

String Sorts

Strings in Java

Sequence of characters

Char data type

C char data type. Typically an 8-bit integer.

- Supports 7-bit ASCII.
- Can represent only 256 characters.

Java char data type. A 16-bit unsigned integer.

- Supports original 16-bit Unicode.
- Supports 21-bit Unicode 3.0 (awkwardly).

String data type

String data type (in Java). Sequence of characters (immutable).

Underlying implementation. Immutable `char[]` array, offset, and length.

String		
operation	guarantee	extra space
<code>length()</code>	1	1
<code>charAt()</code>	1	1
<code>substring()</code>	1	1
<code>concat()</code>	N	N

Memory. $40 + 2N$ bytes for a virgin String of length N .

can use `byte[]` or `char[]` instead of String to save space
(but lose convenience of String data type)

StringBuilder data type

StringBuilder data type. Sequence of characters (mutable).
Underlying implementation. Resizing `char[]` array and length.

operation	String		StringBuilder	
	guarantee	extra space	guarantee	extra space
<code>length()</code>	1	1	1	1
<code>charAt()</code>	1	1	1	1
<code>substring()</code>	1	1	N	N
<code>concat()</code>	N	N	1 *	1 *

* amortized

Alphabets

- **Digital key**---sequence of digits over fixed alphabet
- **Radix**---number of digits R in alphabet

Key-indexed Counting

Basisc

- Sort an array `a[]` of N integers between 0 and $R - 1$
 - count frequencies of each letter using key as index
 - Compute frequency cumulates which specify destinations
 - Access cumulates using keys as index to move items
 - Copy back into original array

Complexity

- Key-indexed counting uses $\sim 11N + 4$ array accesses to sort N items whose keys are integers between 0 and $R - 1$
- Key-indexed counting uses extra space proportional to $N + R$
- stable

LSD radix sort

Least-significant-digit-first sort

Basisc

- Consider characters from right to left
- Stably sort using d^{th} character as the key (using key-indexed counting)

Implementation

```
1 public class LSD {
2     private static final int BITS_PER_BYTE = 8;
3
4     // do not instantiate
5     private LSD() { }
6
7     public static void sort(String[] a, int w) {
8         int n = a.length;
9         int R = 256;    // extend ASCII alphabet size
10        String[] aux = new String[n];
11
12        for (int d = w-1; d >= 0; d--) {
13            // sort by key-indexed counting on dth character
14
15            // compute frequency counts
16            int[] count = new int[R+1];
17            for (int i = 0; i < n; i++)
18                count[a[i].charAt(d) + 1]++;
19
20            // compute cumulates
21            for (int r = 0; r < R; r++)
22                count[r+1] += count[r];
23
24            // move data
25            for (int i = 0; i < n; i++)
26                aux[count[a[i].charAt(d)]++] = a[i];
27
28            // copy back
29            for (int i = 0; i < n; i++)
30                a[i] = aux[i];
31        }
32    }
33
34    public static void sort(int[] a) {
35        final int BITS = 32;                // each int is 32 bits
36        final int R = 1 << BITS_PER_BYTE;  // each byte is between 0 and 255
37        final int MASK = R - 1;            // 0xFF
38        final int w = BITS / BITS_PER_BYTE; // each int is 4 bytes
39
40        int n = a.length;
41        int[] aux = new int[n];
42
43        for (int d = 0; d < w; d++) {
44
45            // compute frequency counts
46            int[] count = new int[R+1];
47            for (int i = 0; i < n; i++) {
48                int c = (a[i] >> BITS_PER_BYTE*d) & MASK;
49                count[c + 1]++;
```

```

50     }
51
52     // compute cumulates
53     for (int r = 0; r < R; r++)
54         count[r+1] += count[r];
55
56     // for most significant byte, 0x80-0xFF comes before 0x00-0x7F
57     if (d == w-1) {
58         int shift1 = count[R] - count[R/2];
59         int shift2 = count[R/2];
60         for (int r = 0; r < R/2; r++)
61             count[r] += shift1;
62         for (int r = R/2; r < R; r++)
63             count[r] -= shift2;
64     }
65
66     // move data
67     for (int i = 0; i < n; i++) {
68         int c = (a[i] >> BITS_PER_BYTE*d) & MASK;
69         aux[count[c]++] = a[i];
70     }
71
72     int[] temp = a;
73     a = aux;
74     aux = temp;
75 }
76
77
78 public static void main(String[] args) {
79     String[] a = StdIn.readAllStrings();
80     int n = a.length;
81
82     // check that strings have fixed length
83     int w = a[0].length();
84     for (int i = 0; i < n; i++)
85         assert a[i].length() == w : "strings must have fixed length";
86     sort(a, w);
87     for (int i = 0; i < n; i++)
88         StdOut.println(a[i]);
89 }
90 }

```

MSD radix sort

most-significant-digit-first sor

Basic

- Partition array into R pieces according to first character (use key-indexed counting)
- Recursively sort all strings that start with each character (key-indexed counts delineate subarrays to sort)

Variable-length strings

Treat strings as if they had an extra char at end (smaller than any char).

0	s	e	a	-1						
1	s	e	a	s	h	e	l	l	s	-1
2	s	e	l	l	s	-1				
3	s	h	e	-1						
4	s	h	e	-1						
5	s	h	e	l	l	s	-1			
6	s	h	o	r	e	-1				
7	s	u	r	e	l	y	-1			

why smaller?

she before shells

```
private static int charAt(String s, int d)
{
    if (d < s.length()) return s.charAt(d);
    else return -1;
}
```

Improvement

Problem

Observation 1. Much too slow for small subarrays.

- Each function call needs its own `count[]` array.
- ASCII (256 counts): 100x slower than copy pass for $N=2$.
- Unicode (65,536 counts): 32,000x slower for $N=2$.

Observation 2. Huge number of small subarrays because of recursion.

Solution

Cutoff to insertion sort

Performance

- MSD examines just enough characters to sort the keys
- Number of characters examined depends on keys
- Can be sublinear in input size

MSD vs. Quicksort

- Disadvantages of MSD
 - access memory randomly (cache inefficient)
 - Inner loop has a lot of instructions
 - Extra space for count[] and aux[]
- Disadvantage of quicksort
 - Linearithmic number of string compares (not linear)
 - Has to rescan many characters in keys with long prefix matches

Implementation

```
1 public class MSD {
2     private static final int BITS_PER_BYTE = 8;
3     private static final int BITS_PER_INT = 32; // each Java int is 32
4     private static final int R = 256; // extended ASCII alphabet
5     private static final int CUTOFF = 15; // cutoff to insertion
6     private static final int size;
7     // do not instantiate
8     private MSD() { }
9
10    public static void sort(String[] a) {
11        int n = a.length;
12        String[] aux = new String[n];
13        sort(a, 0, n-1, 0, aux);
14    }
15
16    // return dth character of s, -1 if d = length of string
17    private static int charAt(String s, int d) {
18        assert d >= 0 && d <= s.length();
19        if (d == s.length()) return -1;
20        return s.charAt(d);
21    }
22
23    // sort from a[lo] to a[hi], starting at the dth character
24    private static void sort(String[] a, int lo, int hi, int d, String[] aux)
25    {
26        // cutoff to insertion sort for small subarrays
27        if (hi <= lo + CUTOFF) {
28            insertion(a, lo, hi, d);
29            return;
30        }
31
32        // compute frequency counts
```

```

33     int[] count = new int[R+2];
34     for (int i = lo; i <= hi; i++) {
35         int c = charAt(a[i], d);
36         count[c+2]++;
37     }
38
39     // transform counts to indices
40     for (int r = 0; r < R+1; r++)
41         count[r+1] += count[r];
42
43     // distribute
44     for (int i = lo; i <= hi; i++) {
45         int c = charAt(a[i], d);
46         aux[count[c+1]++] = a[i];
47     }
48
49     // copy back
50     for (int i = lo; i <= hi; i++)
51         a[i] = aux[i - lo];
52
53
54     // recursively sort for each character (excludes sentinel -1)
55     for (int r = 0; r < R; r++)
56         sort(a, lo + count[r], lo + count[r+1] - 1, d+1, aux);
57 }
58
59
60 // insertion sort a[lo..hi], starting at dth character
61 private static void insertion(String[] a, int lo, int hi, int d) {
62     for (int i = lo; i <= hi; i++)
63         for (int j = i; j > lo && less(a[j], a[j-1], d); j--)
64             exch(a, j, j-1);
65 }
66
67 // exchange a[i] and a[j]
68 private static void exch(String[] a, int i, int j) {}
69
70 // is v less than w, starting at character d
71 private static boolean less(String v, String w, int d) {
72     // assert v.substring(0, d).equals(w.substring(0, d));
73     for (int i = d; i < Math.min(v.length(), w.length()); i++) {
74         if (v.charAt(i) < w.charAt(i)) return true;
75         if (v.charAt(i) > w.charAt(i)) return false;
76     }
77     return v.length() < w.length();
78 }
79
80 public static void sort(int[] a) {
81     int n = a.length;
82     int[] aux = new int[n];
83     sort(a, 0, n-1, 0, aux);
84 }

```

```

85
86 // MSD sort from a[lo] to a[hi], starting at the dth byte
87 private static void sort(int[] a, int lo, int hi, int d, int[] aux) {
88
89     // cutoff to insertion sort for small subarrays
90     if (hi <= lo + CUTOFF) {
91         insertion(a, lo, hi);
92         return;
93     }
94
95     // compute frequency counts (need R = 256)
96     int[] count = new int[R+1];
97     int mask = R - 1;    // 0xFF;
98     int shift = BITS_PER_INT - BITS_PER_BYTE*d - BITS_PER_BYTE;
99     for (int i = lo; i <= hi; i++) {
100         int c = (a[i] >> shift) & mask;
101         count[c + 1]++;
102     }
103
104     // transform counts to indices
105     for (int r = 0; r < R; r++)
106         count[r+1] += count[r];
107
108     // for most significant byte, 0x80-0xFF comes before 0x00-0x7F
109     if (d == 0) {
110         int shift1 = count[R] - count[R/2];
111         int shift2 = count[R/2];
112         count[R] = shift1 + count[1];    // to simplify recursive calls
113
114         later
115         for (int r = 0; r < R/2; r++)
116             count[r] += shift1;
117         for (int r = R/2; r < R; r++)
118             count[r] -= shift2;
119     }
120
121     // distribute
122     for (int i = lo; i <= hi; i++) {
123         int c = (a[i] >> shift) & mask;
124         aux[count[c]++] = a[i];
125     }
126
127     // copy back
128     for (int i = lo; i <= hi; i++)
129         a[i] = aux[i - lo];
130
131     // no more bits
132     if (d == 3) return;
133
134     // special case for most significant byte
135     if (d == 0 && count[R/2] > 0)
136         sort(a, lo, lo + count[R/2] - 1, d+1, aux);

```



```

136         // special case for other bytes
137         if (d != 0 && count[0] > 0)
138             sort(a, lo, lo + count[0] - 1, d+1, aux);
139
140         // recursively sort for each character
141         // (could skip r = R/2 for d = 0 and skip r = R for d > 0)
142         for (int r = 0; r < R; r++)
143             if (count[r+1] > count[r])
144                 sort(a, lo + count[r], lo + count[r+1] - 1, d+1, aux);
145     }
146
147     // insertion sort a[lo..hi]
148     private static void insertion(int[] a, int lo, int hi) {
149         for (int i = lo; i <= hi; i++)
150             for (int j = i; j > lo && a[j] < a[j-1]; j--)
151                 exch(a, j, j-1);
152     }
153
154     public static void main(String[] args) {}
155 }

```

3-way radix quicksort

Basic

- Do 3-way partitioning on the d^{th} character
 - Less overhead than R -way partitioning in MSD string sort
 - Doesn't re-examine characters equal to the partitioning char

vs. Quick Sort

Standard quicksort.

- Uses $\sim 2N \ln N$ **string compares** on average.
- Costly for keys with long common prefixes (and this is a common case!)

3-way string (radix) quicksort.

- Uses $\sim 2N \ln N$ **character compares** on average for random strings.
- Avoids re-comparing long common prefixes.

vs. MDS sort

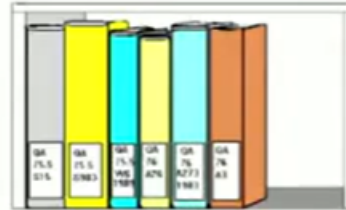
MSD string sort.

- Is cache-inefficient.
- Too much memory storing count[].
- Too much overhead reinitializing count[] and aux[].

3-way string quicksort.

- Has a short inner loop.
- Is cache-friendly.
- Is in-place.

library of Congress call numbers



Implementation

```
1 public class Quick3string {
2     private static final int CUTOFF = 15;    // cutoff to insertion sort
3
4     // do not instantiate
5     private Quick3string() { }
6
7     public static void sort(String[] a) {
8         StdRandom.shuffle(a);
9         sort(a, 0, a.length-1, 0);
10        assert isSorted(a);
11    }
12
13    // return the dth character of s, -1 if d = length of s
14    private static int charAt(String s, int d) {
15        assert d >= 0 && d <= s.length();
16        if (d == s.length()) return -1;
17        return s.charAt(d);
18    }
19
20
21    // 3-way string quicksort a[lo..hi] starting at dth character
22    private static void sort(String[] a, int lo, int hi, int d) {
23
24        // cutoff to insertion sort for small subarrays
25        if (hi <= lo + CUTOFF) {
26            insertion(a, lo, hi, d);
27            return;
28        }
29
30        int lt = lo, gt = hi;
31        int v = charAt(a[lo], d);
32        int i = lo + 1;
33        while (i <= gt) {
34            int t = charAt(a[i], d);
35            if (t < v) exch(a, lt++, i++);
```

```

36         else if (t > v) exch(a, i, gt--);
37         else          i++;
38     }
39
40     // a[lo..lt-1] < v = a[lt..gt] < a[gt+1..hi].
41     sort(a, lo, lt-1, d);
42     if (v >= 0) sort(a, lt, gt, d+1);
43     sort(a, gt+1, hi, d);
44 }
45
46 // sort from a[lo] to a[hi], starting at the dth character
47 private static void insertion(String[] a, int lo, int hi, int d) {
48     for (int i = lo; i <= hi; i++)
49         for (int j = i; j > lo && less(a[j], a[j-1], d); j--)
50             exch(a, j, j-1);
51 }
52
53 // exchange a[i] and a[j]
54 private static void exch(String[] a, int i, int j) {
55     String temp = a[i];
56     a[i] = a[j];
57     a[j] = temp;
58 }
59
60 private static boolean less(String v, String w, int d) {
61     assert v.substring(0, d).equals(w.substring(0, d));
62     for (int i = d; i < Math.min(v.length(), w.length()); i++) {
63         if (v.charAt(i) < w.charAt(i)) return true;
64         if (v.charAt(i) > w.charAt(i)) return false;
65     }
66     return v.length() < w.length();
67 }
68
69 // is the array sorted
70 private static boolean isSorted(String[] a) {
71     for (int i = 1; i < a.length; i++)
72         if (a[i].compareTo(a[i-1]) < 0) return false;
73     return true;
74 }
75
76 public static void main(String[] args) {
77
78     // read in the strings from standard input
79     String[] a = StdIn.readAllStrings();
80     int n = a.length;
81
82     // sort the strings
83     sort(a);
84
85     // print the results
86     for (int i = 0; i < n; i++)
87         StdOut.println(a[i]);

```

```

88 |     }
89 | }

```

Comparison

algorithm	guarantee	random	extra space	stable?	operations on keys
insertion sort	$N^2 / 2$	$N^2 / 4$	1	yes	compareTo()
mergesort	$N \lg N$	$N \lg N$	N	yes	compareTo()
quicksort	$1.39 N \lg N$	$1.39 N \lg N$	$c \lg N$	no	compareTo()
heapsort	$2 N \lg N$	$2 N \lg N$	1	no	compareTo()
LSD †	$2 N W$	$2 N W$	$N + R$	yes	charAt()
MSD †	$2 N W$	$N \log_{\#} N$	$N + D R$	yes	charAt()
3-way string quicksort	$1.39 W N \lg N$	$1.39 N \lg N$	$\log N + W$	no	charAt()

Suffix arrays

Keyword-in-context search

- Steps
 - form suffixes
 - sort suffixes to bring repeated substrings together
 - binary search for query; scan until mismatch

Longest repeated substring

- Applications
 - Bioinformatics
 - cryptanalysis
 - music notes

Brute-force Algorithm

- Steps
 - Try all indices i and j for start of possible match
 - Compute longest common prefix (LCP) for each pair
 - Running time $\leq DN^2$, where D is the length of longest match

A Sorting Solution

- Steps
 - form suffixes
 - sort suffixes to bring repeated substrings together

```
public String lrs(String s)
{
    int N = s.length();

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

    Arrays.sort(suffixes);

    String lrs = "";
    for (int i = 0; i < N-1; i++)
    {
        int len = lcp(suffixes[i], suffixes[i+1]);
        if (len > lrs.length())
            lrs = suffixes[i].substring(0, len);
    }
    return lrs;
}
```

create suffixes
(linear time and space)

sort suffixes

find LCP between
adjacent suffixes in
sorted order

Challenge

Bad input: longest repeated substring very long.

- Ex: same letter repeated N times.
- Ex: two copies of the same Java codebase.

form suffixes	sorted suffixes
0 t w i n s t w i n s	9 i n s
1 w i n s t w i n s	8 i n s t w i n s
2 i n s t w i n s	7 n s
3 n s t w i n s	6 n s t w i n s
4 s t w i n s	5 s
5 t w i n s	4 s t w i n s
6 w i n s	3 t w i n s
7 i n s	2 t w i n s t w i n s
8 n s	1 w i n s
9 s	0 w i n s t w i n s

D = length of longest match

LRS needs at least $1 + 2 + 3 + \dots + D$ character compares.

Running time. Quadratic (or worse) in D for LRS (also for sort).

Manber-Myers MSD Algorithm

Suffix sorting in linearithmic time

- Step
 - Phase 0---sort on firsts character using key-indexed counting sort
 - Phase i ---given array of suffixes sorted on first 2^{i-1} characters, create array of suffixes sorted on first 2^i characters
- constant-time string compare by indexing into inverse

original suffixes		index sort (first four characters)		inverse
0 b a b a a a a b c b a b a a a a a 0		17 0		0 14
1 a b a a a a b c b a b a a a a a 0		16 a 0		1 9
2 b a a a a b c b a b a a a a a 0		15 a a 0		2 12
3 a a a a b c b a b a a a a a 0		14 a a a 0		3 4
4 a a a b c b a b a a a a a 0		3 a a a a b c b a b a a a a a 0		4 7
5 a a b c b a b a a a a a 0		17 a a a a a 0		5 8
6 a b c b a b a a a a a 0		13 a a a a 0		6 11
7 b c b a b a a a a a 0		4 a a a b c b a b a a a a a 0		7 16
8 c b a b a a a a a 0		5 a a b c b a b a a a a a 0		8 17
9 b a b a a a a a 0		1 a b a a a a b c b a b a a a a a 0		9 15
10 a b a a a a a 0		10 a b a a a a a 0		10 10
11 b a a a a a 0		6 a b c b a b a a a a a 0		11 13
12 a a a a a 0		2 b a a a a b c b a b a a a a a 0		12 5
13 a a a a 0		11 b a a a a 0		13 6
14 a a a 0		0 b a b a a a a b c b a b a a a a a 0		14 3
15 a a 0		9 b a b a a a a a 0		15 2
16 a 0		7 b c b a b a a a a a 0		16 1
17 0		8 c b a b a a a a a 0		17 0

$0 + 4 = 4$
 $9 + 4 = 13$

suffixes_i[13] < suffixes_i[4] (because inverse[13] < inverse[4])
 so suffixes_a[9] < suffixes_a[0]