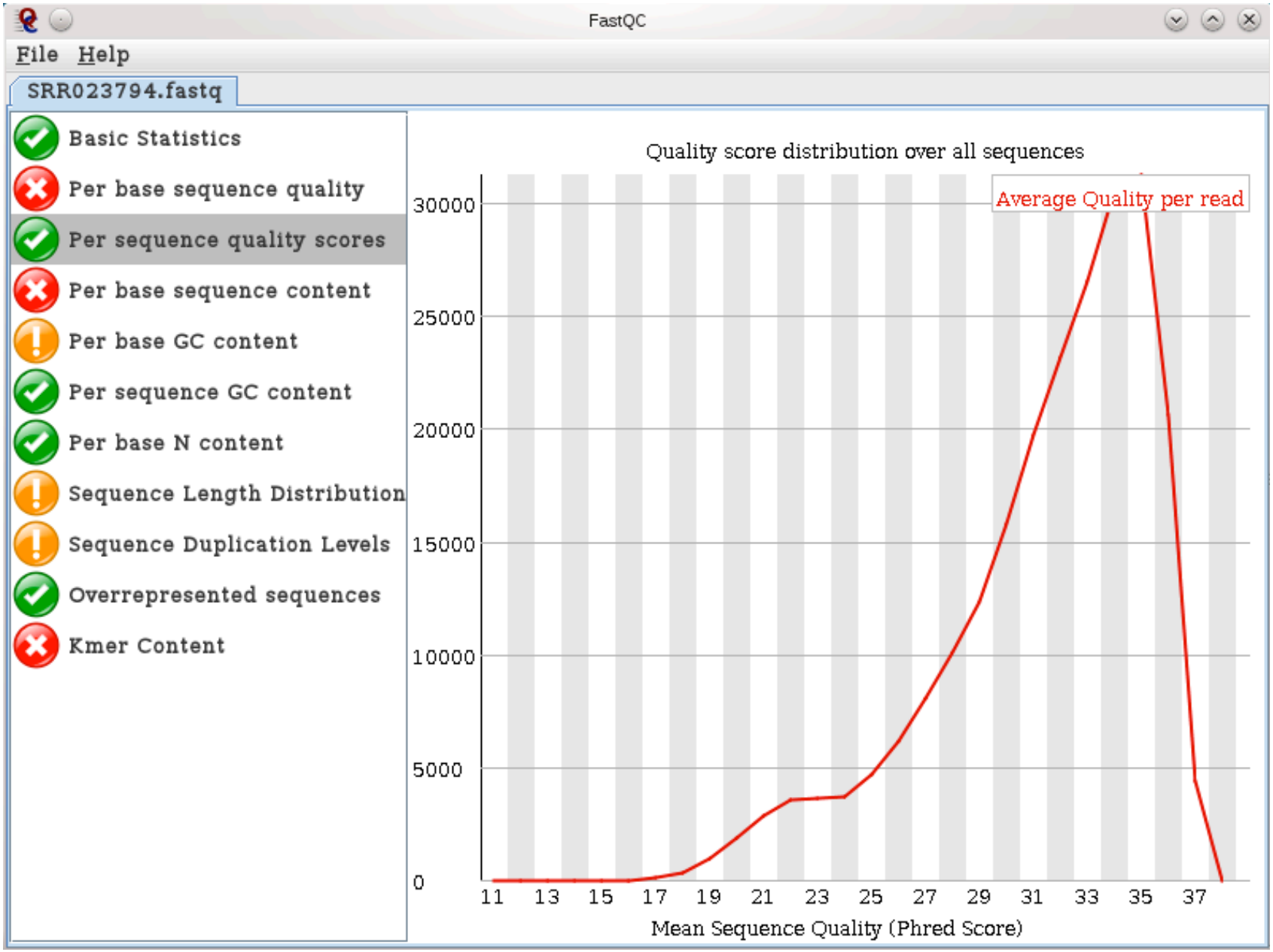
Basic Statistics

Measure	Value
Filename	SRR023794.fastq
File type	Conventional base calls
Encoding	Sanger / Illumina 1.9
Total Sequences	231208
Sequences flagged as poor quality	0
Sequence length	40-580
%GC	38

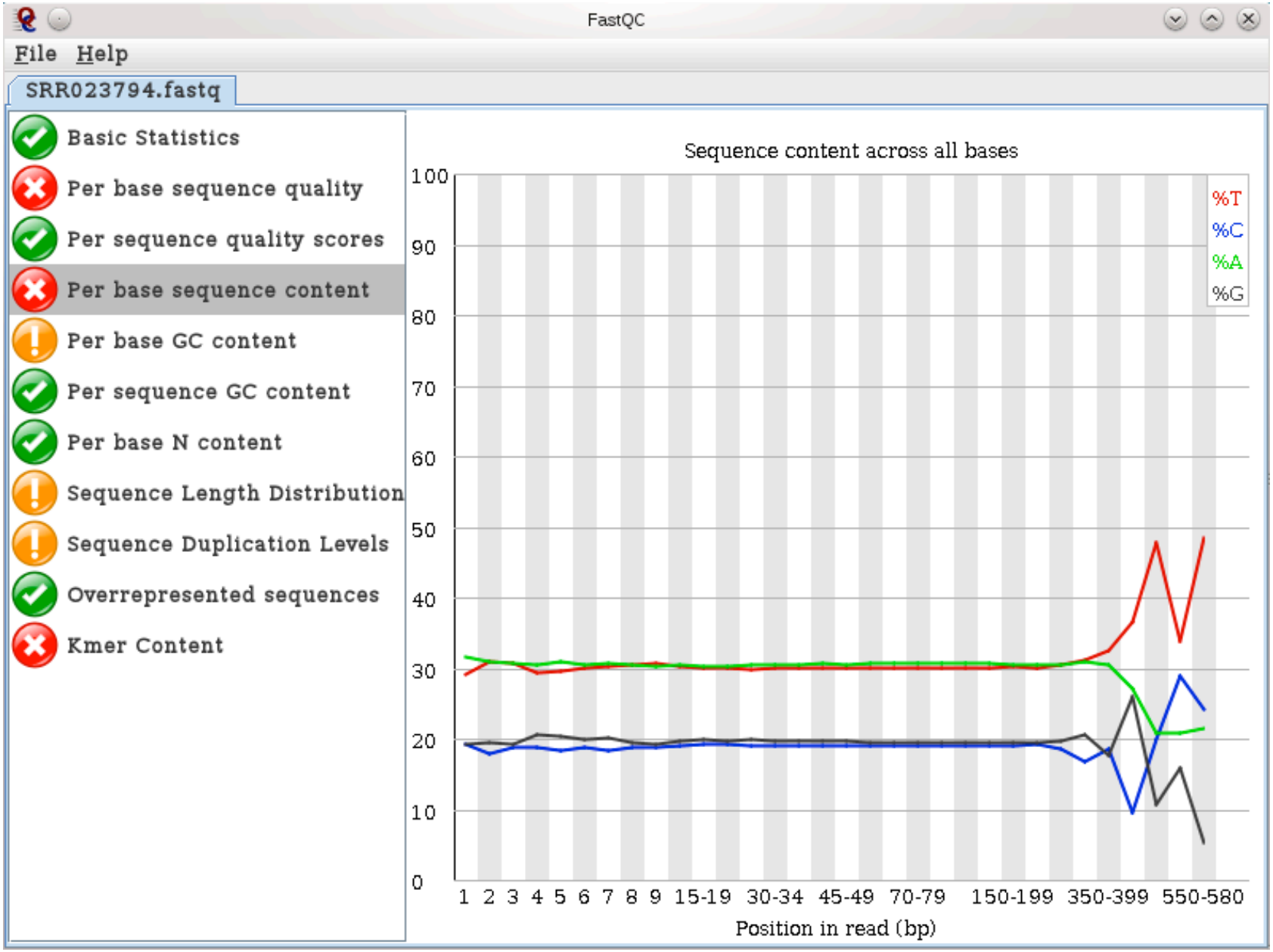
Per base sequence quality



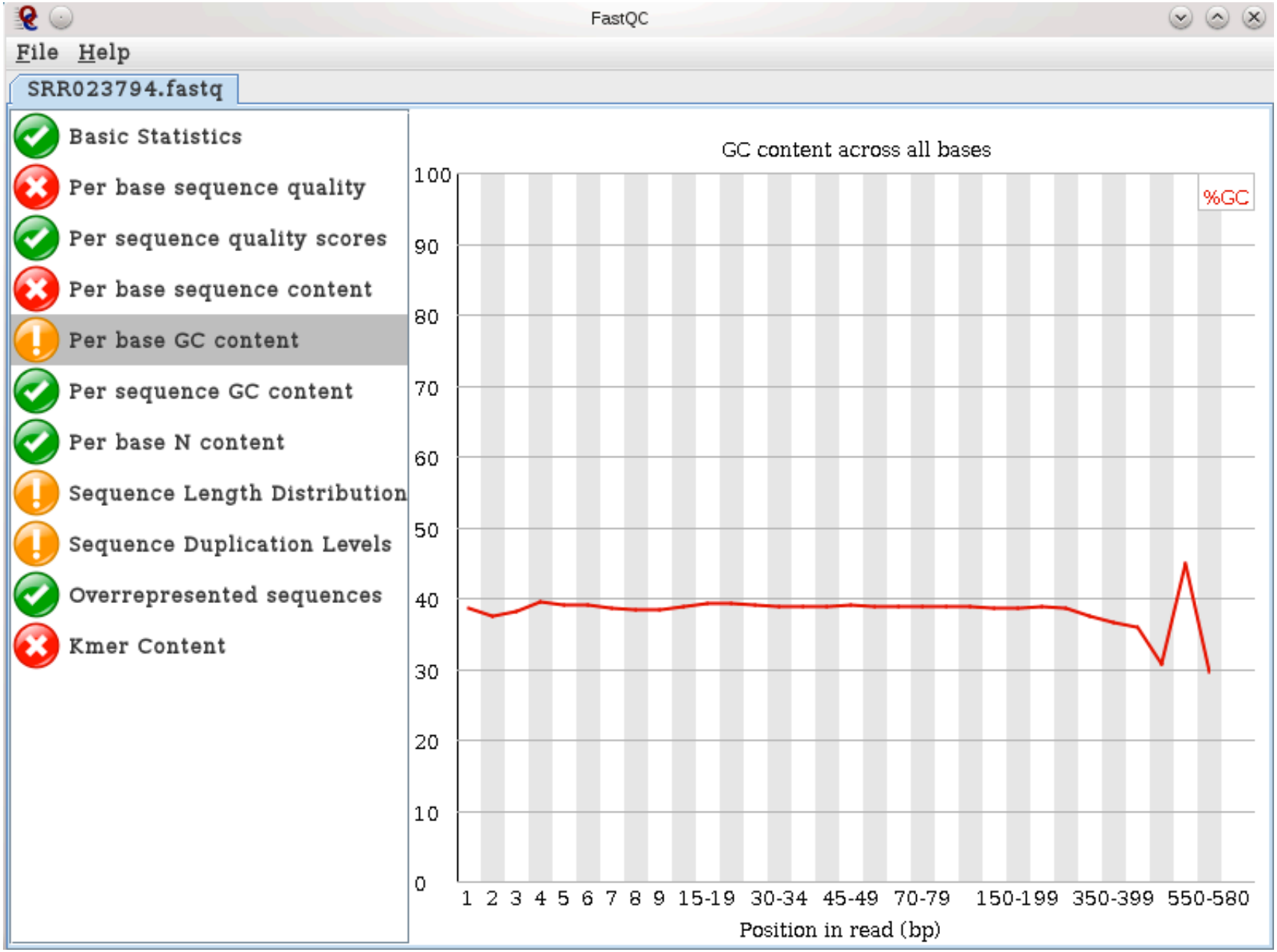
Running FASTQC



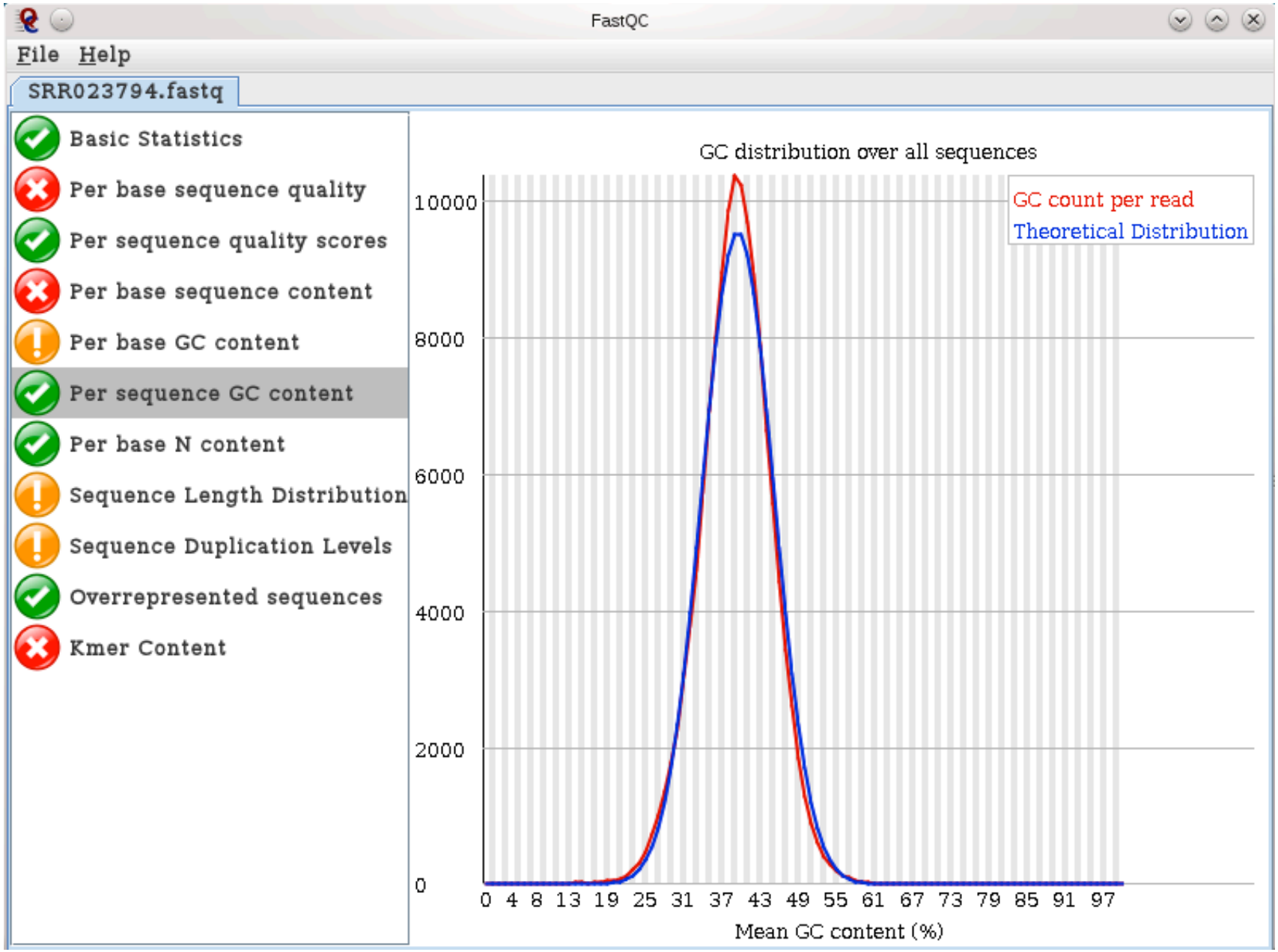
Running FASTQC



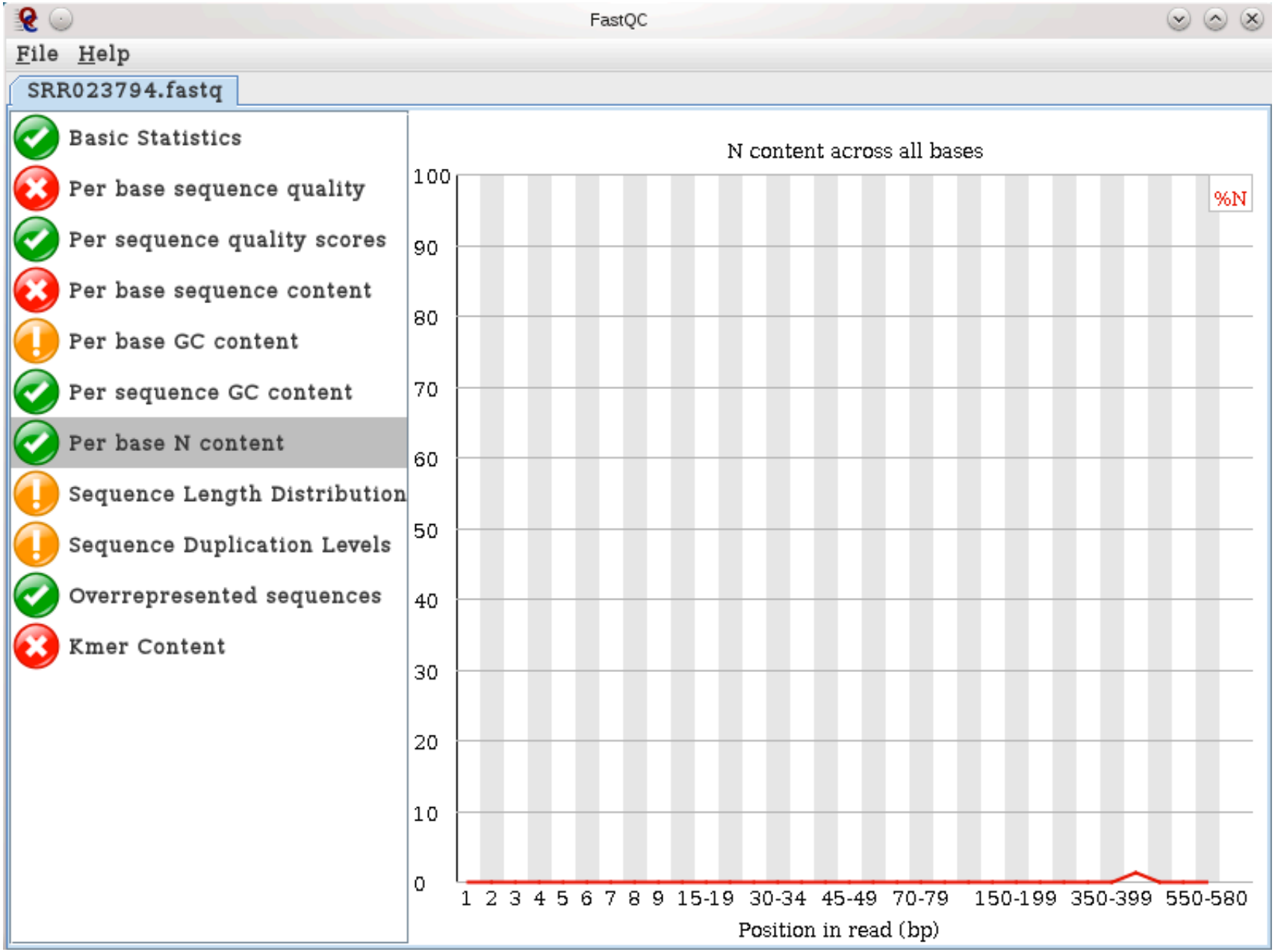
Running FASTQC



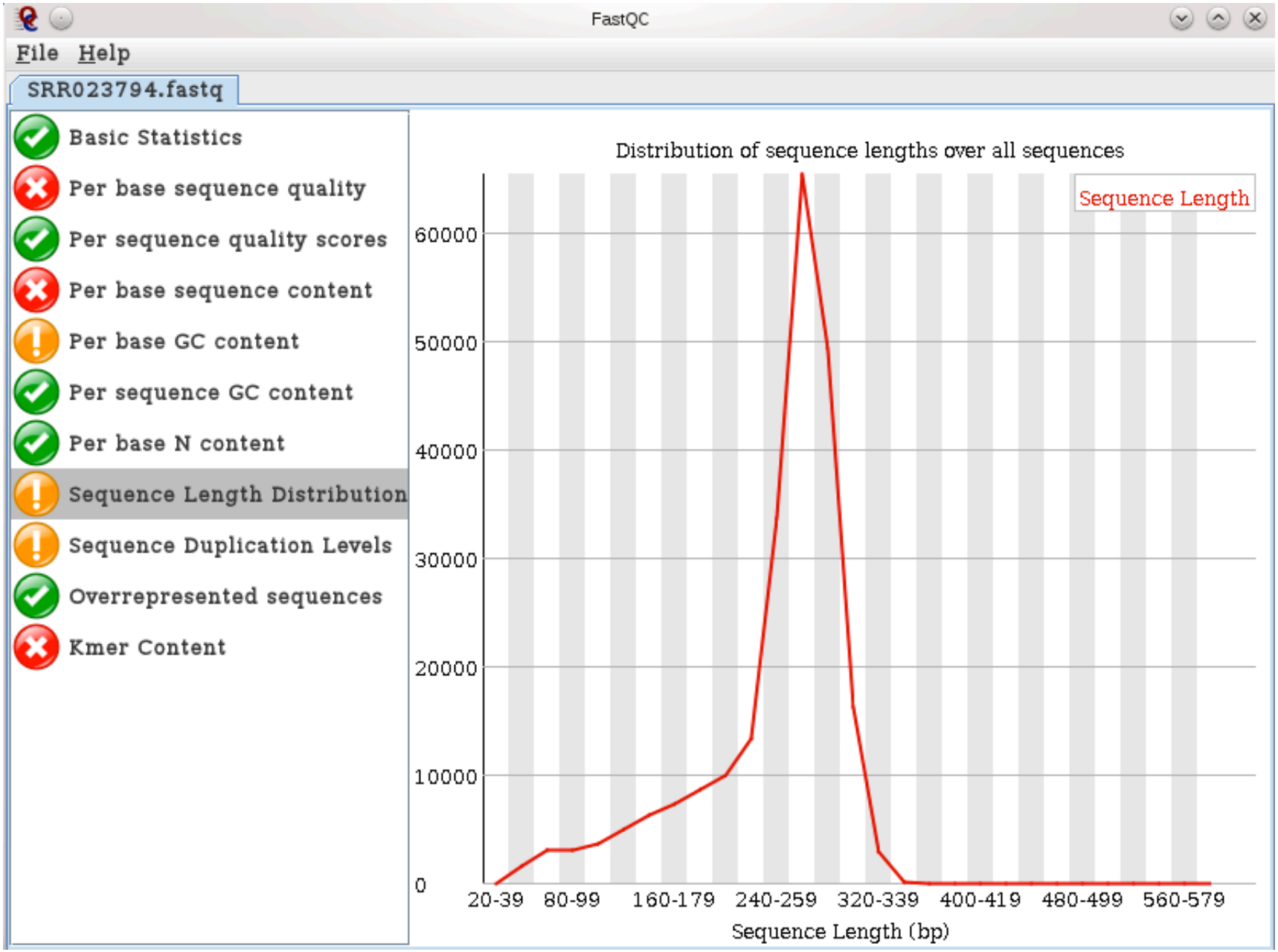
Running FASTQC



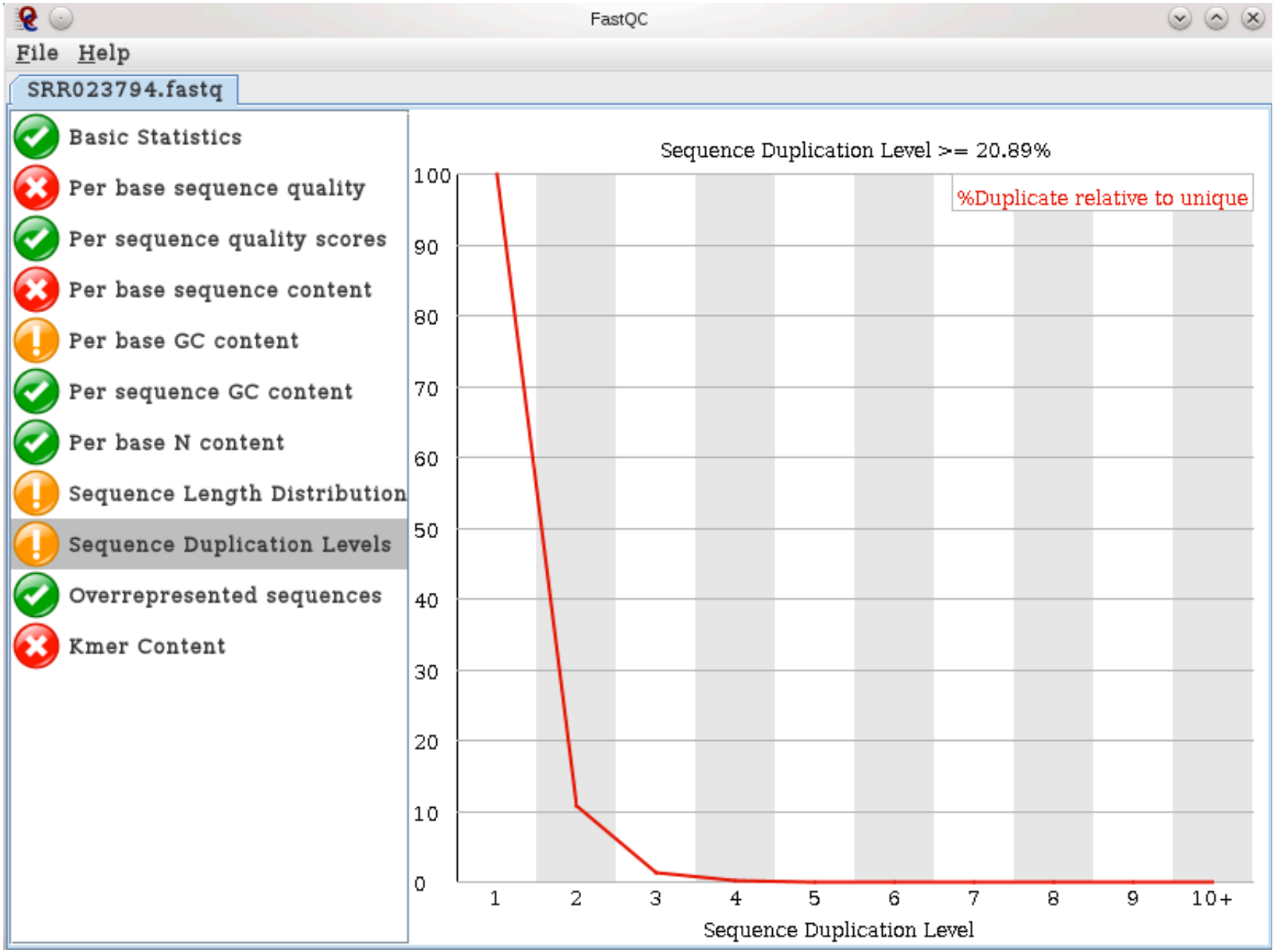
Running FASTQC



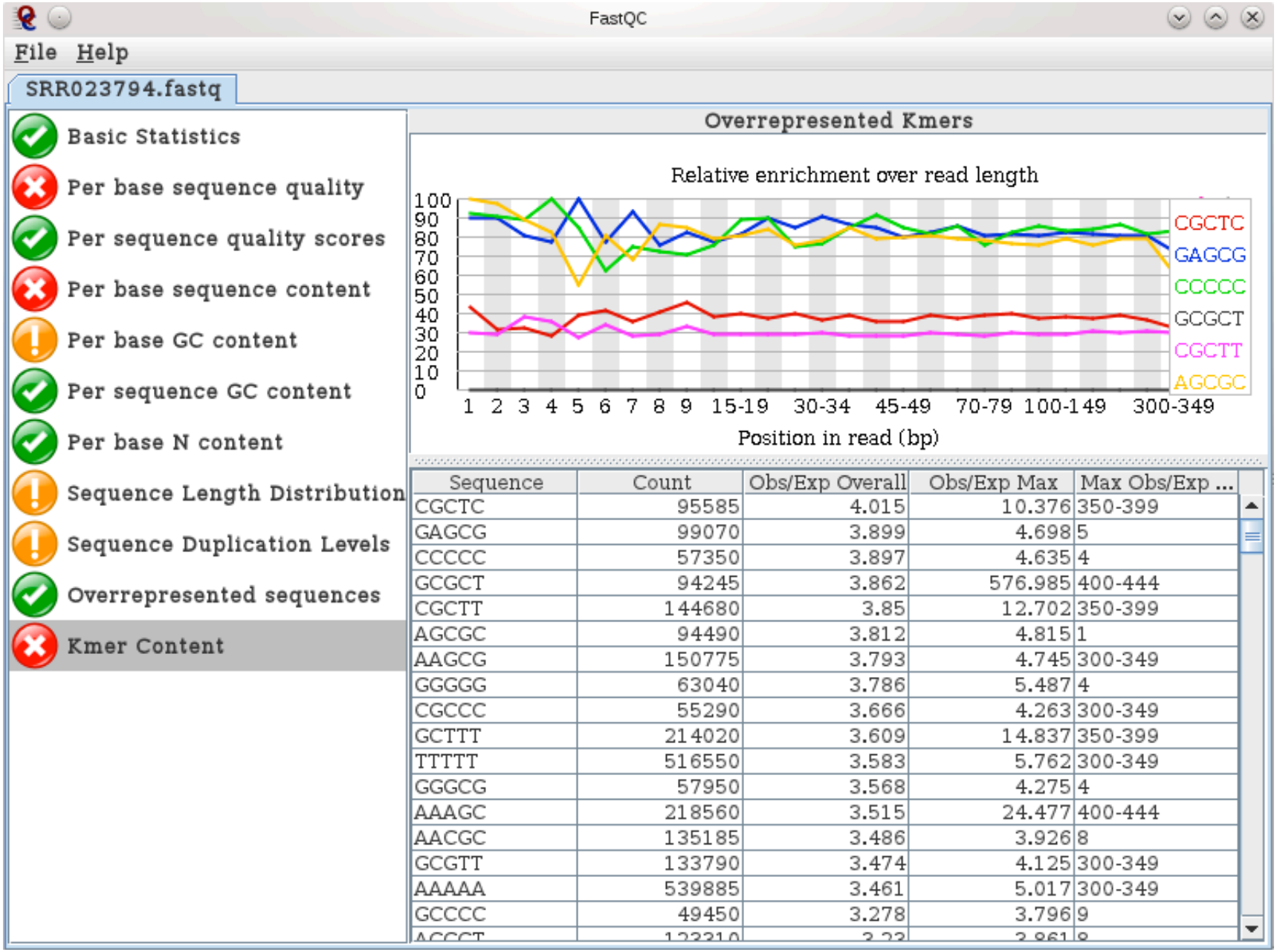
Running FASTQC



Running FASTQC



Running FASTQC



Sequence	Count	Obs/Exp Overall	Obs/Exp Max	Max Obs/Exp ...
CGCTC	95585	4.015	10.376	350-399
GAGCG	99070	3.899	4.698	5
CCCCC	57350	3.897	4.635	4
GCGCT	94245	3.862	576.985	400-444
CGCTT	144680	3.85	12.702	350-399
AGCGC	94490	3.812	4.815	1
AAGCG	150775	3.793	4.745	300-349
GGGGG	63040	3.786	5.487	4
CGCCC	55290	3.666	4.263	300-349
GCTTT	214020	3.609	14.837	350-399
TTTTT	516550	3.583	5.762	300-349
GGGCG	57950	3.568	4.275	4
AAAGC	218560	3.515	24.477	400-444
AACGC	135185	3.486	3.926	8
GCGTT	133790	3.474	4.125	300-349
AAAAA	539885	3.461	5.017	300-349
GCCCC	49450	3.278	3.796	9

Short-Read Mapping

- Initial Problem
  - Text searching: Finding occurrences of a pattern string in a large (static) document
- Solution
  - Text indexing: Trading space for time
- New Problem
  - Succinct Text indexes: Reducing the space cost

Pattern Searching

- Give a text string T of length n and a pattern string P of length m, we look for the occurrences of P in T.
- Three types of Queries
  - Existential queries: Does P occur in T?
  - Cardinality queries: How many times does P occur in T?
  - Listing queries: Where does P occur in T?

Trie structure

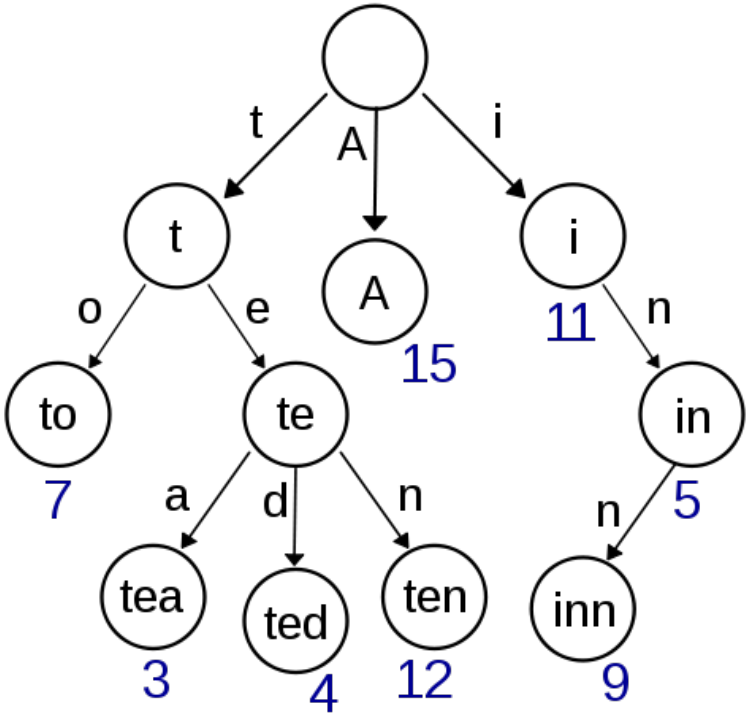
- A family of methods which uses a Trie structure to search a reference sequence
  - Trie (prefix Tree)
  - Suffix Tree
  - Array

Trie

A trie (pronounced “try”) is a search tree—an ordered tree data structure that is used to store a dynamic set or associative array where the keys are usually strings.

Trie

- Smallest tree such that:
  - Each edge is labeled with a character $c \in \Sigma$
  - A node has at most one outgoing edge labeled c, for $c \in \Sigma$
  - Each key is "spelled out" along some path starting at the root
- Natural way to represent either a *set* or a *map* where keys are strings



Advantage of Trie

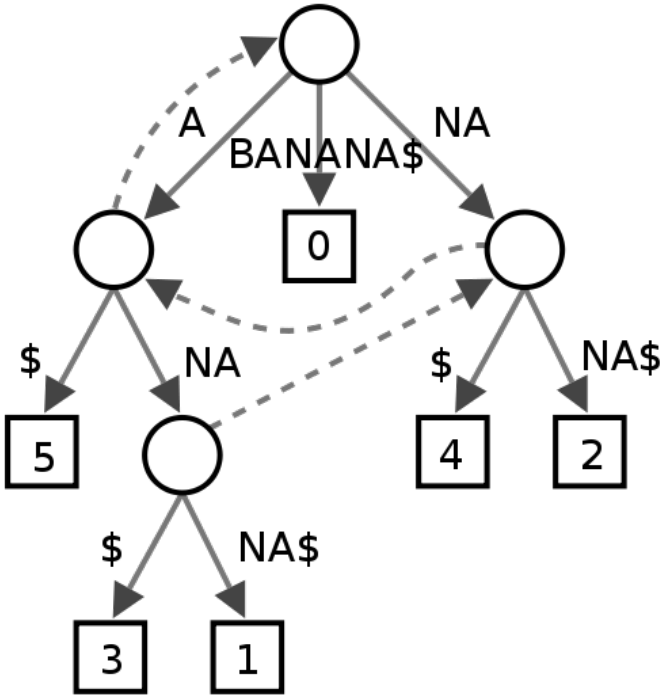
- Looking up data in a trie is faster in the worst case, O(m) time (where m is the length of a search string), compared to an imperfect hash table.
- There are no collisions of different keys in a trie.
- Buckets in a trie, which are analogous to hash table buckets that store key collisions, are necessary only if a single key is associated with more than one value.
- There is no need to provide a hash function or to change hash functions as more keys are added to a trie.
- A trie can provide an alphabetical ordering of the entries by key.

Suffix tree

Compressed trie containing all the suffixes of the given text as their keys and positions in the text as their values. Suffix trees allow particularly fast implementations of many important string operations.

Suffix tree

- The suffix tree for the string S of length n is defined as a tree such that
  - The tree has exactly n leaves numbered from 1 to n.
  - Except for the root, every internal node has at least two children.
  - Each edge is labeled with a non-empty substring of S.
  - No two edges starting out of a node can have string-labels beginning with the same character.
  - The string obtained by concatenating all the string-labels found on the path from the root to leaf i spells out suffix S[i..n], for i from 1 to n.



Features of suffix tree

- The construction of such a tree for the string *S* takes time and space linear in the length of S.
- Once constructed, several operations can be performed quickly, for instance locating a substring in S, locating a substring if a certain number of mistakes are allowed, locating matches for a regular expression pattern etc.
- Suffix trees also provide one of the first linear-time solutions for the longest common substring problem.
- These speedups come at a cost: storing a string's suffix tree typically requires significantly more space than storing the string itself.

Suffix array

- a suffix array is a sorted array of all suffixes of a string.
  - a data structure used, among others, in full text indices, data compression algorithms and within the field of bibliometrics.
  - as a simple, space efficient alternative to suffix trees

Definition of suffix array

- Let S=S[1]S[2]...S[n] be a string and let S[i,j] denote the substring of S ranging from i to j.

- The suffix array A of S is now defined to be an array of integers providing the starting positions of suffixes of S in lexicographical order. This means, an entry A[i] contains the starting position of the i-th smallest suffix in S and thus for all $1<I \leq n$: $S[A[i-1],n]<S[A[i],n]$.

Example of suffix array(1)

Consider the text S= `banana$` to be indexed:

i	1	2	3	4	5	6	7
$S[i]$	b	a	n	a	n	a	$

Suffix	i
banana$	1
anana$	2
nana$	3
ana$	4
na$	5
a$	6
$	7

Suffix	i
$	7
a$	6
ana$	4
anana$	2
banana$	1
na$	5
nana$	3

The text ends with the special sentinel letter $ that is unique and lexicographically smaller than any other character. The text has the following suffixes.

These suffixes can be sorted in ascending order.

Example of suffix array(2)

Suffix	i
$	7
a$	6
ana$	4
anana$	2
banana$	1
na$	5
nana$	3

→

i	1	2	3	4	5	6	7
$A[i]$	7	6	4	2	1	5	3

→

i	1	2	3	4	5	6	7
$A[i]$	7	6	4	2	1	5	3
1	$	a	a	a	b	n	n
2		$	n	n	a	a	a
3			a	a	n	$	n
4			$	n	a		a
5				a	n		$
6				$	a		
7					$		

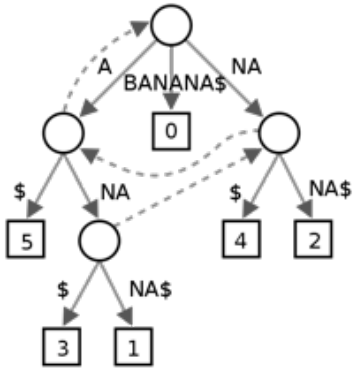
- The suffix array A contains the starting positions of these sorted suffixes:
- The suffix array with the suffixes written out vertically underneath for clarity.
- So for example, A[3] contains the value 4, and therefore refers to the suffix starting at position 4 within S, which is the suffix ana$.

Suffix array: summary

Suffix array gives us index that is:

(a) Just *m* integers, with O(*n* log *m*) worst-case query time, but close to O(*n* + log *m*) in practice

or **(b)** 3*m* integers, with O(*n* + log *m*) worst case



Suffix Tree

6	$
5	A$
3	ANA$
1	ANANA$
0	BANANA$
4	NA$
2	NANA$

Suffix Array

(a) will often be preferable: index for entire human genome fits in ~12 GB instead of > 45 GB

Tally (count sorting)

- Counting sort is a sorting technique based on keys between a specific range.
- It works by counting the number of objects having distinct key values (kind of hashing).
- Then doing some arithmetic to calculate the position of each object in the output sequence.

Implementation of Tally

- 1) Take a count array to store the count of each unique object.
- 2) Modify the count array such that each element at each index stores the sum of previous counts.
-
- 3) Output each object from the input sequence followed by decreasing its count by 1.

Thank you for your attention!