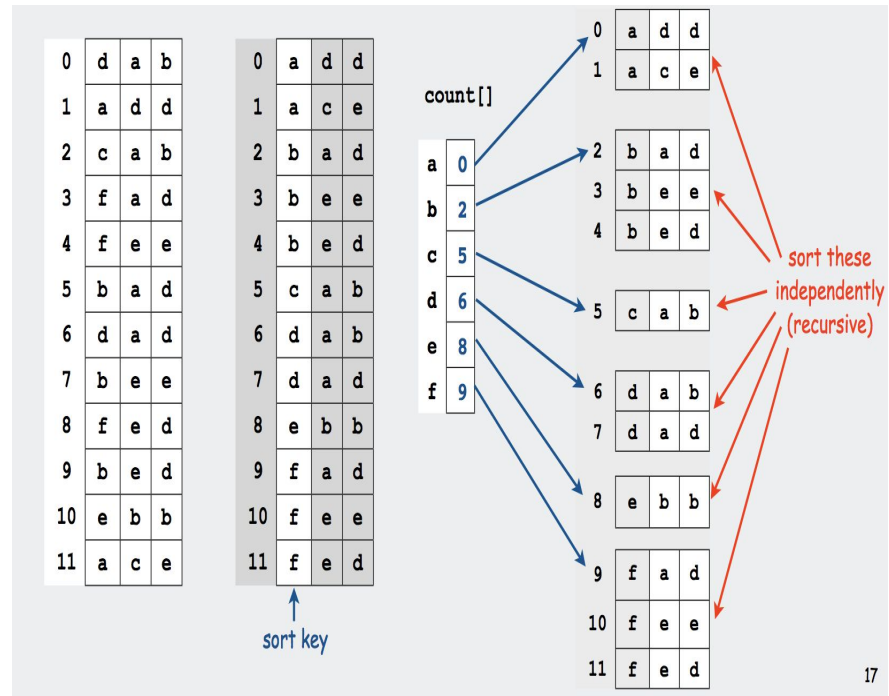

MSD Radix Sort

— Lindsey Chapman —
Tae Yeon Kim

What is MSD Radix Sort?

- Most Significant Digit (MSD) radix sort
- Redistributes digits in left to right order at the leftmost digit into buckets
- Recursion occurs within the buckets to further sort
- Key indexed counting and bucket sort rather than comparisons



Code Implementation

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

private static void msd(String[] a, int lo, int hi, int d)
{
    if (hi <= lo + 1) return;
    int[] count = new int[256+1];
    for (int i = 0; i < N; i++)
        count[a[i].charAt(d) + 1]++;
    for (int k = 1; k < 256; k++)
        count[k] += count[k-1];
    for (int i = 0; i < N; i++)
        temp[count[a[i].charAt(d)]++] = a[i];
    for (int i = 0; i < N; i++)
        a[i] = temp[i];
    for (int i = 0; i < 255; i++)
        msd(a, 1 + count[i], 1 + count[i+1], d+1);
}
```

key-indexed
counting →

← count
frequencies

← compute
cumulates

← move
records

← copy back

MSD vs LSD

- When working with integers of fixed length, MSD is more efficient than LSD because it may not have to examine every digit of each integer

| | | | |
|---|---|---|---|
| 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 |

← 19/24 \approx 80% of the characters examined

MSD vs LSD

- MSD can be used to sort strings of variable length, unlike LSD

| | | | | | | | | | | |
|---|---|---|---|----|---|----|----|----|----|----|
| 0 | a | c | e | t | o | n | e | \0 | | |
| 1 | a | d | d | i | t | i | o | n | \0 | |
| 2 | b | a | d | g | e | \0 | | | | |
| 3 | b | e | d | a | z | z | l | e | d | \0 |
| 4 | b | e | e | h | i | v | e | \0 | | |
| 5 | c | a | b | i | n | e | t | r | y | \0 |
| 6 | d | a | b | b | l | e | \0 | | | |
| 7 | d | a | d | \0 | | | | | | |

MSD vs LSD

- MSD uses recursion, so it requires more space than LSD
 - This means that MSD is much slower than LSD when working with small files--however, we can switch to another sorting algorithm for small values of N
- LSD has to be stable in order to work correctly, but MSD can either be made stable or unstable

Complexity Comparisons

| | stable? | inplace? | run time? | extra space | sweet spot |
|------------------------|-----------|----------|----------------------------|-------------|----------------------------|
| <i>LSD string sort</i> | yes | no | $O(NM)$ | $O(N + B)$ | short fixed-length strings |
| <i>MSD string sort</i> | sometimes | no | Between $O(N)$ and $O(Nm)$ | $O(N + MB)$ | random strings |

N = number of strings to sort

R = number of characters in alphabet

m = average length of string

M = max length of string

Time and Space Complexity of MSD

- Best case time complexity is $O(N)$ and worst case time complexity is $O(Nm)$, where N is the number of entries and m is the average length of the strings
 - LSD has time complexity $O(NM)$ in both the best and worst case scenario, where M is the fixed length of all the strings
- MSD requires $O(N + MB)$, where B is the size of the radix
 - The size of the radix corresponds to the range of values that a given type can take--for example, there are $256 = B$ possible characters in a string, but only 10 possible characters in an integer in base 10, and there are 2 possible characters in binary
 - LSD requires only $O(N + B)$ extra space

Example

| use key-indexed counting on first character | | | | | recursively sort subarrays | |
|---|-------------------|-----------------------------|--------------------------|---|----------------------------|-------------|
| | count frequencies | transform counts to indices | distribute and copy back | indices at completion of distribute phase | | |
| 0 | she | 0 a 0 | 0 are | 0 0 0 | sort(a, 0, 0, 1); | 0 are |
| 1 | sells | 1 b 1 | 1 by | 1 a 1 | sort(a, 1, 1, 1); | 1 by |
| 2 | seashells | 3 c 1 | 2 she | 2 b 2 | sort(a, 2, 1, 1); | 2 sea |
| 3 | by | 4 d 0 | 3 sells | 3 c 2 | sort(a, 2, 1, 1); | 3 seashells |
| 4 | the | 5 e 0 | 4 seashells | 4 d 2 | sort(a, 2, 1, 1); | 4 seashells |
| 5 | sea | 6 f 0 | 5 sea | 5 e 2 | sort(a, 2, 1, 1); | 5 sells |
| 6 | shore | 7 g 0 | 6 shore | 6 f 2 | sort(a, 2, 1, 1); | 6 sells |
| 7 | the | 8 h 0 | 7 shells | 7 g 2 | sort(a, 2, 1, 1); | 7 she |
| 8 | shells | 9 i 0 | 8 she | 8 h 2 | sort(a, 2, 1, 1); | 8 she |
| 9 | she | 10 j 0 | 9 sells | 9 i 2 | sort(a, 2, 1, 1); | 9 shells |
| 10 | sells | 11 k 0 | 10 surely | 10 j 2 | sort(a, 2, 1, 1); | 10 shore |
| 11 | are | 12 l 0 | 11 seashells | 11 k 2 | sort(a, 2, 1, 1); | 11 surely |
| 12 | surely | 13 m 0 | 12 the | 12 l 2 | sort(a, 2, 11, 1); | 12 the |
| 13 | seashells | 14 n 0 | 13 the | 13 m 2 | sort(a, 2, 13, 1); | 13 the |
| | | 15 o 0 | | 14 n 2 | sort(a, 14, 13, 1); | |
| | | 16 p 0 | | 15 o 2 | sort(a, 14, 13, 1); | |
| | | 17 q 0 | | 16 p 2 | sort(a, 14, 13, 1); | |
| | | 18 r 0 | | 17 q 2 | sort(a, 14, 13, 1); | |
| | | 19 s 0 | | 18 r 2 | sort(a, 14, 13, 1); | |
| | | 20 t 10 | | 19 s 12 | sort(a, 14, 13, 1); | |
| | | 21 u 2 | | 20 t 14 | sort(a, 14, 13, 1); | |
| | | 22 v 0 | | 21 u 14 | sort(a, 14, 13, 1); | |
| | | 23 w 0 | | 22 v 14 | sort(a, 14, 13, 1); | |
| | | 24 x 0 | | 23 w 14 | sort(a, 14, 13, 1); | |
| | | 25 y 0 | | 24 x 14 | sort(a, 14, 13, 1); | |
| | | 26 z 0 | | 25 y 14 | sort(a, 14, 13, 1); | |
| | | 27 0 | | 26 z 14 | sort(a, 14, 13, 1); | |
| | | | | 27 14 | sort(a, 14, 13, 1); | |

start of s subarray
1 + end of s subarray

Trace of MSD string sort: top level of sort(a, 0, 14, 0)

Citations

M. T. Goodrich, R. Tamassia. *Data Structures and Algorithms in Java, 6th edition.*

Wiley, 2014. Print.

Sedgewick, Wayne. *Algorithms (4th edition)*. Boston: Addison-Wesley

Professional, 2011. Print.