Introduction to Algorithms

Lecture 8: Counting Sort Algorithm

Yao-Chung Fan
yfan@nchu.edu.tw

COUNTING SORT

- *key-indexed counting sort*
- *LSD radix sort*
- *MSD radix sort*

Review: summary of the performance of sorting algorithms

Frequency of operations.

algorithm	guarantee	random	stable?	operations on keys
insertion sort	$\frac{1}{2} N^2$	$\frac{1}{4} N^2$	✔	`compareTo()`
mergesort	$N \lg N$	$N \lg N$	✔	`compareTo()`
quicksort	$1.39\ N \lg N$ *	$1.39\ N \lg N$		`compareTo()`
heapsort	$2\ N \lg N$	$2\ N \lg N$		`compareTo()`

* probabilistic

Lower bound. $\sim N \lg N$ compares required by any compare-based algorithm.

Q. Can we do better (despite the lower bound)?

A. Yes, if we don't depend on key compares. ← use array accesses to make R-way decisions (instead of binary decisions)

Key-indexed counting: assumptions about keys

Assumption. Keys are integers between 0 and $R - 1$.

Implication. Can use key as an array index.

Applications.

- Sort string by first letter.
- Sort class roster by section.
- Sort phone numbers by area code.

input *name*	*section*	sorted result *(by section)*	
Anderson	2	Harris	1
Brown	3	Martin	1
Davis	3	Moore	1
Garcia	4	Anderson	2
Harris	1	Martinez	2
Jackson	3	Miller	2
Johnson	4	Robinson	2
Jones	3	White	2
Martin	1	Brown	3
Martinez	2	Davis	3
Miller	2	Jackson	3
Moore	1	Jones	3
Robinson	2	Taylor	3
Smith	4	Williams	3
Taylor	3	Garcia	4
Thomas	4	Johnson	4
Thompson	4	Smith	4
White	2	Thomas	4
Williams	3	Thompson	4
Wilson	4	Wilson	4

keys are small integers

Key-indexed counting demo

Goal. Sort an array `a[]` of N integers between 0 and $R - 1$.

R = 6

- Count frequencies of each letter using key as index.
- Compute frequency cumulates which specify destinations.
- Access cumulates using key as index to move items.
- Copy back into original array.

```
int N = a.length;
int[] count = new int[R+1];

for (int i = 0; i < N; i++)
   count[a[i]+1]++;

for (int r = 0; r < R; r++)
   count[r+1] += count[r];

for (int i = 0; i < N; i++)
   aux[count[a[i]]++] = a[i];

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

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

use a for 0
b for 1
c for 2
d for 3
e for 4
f for 5

Key-indexed counting demo

Goal. 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 key as index to move items.
- Copy back into original array.

```
int N = a.length;
int[] count = new int[R+1];

for (int i = 0; i < N; i++)
   count[a[i]+1]++;

for (int r = 0; r < R; r++)
   count[r+1] += count[r];

for (int i = 0; i < N; i++)
   aux[count[a[i]]++] = a[i];

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

copy back

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

r	count[r]
a	2
b	5
c	6
d	8
e	9
f	12
-	12

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

Key-indexed counting: analysis

Proposition. Key-indexed takes time proportional to $N + R$.

Proposition. Key-indexed counting uses extra space proportional to $N + R$.

Stable? ✔

a[0]	Anderson	2	Harris	1	aux[0]
a[1]	Brown	3	Martin	1	aux[1]
a[2]	Davis	3	Moore	1	aux[2]
a[3]	Garcia	4	Anderson	2	aux[3]
a[4]	Harris	1	Martinez	2	aux[4]
a[5]	Jackson	3	Miller	2	aux[5]
a[6]	Johnson	4	Robinson	2	aux[6]
a[7]	Jones	3	White	2	aux[7]
a[8]	Martin	1	Brown	3	aux[8]
a[9]	Martinez	2	Davis	3	aux[9]
a[10]	Miller	2	Jackson	3	aux[10]
a[11]	Moore	1	Jones	3	aux[11]
a[12]	Robinson	2	Taylor	3	aux[12]
a[13]	Smith	4	Williams	3	aux[13]
a[14]	Taylor	3	Garcia	4	aux[14]
a[15]	Thomas	4	Johnson	4	aux[15]
a[16]	Thompson	4	Smith	4	aux[16]
a[17]	White	2	Thomas	4	aux[17]
a[18]	Williams	3	Thompson	4	aux[18]
a[19]	Wilson	4	Wilson	4	aux[19]

COUNTING SORT

- *key-indexed counting sort*
- *LSD radix sort*
- *MSD radix sort*

Least-significant-digit-first sort

LSD string (radix) sort.

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

從最右邊的開始排，最後會產生從左邊往右看的sort

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

sort is stable
(arrows do not cross)

LSD string sort: correctness proof

Proposition. LSD sorts fixed-length strings in ascending order.

Pf. [by induction on i]

After pass i, strings are sorted by last i characters.

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

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

sorted from previous passes (by induction)

sort key

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

Proposition. LSD sort is stable.

Pf. Key-indexed counting is stable.

LSD string sort: Java implementation

```
public class LSD
{
   public static void sort(String[] a, int W)
   {
      int R = 256;
      int N = a.length;
      String[] aux = new String[N];

      for (int d = W-1; d >= 0; d--)
      {
         int[] count = new int[R+1];
         for (int i = 0; i < N; i++)
            count[a[i].charAt(d) + 1]++;
         for (int r = 0; r < R; r++)
            count[r+1] += count[r];
         for (int i = 0; i < N; i++)
            aux[count[a[i].charAt(d)]++] = a[i];
         for (int i = 0; i < N; i++)
            a[i] = aux[i];
      }
   }
}
```

fixed-length W strings

radix R

do key-indexed counting for each digit from right to left

key-indexed counting

Summary of the performance of sorting algorithms

Frequency of operations.

algorithm	guarantee	random	stable?
insertion sort	$\frac{1}{2} N^2$	$\frac{1}{4} N^2$	✔
mergesort	$N \lg N$	$N \lg N$	✔
quicksort	$1.39\, N \lg N$ *	$1.39\, N \lg N$	
heapsort	$2\, N \lg N$	$2\, N \lg N$	
LSD sort †	$2\, W (N + R)$	$2\, W (N + R)$	✔

理論上比其他sort快，但要一直做array的讀取，實際上會很慢，適合用在值域(R)較少的情況下

Q. What if strings are not all of same length?

String sorting challenge 1

Problem. Sort a huge commercial database on a fixed-length key.

Ex. Account number, date, Social Security number, ...

Which sorting method to use?

- Insertion sort.
- Mergesort.
- Quicksort.
- Heapsort.
- ✓ LSD string sort.

256 (or 65,536) counters;
Fixed-length strings sort in W passes.

B14-99-8765
756-12-AD46
CX6-92-0112
332-WX-9877
375-99-QWAX
CV2-59-0221
[illegible]87-SS-0321
KJ-0[illegible]12388
715-YT-013C
MJ0-PP-983F
908-KK-33TY
BBN-63-23RE
48G-BM-912D
982-ER-9P1B
WBL-37-PB81
810-F4-J87Q
LE9-N8-XX76
908-KK-33TY
B14-99-8765
CX6-92-0112
CV2-59-0221
332-WX-23SQ
332-6A-9877

String sorting interview question

Problem. Sort one million 32-bit string.

Ex. Google (or presidential) interview.

Which sorting method to use?

- Insertion sort.
- Mergesort.
- Quicksort.
- Heapsort.
- LSD string sort.

Reverse LSD

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

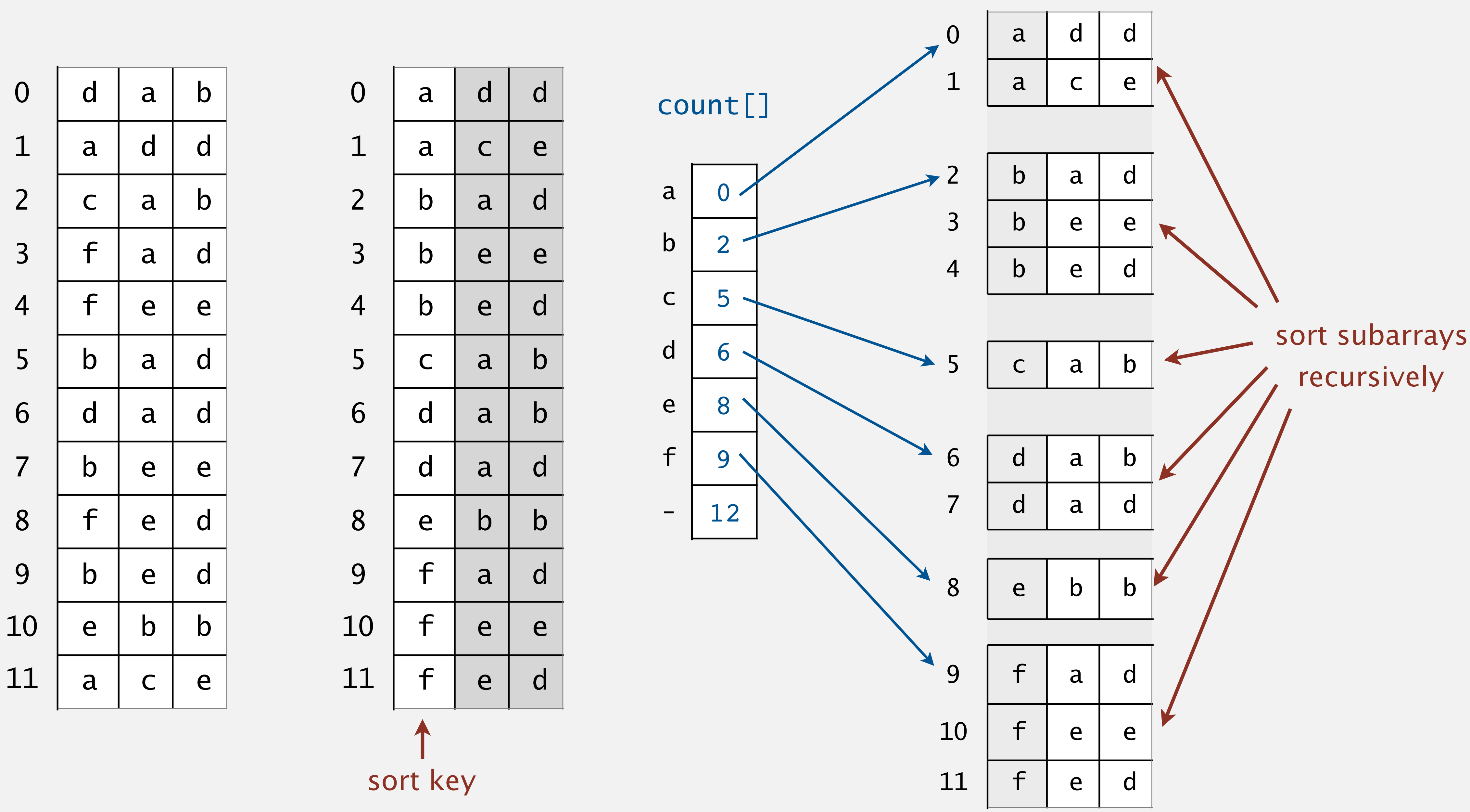
					sort key (d = 0)					sort key (d = 1)					sort key (d = 2)
0	d	a	b	0	a	d	d	0	b	a	d	0	c	a	b
1	a	d	d	1	a	c	e	1	c	a	b	1	d	a	b
2	c	a	b	2	b	a	d	2	d	a	b	2	e	b	b
3	f	a	d	3	b	e	e	3	d	a	d	3	b	a	d
4	f	e	e	4	b	e	d	4	f	a	d	4	d	a	d
5	b	a	d	5	c	a	b	5	e	b	b	5	f	a	d
6	d	a	d	6	d	a	b	6	a	c	e	6	a	d	d
7	b	e	e	7	d	a	d	7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

not sorted!

Most-significant-digit-first string sort

MSD (radix) sort.

- 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).



MSD string sort: top-level trace

use key-indexed counting on first character

recursively sort subarrays

i	input		*distribute and copy back*		output
0	she		are		are
1	sells		by		by
2	seashells		she		sea
3	by		sells		seashells
4	the		seashells		seashells
5	sea		sea		sells
6	shore		shore		sells
7	the		shells		she
8	shells		she		she
9	she		sells		shells
10	sells		surely		shore
11	are		seashells		surely
12	surely		the		the
13	seashells		the		the

r	char	*count frequencies*	*transform counts to indices*	*indices at completion of distribute phase*	recursive calls
0		0	0	0	sort(a, 0, 0);
1	a	0	0	1	sort(a, 1, 1);
2	b	1	1	2	sort(a, 2, 1);
3	c	1	2	2	sort(a, 2, 1);
4	d	0	2	2	sort(a, 2, 1);
5	e	0	2	2	sort(a, 2, 1);
6	f	0	2	2	sort(a, 2, 1);
7	g	0	2	2	sort(a, 2, 1);
8	h	0	2	2	sort(a, 2, 1);
9	i	0	2	2	sort(a, 2, 1);
10	j	0	2	2	sort(a, 2, 1);
11	k	0	2	2	sort(a, 2, 1);
12	l	0	2	2	sort(a, 2, 1);
13	m	0	2	2	sort(a, 2, 1);
14	n	0	2	2	sort(a, 2, 1);
15	o	0	2	2	sort(a, 2, 1);
16	p	0	2	2	sort(a, 2, 1);
17	q	0	2	2	sort(a, 2, 1);
18	r	0	2	2	sort(a, 2, 11);
19	s	0	2	12	sort(a, 12, 13);
20	t	10	12	14	sort(a, 14, 13);
21	u	2	14	14	sort(a, 14, 13);
22	v	0	14	14	sort(a, 14, 13);
23	w	0	14	14	sort(a, 14, 13);
24	x	0	14	14	sort(a, 14, 13);
25	y	0	14	14	sort(a, 14, 13);
26	z	0	14	14	sort(a, 14, 13);
27		0	14	14	sort(a, 14, 13);

start of s subarray

1 + end of s subarray

MSD string sort: example

input		d						
she	are	are	are	are	are	are	are	are
sells	by	by	by	by	by	by	by	by
seashells	she	lo sells	seashells	sea	sea	sea	sea	sea
by	sells	seashells	sea	seashells	seashells	seashells	seashells	seashells
the	seashells	sea	seashells	seashells	seashells	seashells	seashells	seashells
sea	sea	sells	sells	sells	sells	sells	sells	sells
shore	shore	seashells	sells	sells	sells	sells	sells	sells
the	shells	she	she	she	she	she	she	she
shells	she	shore	shore	shore	shore	shore	shore	shore
she	sells	shells	shells	shells	shells	shells	shells	shells
sells	surely	she	she	she	she	she	she	she
are	seashells	surely	surely	surely	surely	surely	surely	surely
surely	the	hi the	the	the	the	the	the	the
seashells	the	the	the	the	the	the	the	the

need to examine every character in equal keys

end of string goes before any char value

							output
are	are	are	are	are	are	are	are
by	by	by	by	by	by	by	by
sea	sea	sea	sea	sea	sea	sea	sea
seashells	seashells	seashells	seashells	seashells	seashells	seashells	seashells
seashells	seashells	seashells	seashells	seashells	seashells	seashells	seashells
sells	sells	sells	sells	sells	sells	sells	sells
sells	sells	sells	sells	sells	sells	sells	sells
she	she	she	she	she	she	she	she
shore	sshore	shore	shells	she	she	she	she
shells	hells	shells	she	shells	shells	shells	shells
she	she	she	shore	shore	shore	shore	shore
surely	surely	surely	surely	surely	surely	surely	surely
the	the	the	the	the	the	the	the
the	the	the	the	the	the	the	the

Trace of recursive calls for MSD string sort (no cutoff for small subarrays, subarrays of size 0 and 1 omitted)

Variable-length strings

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

why smaller?

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			

she before shells

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

MSD string sort: Java implementation

```
public static void sort(String[] a)
{
   aux = new String[a.length];
   sort(a, aux, 0, a.length - 1, 0);
}

private static void sort(String[] a, String[] aux, int lo, int hi, int d)
{
   if (hi <= lo) return;
   int[] count = new int[R+2];
   for (int i = lo; i <= hi; i++)
      count[charAt(a[i], d) + 2]++;
   for (int r = 0; r < R+1; r++)
      count[r+1] += count[r];
   for (int i = lo; i <= hi; i++)
      aux[count[charAt(a[i], d) + 1]++] = a[i];
   for (int i = lo; i <= hi; i++)
      a[i] = aux[i - lo];

   for (int r = 0; r < R; r++)
      sort(a, aux, lo + count[r], lo + count[r+1] - 1, d+1);
}
```

recycles aux[] array but not count[] array

key-indexed counting

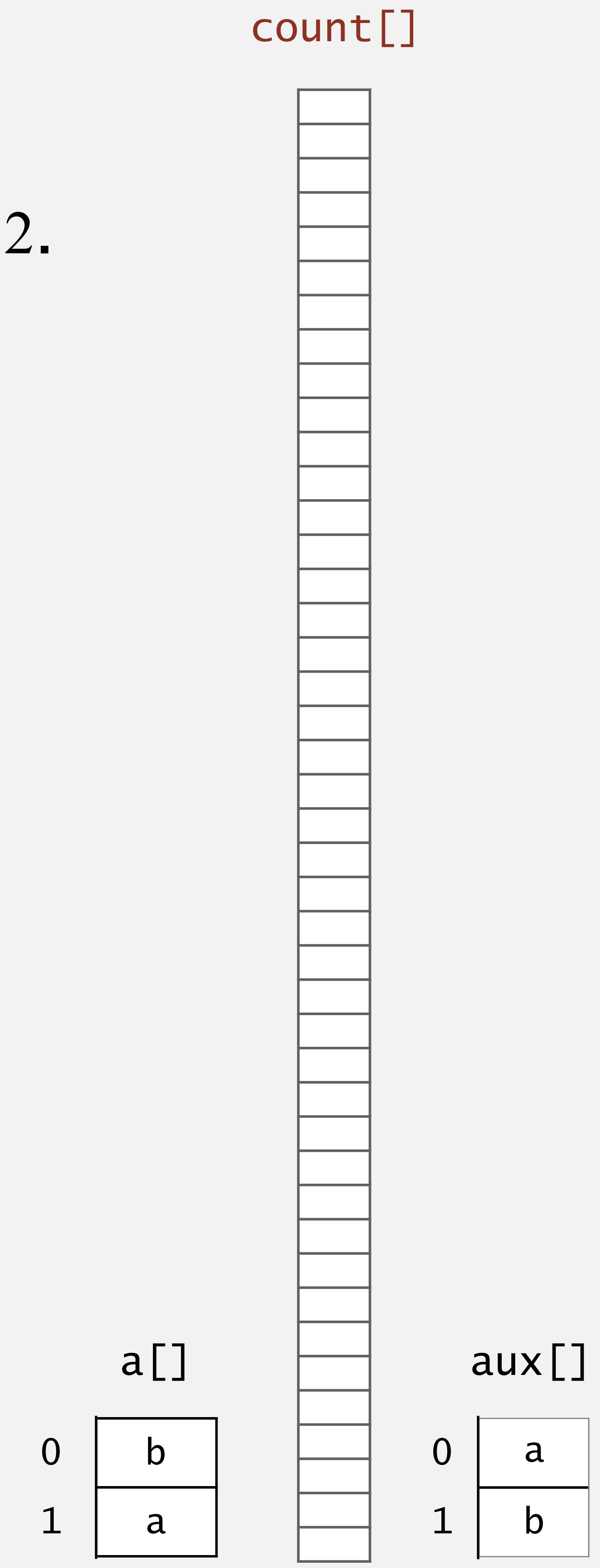
sort R subarrays recursively

MSD string sort: potential for disastrous performance

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.



MSD string sort: performance

Number of characters examined.

- MSD examines just enough characters to sort the keys.
- Number of characters examined depends on keys.

111
222
333
444

Random (sublinear)	Non-random with duplicates (nearly linear)	Worst case (linear)
1EIO402	are	1DNB377
1HYL490	by	1DNB377
1ROZ572	sea	1DNB377
2HXE734	seashells	1DNB377
2IYE230	seashells	1DNB377
2XOR846	sells	1DNB377
3CDB573	sells	1DNB377
3CVP720	she	1DNB377
3IGJ319	she	1DNB377
3KNA382	shells	1DNB377
3TAV879	shore	1DNB377
4CQP781	surely	1DNB377
4QGI284	the	1DNB377
4YHV229	the	1DNB377

Characters examined by MSD string sort

Summary of the performance of sorting algorithms

Frequency of operations.

algorithm	guarantee	random	extra space	stable?
insertion sort	½ N^2	¼ N^2	1	✔
mergesort	$N \lg N$	$N \lg N$	N	✔
quicksort	$1.39\, N \lg N$ *	$1.39\, N \lg N$	$c \lg N$	
heapsort	$2\, N \lg N$	$2\, N \lg N$	1	
LSD sort †	$2\, W (N + R)$	$2\, W (N + R)$	$N + R$	✔
MSD sort ‡	$2\, W (N + R)$	$N \log_R N$	$N + D R$	✔

D = function-call stack depth
(length of longest prefix match)

1024—>512—>256—>128—>.......

* probabilistic
† fixed-length W keys
‡ average-length W keys