6. Strings

Robert Sedgewick and Kevin Wayne • • http://www.Princeton.EDU/~cos226

Strings

String. Sequence of characters.

Ex. Natural languages, Java programs, genomic sequences, …

> The digital information that underlies biochemistry, cell biology, and development can be represented by a simple string of G's, A's, T's and C's. This string is the root data structure of an organism's biology. -M. V. Olson

Using Strings in Java

String concatenation. Append one string to end of another string.

Substring. Extract a contiguous list of characters from a string.

s	t	r	i	n	g	s
0	1	2	3	4	5	6

```
String s = "strings";          // s = "strings"
char   c = s.charAt(2);        // c = 'r'
String t = s.substring(2, 6);  // t = "ring"
String u = s + t;              // u = "stringsring"
```


Implementing Strings In Java

Memory. 40 + 2N bytes for a virgin string!

↖ could use byte array instead of String to save space

```
public final class String implements Comparable<String> {
   private char[] value;   // characters
   private int offset;     // index of first char into array
   private int count;      // length of string
   private int hash;       // cache of hashCode()

   private String(int offset, int count, char[] value) {
       this.offset = offset;
       this.count  = count;
       this.value  = value;
   }
   public String substring(int from, int to) {
      return new String(offset + from, to - from, value);
   }
   …
}
                              java.lang.String
```

String vs. StringBuilder

`String`. [immutable] Fast substring, slow concatenation.
`StringBuilder`. [mutable] Slow substring, fast (amortized) append.

```
public static String reverse(String s) {
   String rev = "";
   for (int i = s.length() - 1; i >= 0; i--)
         rev += s.charAt(i);
   return rev;
}
```

quadratic time

```
public static String reverse(String s) {
   StringBuilder rev = new StringBuilder();
   for (int i = s.length() - 1; i >= 0; i--)
      rev.append(s.charAt(i));
   return rev.toString();
}
```

linear time

Longest Common Prefix

Longest common prefix. Given two strings, find the common prefix that is as long as possible.

p	r	e	f	i	x		
0	1	2	3	4	5	6	7
p	r	e	f	e	t	c	h

```
public static String lcp(String s, String t) {
   int n = Math.min(s.length(), t.length());
   for (int i = 0; i < n; i++) {
      if (s.charAt(i) != t.charAt(i))
         return s.substring(0, i);
   }
   return s.substring(0, n);
}
```


Radix Sorting

Reference: Chapter 13, Algorithms in Java, 3rd Edition, Robert Sedgewick.

Radix Sorting

Radix sorting.

- Specialized sorting solution for strings.
- Same ideas for bits, digits, etc.

Applications.

- Bioinformatics.
- Sorting strings.
- Full text indexing.
- Plagiarism detection.
- Burrows-Wheeler transform. [see data compression]

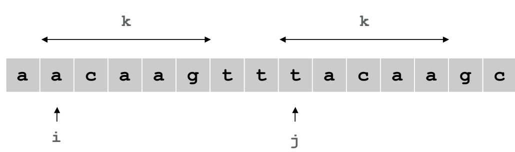
An Application: Redundancy Detector

Longest repeated substring.

- Given a string of N characters, find the longest repeated substring.
- Ex: `a a c a a g t t t a c a a g c`
- Application: bioinformatics.

Dumb brute force.

- Try all indices `i` and `j`, and all match lengths `k`, and check.
- $O(W N^3)$ time, where W is length of longest match.



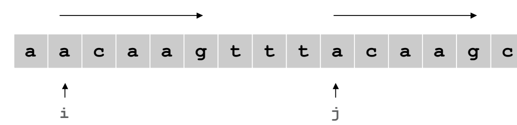
An Application: Redundancy Detector

Longest repeated substring.

- Given a string of N characters, find the longest repeated substring.
- Ex: `a a c a a g t t t a c a a g c`
- Application: bioinformatics.

Brute force.

- Try all indices `i` and `j` for start of possible match, and check.
- $O(W N^2)$ time, where W is length of longest match.



An Application: Redundancy Detector

Longest repeated substring.

- Given a string of N characters, find the longest repeated substring.
- Ex: `a a c a a g t t t a c a a g c`
- Application: bioinformatics.

Suffix sort solution.

- Form N suffixes of original string.
- Sort to bring longest repeated substrings together.
- $O(W N \log N)$ time, where W is length of longest match.

Suffix Sorting

suffixes

0	a a c a a g t t t a c a a g c
1	a c a a g t t t a c a a g c
2	c a a g t t t a c a a g c
3	a a g t t t a c a a g c
4	a g t t t a c a a g c
5	g t t t a c a a g c
6	t t t a c a a g c
7	t t a c a a g c
8	t a c a a g c
9	a c a a g c
10	c a a g c
11	a a g c
12	a g c
13	g c
14	c

sorted suffixes

0	a a c a a g t t t a c a a g c
11	a a g c
3	a a g t t t a c a a g c
▶ 9	a c a a g c
▶ 1	a c a a g t t t a c a a g c
12	a g c
4	a g t t t a c a a g c
14	c
10	c a a g c
2	c a a g t t t a c a a g c
13	g c
5	g t t t a c a a g c
8	t a c a a g c
7	t t a c a a g c
6	t t t a c a a g c

Suffix Sorting: Java Implementation

```
public class LRS {
   public static void main(String[] args) {
      String s = StdIn.readAll();                       read input
      int N = s.length();

      String[] suffixes = new String[N];                create suffixes
      for (int i = 0; i < N; i++)                       (linear time)
         suffixes[i] = s.substring(i, N);

      Arrays.sort(suffixes);                            sort suffixes

      String lrs = "";
      for (int i = 0; i < N - 1; i++) {
         String x = lcp(suffixes[i], suffixes[i+1]);
         if (x.length() > lrs.length()) lrs = x;
      }
      System.out.println(lrs);                          find longest match
   }
}
```

```
% java LRS < mobydick.txt
,- Such a funny, sporty, gamy, jesty, joky, hoky-poky lad, is the Ocean, oh! Th
```

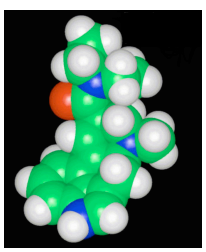

String Sorting Performance

	String Sort	Suffix (sec)
	Worst Case	Moby Dick
Brute	W N^2	36,000 §
Quicksort	W N log N †	9.5

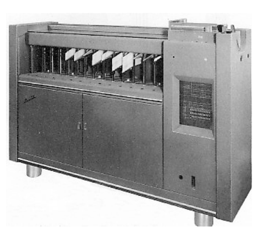
W = max length of string
N = number of strings ← 1.2 million for Moby Dick

§ estimate
† probabilistic guarantee

LSD Radix Sort



Lysergic Acid Diethylamide, Circa 1960



Card Sorter, Circa 1960

String Sorting Notation

Notation.

- String = sequence of characters.
- W = max # characters per string.
- N = # input strings.
- R = radix.

  ← 256 for extended ASCII, 65,536 for original Unicode

Java syntax.

- Array of strings: `String[] a`
- Number of strings: `N = a.length`
- The i^{th} string: `a[i]`
- The d^{th} character of the i^{th} string: `a[i].charAt(d)`
- Strings to be sorted: `a[0], …, a[N-1]`

Key Indexed Counting

Key indexed counting.
- Count frequencies of each letter. [0^{th} character]

```
int N = a.length;
int[] count = new int[256+1];
for (int i = 0; i < N; i++) {
   char c = a[i].charAt(d);
   count[c+1]++;
}
```

d = 0

frequencies

a

0	d	a	b
1	a	d	d
2	c	a	b
3	f	a	d
4	f	e	e
5	b	a	d
6	d	a	d
7	b	e	e
8	f	e	d
9	b	e	d
10	e	b	b
11	a	c	e

count

a	0
b	2
c	3
d	1
e	2
f	1
g	3

Key Indexed Counting

Key indexed counting.
- Count frequencies of each letter. [0^{th} character]
- Compute cumulative frequencies.

```
for (int i = 1; i < 256; i++)
   count[i] += count[i-1];
```

cumulative counts

a

0	d	a	b
1	a	d	d
2	c	a	b
3	f	a	d
4	f	e	e
5	b	a	d
6	d	a	d
7	b	e	e
8	f	e	d
9	b	e	d
10	e	b	b
11	a	c	e

a	0
b	2
c	3
d	1
e	2
f	1
g	3

count

a	0
b	2
c	5
d	6
e	8
f	9
g	12

Key Indexed Counting

Key indexed counting.
- Count frequencies of each letter. [0^{th} character]
- Compute cumulative frequencies.
- Use cumulative frequencies to rearrange strings.

```
for (int i = 0; i < N; i++) {
   char c = a[i].charAt(d);
   temp[count[c]++] = a[i];
}
```

rearrange

a

0	d	a	b
1	a	d	d
2	c	a	b
3	f	a	d
4	f	e	e
5	b	a	d
6	d	a	d
7	b	e	e
8	f	e	d
9	b	e	d
10	e	b	b
11	a	c	e

count

a	2
b	5
c	6
d	8
e	9
f	12
g	12

temp

0	a	d	d
1	a	c	e
2	b	a	d
3	b	e	e
4	b	e	d
5	c	a	b
6	d	a	b
7	d	a	d
8	e	b	b
9	f	a	d
10	f	e	e
11	f	e	d

Key Indexed Counting

Key indexed counting.
- Count frequencies of each letter. [0^{th} character]
- Compute cumulative frequencies.
- Use cumulative frequencies to rearrange strings.

```
for (int i = 0; i < N; i++)
   a[i] = temp[i];
```

copy back

a

0	a	d	d
1	a	c	e
2	b	a	d
3	b	e	e
4	b	e	d
5	c	a	b
6	d	a	b
7	d	a	d
8	e	b	b
9	f	a	d
10	f	e	e
11	f	e	d

count

a	2
b	5
c	6
d	8
e	9
f	12
g	12

temp

0	a	d	d
1	a	c	e
2	b	a	d
3	b	e	e
4	b	e	d
5	c	a	b
6	d	a	b
7	d	a	d
8	e	b	b
9	f	a	d
10	f	e	e
11	f	e	d

Least Significant Digit Radix Sort

LSD. Consider digits d from right to left: stably sort using dth character as the key via key-indexed counting.

		sort key (3rd char)	sort key (2nd char)	sort key (1st char)
0	d a b	d a b	d a b	a c e
1	a d d	c a b	c a b	a d d
2	c a b	e b b	f a d	b a d
3	f a d	a d d	b a d	b e d
4	f e e	f a d	d a d	b e e
5	b a d	b a d	e b b	c a b
6	d a d	d a d	a c e	d a b
7	b e e	f e d	a d d	d a d
8	f e d	b e d	f e d	e b b
9	b e d	f e e	b e d	f a d
10	e b b	b e e	f e e	f e d
11	a c e	a c e	b e e	f e e

Least Significant Digit Radix Sort

LSD. Consider digits d from right to left: stably sort using dth character as the key via key-indexed counting.

```
public static void lsd(String[] a) {
   int W = a[0].length();
   for (int d = W-1; d >= 0; d--) {
      // do key-indexed counting sort on digit d
      ...
   }
}
```

Assumes fixed length strings (length = W)

LSD Radix Sort: Correctness

Pf 1. [left-to-right]

- If two strings differ on first character, key-indexed sort puts them in proper relative order.
- If two strings agree on first character, stability keeps them in proper relative order.

Pf 2. [right-to-left]

- If the characters not yet examined differ, it doesn't matter what we do now.
- If the characters not yet examined agree, stability ensures later pass won't affect order.

now	sob	cab	ace
for	nob	wad	ago
tip	cab	tag	and
ilk	wad	jam	bet
dim	and	rap	cab
tag	ace	tap	caw
jot	wee	tar	cue
sob	cue	was	dim
nob	fee	caw	dug
sky	tag	raw	egg
hut	egg	jay	fee
ace	gig	ace	few
bet	dug	wee	for
men	ilk	fee	gig
egg	owl	men	hut
few	dim	bet	ilk
jay	jam	few	jam
owl	men	egg	jay
joy	ago	ago	jot
rap	tip	gig	joy
gig	rap	dim	men
wee	tap	tip	nob
was	for	sky	now
cab	tar	ilk	owl
wad	was	and	rap
tap	jot	sob	raw
caw	hut	nob	sky
cue	bet	for	sob
fee	you	jot	tag
raw	now	you	tap
ago	few	now	tar
tar	caw	joy	tip
jam	raw	cue	wad
dug	sky	dug	was
you	jay	hut	wee
and	joy	owl	you

LSD Radix Sort Correctness

Running time. $\Theta(W(N + R))$.

↖ why doesn't it violate N log N lower bound?

Advantage. Fastest sorting method for random fixed length strings.

Disadvantages.

- Accesses memory "randomly."
- Inner loop has a lot of instructions.
- Wastes time on low-order characters.
- Doesn't work for variable-length strings.
- Not much semblance of order until very last pass.

Goal. Find fast algorithm for variable length strings.

MSD Radix Sort

MSD Radix Sort

Most significant digit radix sort.

- Partition file into 256 pieces according to first character.
- Recursively sort all strings that start with the same character, etc.

Q. How to sort on dth character?
A. Key-indexed counting.

now	a	ce	ac	e	ace
for	a	go	ag	o	ago
tip	a	nd	an	d	and
ilk	b	et	be	t	bet
dim	c	ab	ca	b	cab
tag	c	aw	ca	w	caw
jot	c	ue	cu	e	cue
sob	d	im	di	m	dim
nob	d	ug	du	g	dug
sky	e	gg	eg	g	egg
hut	f	or	fe	w	fee
ace	f	ee	fe	e	few
bet	f	ew	fo	r	for
men	g	ig	gi	g	gig
egg	h	ut	hu	t	hut
few	i	lk	il	k	ilk
jay	j	am	ja	y	jam
owl	j	ay	ja	m	jay
joy	j	ot	jo	t	jot
rap	j	oy	jo	y	joy
gig	m	en	me	n	men
wee	n	ow	no	w	nob
was	n	ob	no	b	now
cab	o	wl	ow	l	owl
wad	r	ap	ra	p	rap
caw	s	ob	sk	y	sky
cue	s	ky	so	b	sob
fee	t	ip	ta	g	tag
tap	t	ag	ta	p	tap
ago	t	ap	ta	r	tar
tar	t	ar	ti	p	tip
jam	w	ee	wa	d	wad
dug	w	as	wa	s	was
and	w	ad	we	e	wee

MSD Radix Sort Implementation

```
public static void msd(String[] a) {
   msd(a, 0, a.length, 0);
}
                                     inclusive    exclusive

private static void msd(String[] a, int l, int r, int d) {
   if (r <= l + 1) return;

   // key-indexed counting sort on digit d of a[l] to a[r]
   int[] count = new int[256+1];
   ...

   // recursively sort 255 subfiles - assumes '\0' terminated
   for (int i = 0; i < 255; i++)
      msd(a, l + count[i], l + count[i+1], d+1);
}
```


String Sorting Performance

	String Sort	Suffix (sec)
	Worst Case	Moby Dick
Brute	W N²	36,000 §
Quicksort	W N log N †	9.5
LSD *	W(N + R)	-
MSD	W(N + R)	395

R = radix
W = max length of string
N = number of strings (1.2 million for Moby Dick)

§ estimate
* fixed length strings only
† probabilistic guarantee

MSD Radix Sort: Small Files

Disadvantages.
- Too slow for small files.
  - ASCII: 100x slower than insertion sort for N = 2
  - Unicode: 30,000x slower for N = 2
- Huge number of recursive calls on small files.

Solution. Cutoff to insertion sort for small N.
Consequence. Competitive with quicksort for string keys.

String Sorting Performance

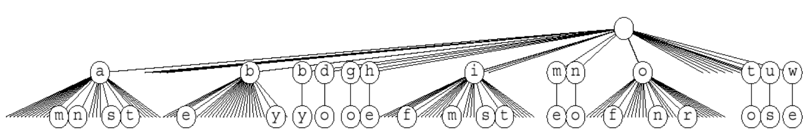
	String Sort	Suffix (sec)
	Worst Case	Moby Dick
Brute	$W N^2$	36,000 §
Quicksort	W N log N †	9.5
LSD *	W(N + R)	-
MSD	W(N + R)	395
MSD with cutoff	W(N + R)	6.8

R = radix
W = max length of string
N = number of strings (1.2 million for Moby Dick)

§ estimate
* fixed length strings only
† probabilistic guarantee

Recursive Structure: MSD

Recursive structure. R-way branching leads to lots of empty calls.



3-Way Radix Quicksort

3-Way Radix Quicksort

Idea 1. Use d^{th} character to "sort" into 3 pieces instead of 256, and sort each piece recursively.
Idea 2. Keep all duplicates together in partitioning step.

actinian	coenobite	actinian
jeffrey	conelrad	bracteal
coenobite	actinian	coenobite
conelrad	bracteal	conelrad
secureness	secureness	cumin
cumin	dilatedly	chariness
chariness	inkblot	centesimal
bracteal	jeffrey	cankerous
displease	displease	circumflex
millwright	millwright	millwright
repertoire	repertoire	repertoire
dourness	dourness	dourness
centesimal	southeast	southeast
fondler	fondler	fondler
interval	interval	interval
reversionary	reversionary	reversionary
dilatedly	cumin	secureness
inkblot	chariness	dilatedly
southeast	centesimal	inkblot
cankerous	cankerous	jeffrey
circumflex	circumflex	displease

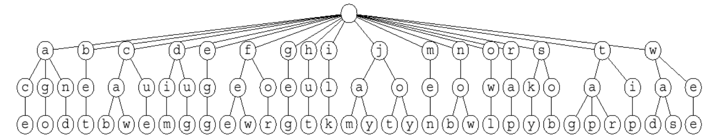
3-way partition

now	gig	ace	ago	a go	
for	for	bet	bet	a ce	
tip	dug	dug	and	a nd	
ilk	ilk	cab	ace	b et	
dim	dim	dim	c ab		
tag	ago	ago	c aw		
jot	and	and	c ue		
sob	fee	egg	egg		
nob	cue	cue	dug		
sky	caw	caw	dim		
hut	hut	f ee			
ace	ace	f or			
bet	bet	f ew			
men	cab	ilk			
egg	egg	gig			
few	few	hut			
jay	j ay	ja m			
owl	j ot	ja y			
joy	j oy	jo y			
rap	j am	jo t			
gig	owl	owl	m en		
wee	wee	now	owl		
was	was	nob	nob		
cab	men	men	now		
wad	wad	r ap			
caw	sky	sky	sky	sky	
cue	nob	was	tip	sob	
fee	sob	sob	sob	t ip	ta r
tap	tap	tap	tap	t ap	ta p
ago	tag	tag	tag	t ag	ta g
tar	tar	tar	tar	t ar	ti p
dug	tip	tip	w as		
and	now	wee	w ee		
jam	rap	wad	w ad		

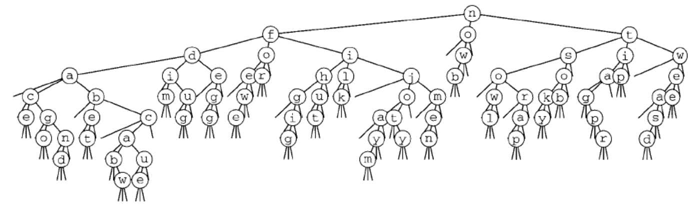
3-way radix quicksort

Recursive Structure: MSD vs. 3-Way Quicksort

3-way radix quicksort collapses empty links in MSD tree.



MSD radix sort recursion tree
(1035 null links, not shown)



3-way radix quicksort recursion tree
(155 null links)

3-Way Radix Quicksort

```
private static void quicksortX(String a[], int lo, int hi, int d) {
   if (hi - lo <= 0) return;
   int i = lo-1, j = hi;
   int p = lo-1, q = hi;
   char v = a[hi].charAt(d);

   while (i < j) {                                         repeat until pointers cross
      while (a[++i].charAt(d) < v) if (i == hi) break;     find i on left and
      while (v < a[--j].charAt(d)) if (j == lo) break;     j on right to swap
      if (i > j) break;
      exch(a, i, j);
      if (a[i].charAt(d) == v) exch(a, ++p, i);            swap equal chars
      if (a[j].charAt(d) == v) exch(a, j, --q);            to left or right
   }
   if (p == q) {
      if (v != '\0') quicksortX(a, lo, hi, d+1);           special case for
      return;                                              all equal chars
   }
   if (a[i].charAt(d) < v) i++;
   for (int k = lo; k <= p; k++) exch(a, k, j--);          swap equal ones
   for (int k = hi; k >= q; k--) exch(a, k, i++);          back to middle
   quicksortX(a, lo, j, d);
   if ((i == hi) && (a[i].charAt(d) == v)) i++;            sort 3 pieces recursively
   if (v != '\0') quicksortX(a, j+1, i-1, d+1);
   quicksortX(a, i, hi, d);
}
```


Quicksort vs. 3-Way Radix Quicksort

Quicksort.
- 2N ln N string comparisons on average.
- Long keys are costly to compare if they differ only at the end, and this is common case!
- `absolutism`, `absolut`, `absolutely`, `absolute`.

3-way radix quicksort.
- Avoids re-comparing initial parts of the string.
- Uses just "enough" characters to resolve order.
- 2 N ln N character comparisons on average for random strings.
- Sub-linear sort for large W since input is of size NW.

Theorem. Quicksort with 3-way partitioning is OPTIMAL.
Pf. Ties cost to entropy. Beyond scope of 226.

String Sorting Performance

	String Sort	Suffix (sec)
	Worst Case	Moby Dick
Brute	W N^2	36,000 §
Quicksort	W N log N †	9.5
LSD *	W(N + R)	-
MSD	W(N + R)	395
MSD with cutoff	W(N + R)	6.8
3-way radix quicksort	W N log N †	2.8

R = radix
W = max length of string
N = number of strings

§ estimate
* fixed length strings only
† probabilistic guarantee

1.2 million for Moby Dick

Suffix Sorting: Worst Case Input

Length of longest match small.

- Hard to beat 3-way radix quicksort.

Length of longest match very long.

- 3-way radix quicksort is quadratic.
- Ex: two copies of Moby Dick.

Can we do better? Θ(N log N) ? Θ(N) ?

Observation. Must find longest repeated substring while suffix sorting to beat N^2.

```
abcdefghi
abcdefghiabcdefghi
bcdefghi
bcdefghiabcdefghi
cdefghi
cdefghiabcdefgh
defghi
efghiabcdefghi
efghi
fghiabcdefghi
fghi
ghiabcdefghi
fhi
hiabcdefghi
hi
iabcdefghi
i
```

Input: "abcdeghiabcdefghi"

Suffix Sorting in Linearithmic Time: Key Idea

				sorted
0	babaaaabcbabaaaaa0		17	0bab aaaa bcbabaaa aa
1	abaaaabcbabaaaaa0b		16	a0ba baaa abcbabaa aa
2	baaaabcbabaaaaa0ba		15	aa0b abaa aabcbaba aa
3	aaaabcbabaaaaa0bab		14	aaa0 baba aaabcbab aa
4	aaabcbabaaaaa0baba		3	aaaa bcba baaaaa0b ab
5	aabcbabaaaaa0babaa		12	aaaa a0ba baaaabcb ab
6	abcbabaaaaa0babaaa		13	aaaa 0bab aaaabcba ba
7	bcbabaaaaa0babaaaa		4	aaab cbab aaaaa0ba ba
8	cbabaaaaa0babaaaab		5	aabc baba aaaa0bab aa
9	babaaaaa0babaaaabc		1	abaa aabc babaaaaa 0b
10	abaaaaa0babaaaabcb		10	abaa aaa0 babaaaab cb
11	baaaaa0babaaaabcba		6	abcb abaa aaa0baba aa
12	aaaaa0babaaaabcbab		2	baaa abcb abaaaaa0 ba
13	aaaa0babaaaabcbaba		11	baaa aa0b abaaaabc ba
14	aaa0babaaaabcbabaa	0 + 4 = 4 →	0	baba aaab cbabaaaa a0
15	aa0babaaaabcbabaaa	9 + 4 = 13 →	9	baba aaaa 0babaaaa bc
16	a0babaaaabcbabaaaa		7	bcba baaa aa0babaa aa
17	0babaaaabcbabaaaaa		8	cbab aaaa a0babaaa ab

text = "babaaaabcbabaaaaa"

Suffix Sorting in Sub-Quadratic Time

Manber's MSD algorithm.

- Phase 0: sort on first character using key-indexed sorting.
- Phase i: given list of suffixes sorted on first 2^{i-1} characters, create list of suffixes sorted on first 2^i characters
- Finishes after lg N phases.

Manber's LSD algorithm.

- Same idea but go from right to left.
- O(N log N) guaranteed running time.
- O(N) extra space (but need several auxiliary arrays).

Best in theory. O(N) but more complicated to implement.

String Sorting Performance

	String Sort	Suffix Sort (seconds)	
	Worst Case	Moby Dick	AesopAesop
Brute	$W N^2$	36,000 §	3,990 §
Quicksort	W N log N †	9.5	167
LSD *	W(N + R)	-	-
MSD	W(N + R)	395	memory
MSD with cutoff	W(N + R)	6.8	162
3-way radix quicksort	W N log N †	2.8	400
Manber ‡	N log N	17	8.5

R = radix
W = max length of string.
N = number of strings

1.2 million for Moby Dick
191 thousand for Aesop's Fables

§ estimate
* fixed length strings only
† probabilistic guarantee
‡ suffix sorting only