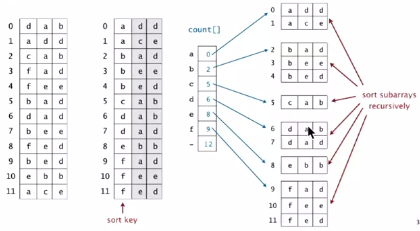
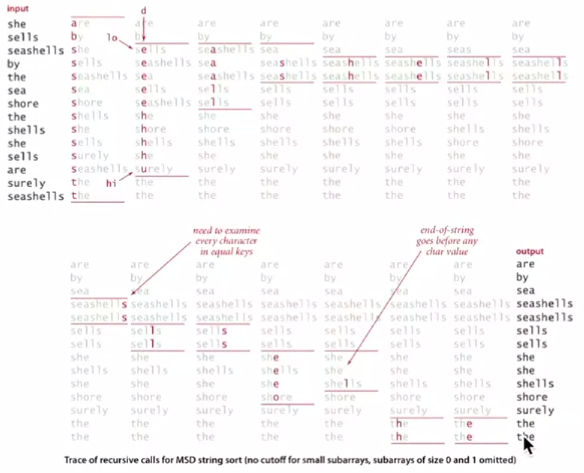
MSD (most –significant-digit-first) string sort

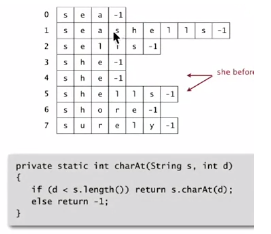
* Partition array into R pieces according to first character (use key-indexed counting)
* Recursively sort all strings that start with each character (key-indexed counts delineate subarrays to sort)



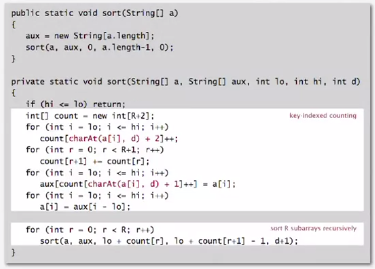
MSD Recursion trace



If strings are variable length, treat them as if they have an extra char at the end (smaller than any char)



MSD Sort implementation in Java

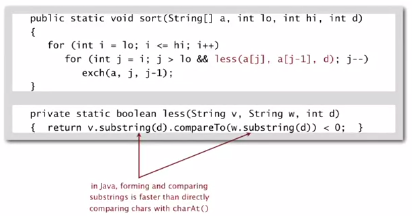


Problems:

* Much too small 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
* Huge number of small subarrays due to recursion

Solution: Cutoff to insertion sort for small subarrays

* Insertion sort, but start at dth character
* Implement less() so that it compares starting at dth character



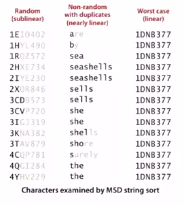
MSD string sort: performance

Main characteristic is that MSD only examines the keys it needs to examine

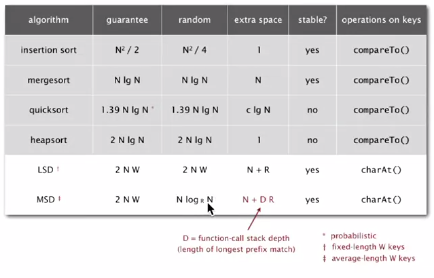
* MSD examines just enough characters to sort the keys
* Number of characters examined depends on the keys
* Can be sublinear in input size!  
   *compareTo() based on sorts can also be sublinear!*

Performance

1. Worst case time is when it examines all characters in all strings (e.g. when all equal).
2. Non-random with duplicates is nearly linear
3. Random is sublinear



Operation performance



MSD string sort vs quicksort

Disadvantages of MSD string sort

* Accesses memory randomly (cache inefficient)
* Inner loop has a lot of instructions
* Extra space for count[]
* Extra space for aux[]

Disadvantages of quicksort:

* Linearithmic number of string compares (not linear)
* Has to rescan many characters in keys with long prefix matches