Time complexity of Sorting

- Several sorting algorithms have been discussed and the best ones, so far:
  - Heap sort and Merge sort: O(n log n)
  - Quick sort (best one in practice): O(n log n) on average, O(n2) worst case

- 4319, 4628, 9329, 8618, 9615, 4595,
- 35, 129, 328, 115

Time complexity of Sorting

- Can we do better than O(n log n)?
  - No.
  - It can be proven that any comparison-based sorting algorithm will need to carry out at least O(n log n) operations

Can we do better?

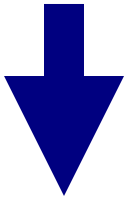
- Linear sorting algorithms
  - Bucket sort
  - Radix Sort
  - Counting Sort
- Make certain assumptions about the data
- Linear sorts are NOT “comparison sorts”

BinSort(not binary)
(a.k.a. BucketSort)

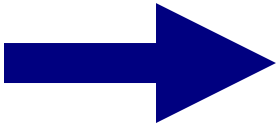
- If all keys are 1…K
- Have array of size K
- Put keys into correct bin (cell) of array

BinSort example

- K=5. list=(5,1,3,4,3,2,1,1,5,4,5)



Bins in array	
key = 1	1,1,1
key = 2	2
key = 3	3,3
key = 4	4,4
key = 5	5,5,5



Sorted list:
1,1,1,2,3,3,4,4,5,5,5

BinSort Pseudocode

```
procedure BinSort (List L,K)

LinkedList bins[1..K]
{ Each element of array bins is linked list.
Could also do a BinSort with array of arrays. }

For Each number x in L
      bins[x].Append(x)
End For
For i = 1..K
      For Each number x in bins[i]
            Print x
      End For
End For
```

BinSort Conclusion:

- K is a constant
  - BinSort is linear time
- K is large (e.g. 2^{32})
  - Impractical

BinSort is “stable”

- Stable Sorting algorithm.
  - Items in input with the same key end up in the same order as when they began.
  - Important if keys have associated values
  - Critical for RadixSort

Time complexity

- Bucket initialization: O(m)
- From array to buckets: O(n)
- From buckets to array: O(n)
  - Even though this stage is a nested loop, notice that all we do is dequeue from each bucket until they are all empty –> n dequeue operations in all

Time complexity

- Since m will likely be small compared to n, Bucket sort is O(n)
- Strictly speaking, time complexity is O (n + m)

Sorting integers

- Can we perform bucket sort on any array of (non-negative) integers?
  - Yes, but note that the number of buckets will depend on the maximum integer value
- If you are sorting 1000 integers and the maximum value is 999999, you will need 1 million buckets!
  - Time complexity is not really O(n) because m is much > than n. Actual time complexity is O(m)
- Can we do better?

Radixes

- The word "radix" is used as a synonym for "base".
- A radix-R representation is the same as a base-R representation.
- For example:
  - What is a radix-2 representation?
  - What is a radix-10 representation?
  - What is a radix-16 representation?

Radixes

- The word "radix" is used as a synonym for "base".
- A radix-R representation is the same as a base-R representation.
- For example:
  - What is a radix-2 representation? Binary.
  - What is a radix-10 representation? Decimal.
  - What is a radix-16 representation? Hexadecimal.
  - We often use radixes that are powers of 2, but not always.

Radix sort

- Two versions
  - MSD Radix sort – Most Significant Digit
  - LSD Radix sort – Least Siginificant Digit

MSD Radix Sort

- If the radix is R, the first pass of radix sort works as follows:
  - Create R buckets.
  - In bucket M, store all items whose most significant digit (in R-based representation) is M.
  - Reorder the array by concatenating the contents of all buckets.
- In the second pass, we sort each of the buckets separately.
  - All items in the same bucket have the same most significant digit.
  - Thus, we sort each bucket (by creating sub buckets of the bucket) based on the second most significant digit.
- We keep doing passes until we have used all digits.

Example

- Example: suppose our items are 3-letter words:
  - cat, dog, cab, ate, cow, dip, ago, cot, act, din, any.
- Let R = 26.
- This means that we will be creating 26 buckets at each pass.
- What would be the "digits" of the items, that we use to assign them to buckets?

Example

- Example: suppose our items are 3-letter words:
  - cat, dog, cab, ate, cow, dip, ago, cot, act, din, any.
- Let R = 26.
- This means that we will be creating 26 buckets at each pass.
- What would be the "digits" of the items, that we use to assign them to buckets?
- What will the buckets look like after the first pass?

Example

- Example: suppose our items are 3-letter words:
  - cat, dog, cab, ate, cow, dip, ago, cot, act, din, any.
- What will the buckets look like after the first pass?
- Bucket 'a' = ate, ago, act, any.
- Bucket 'c' = cat, cab, cow, cot.
- Bucket 'd' = dog, dip, din.
- **All other buckets are empty.**
- What happens at the second pass?

Example

- What happens at the second pass?
- Recursively do MSD Radix on each non empty bucket.
- Bucket 'a' = ate, ago, act, any.
  - subbucket 'c' = act.
  - subbucket 'g' = ago.
  - subbucket 'n' = any.
  - subbucket 't' = ate.
- Bucket 'a' is rearranged as act, ago, any, ate.
- Repeat same for bucket ‘c’ and bucket ‘d’

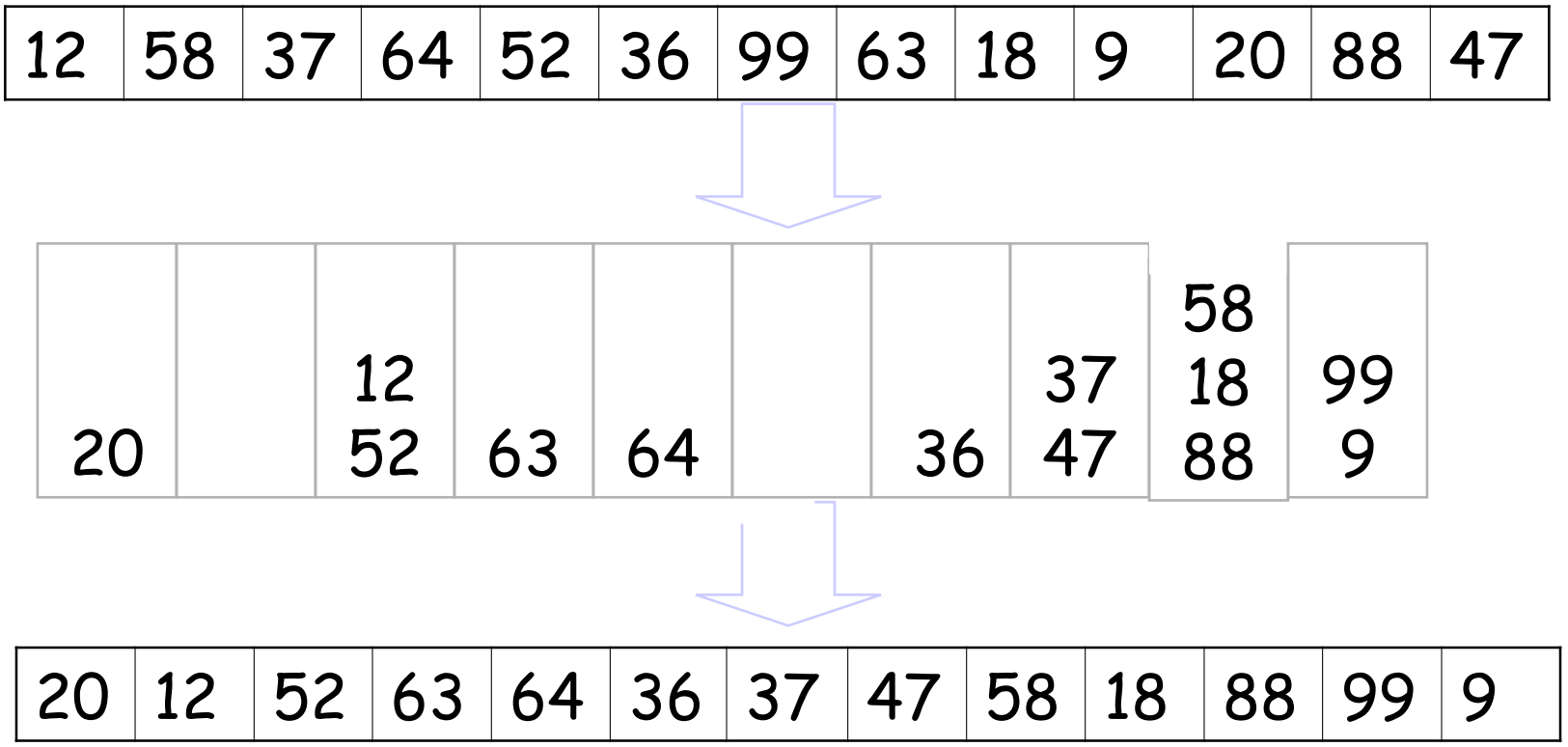
Programming MSD Radix Sort

- Sort numbers on most-significant-digit (MSD)
  - Initialise R bucket
  - For each MSD put the number In bucket
  - sort each of the resulting bins recursively
  - then, combine the bins in order

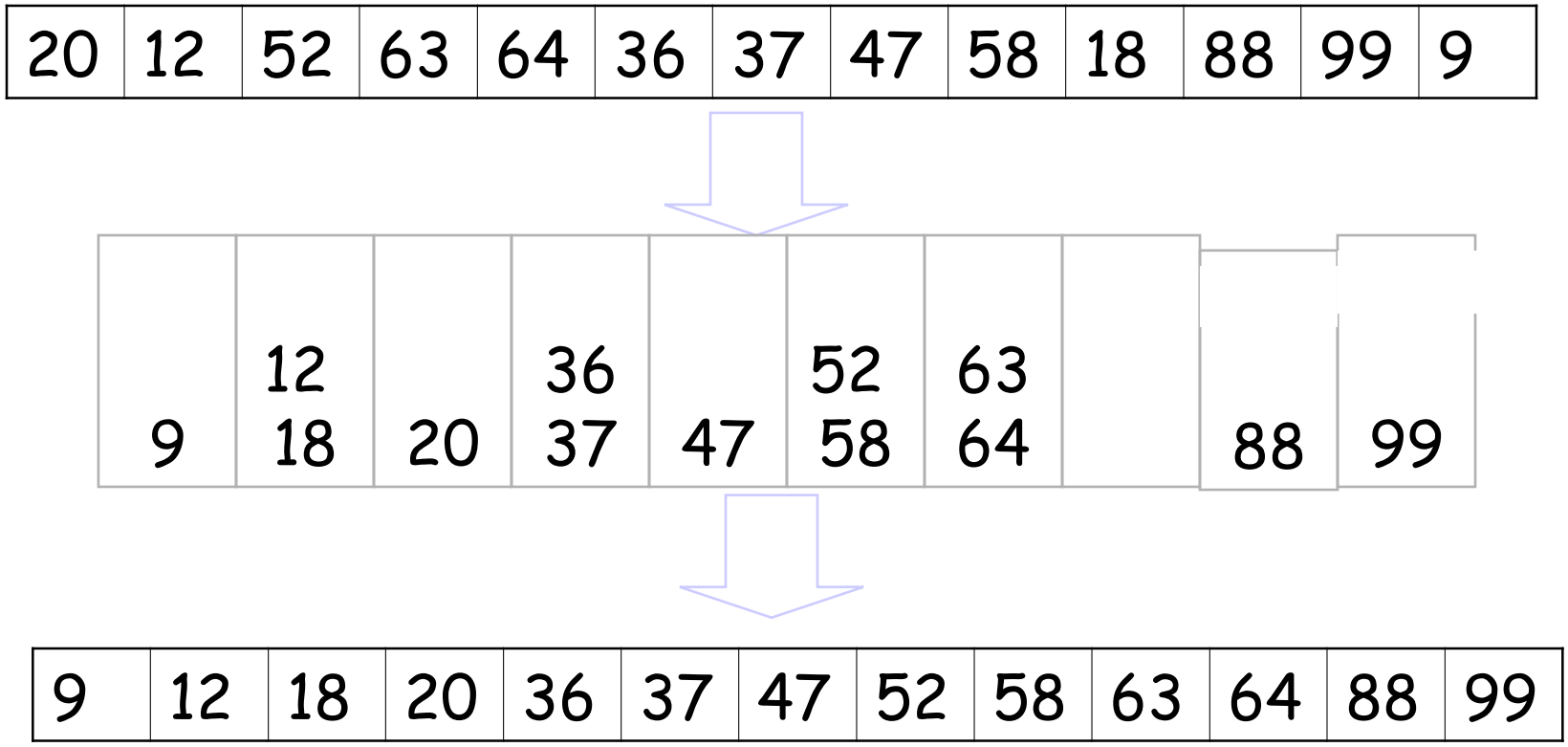
Radix sort - LSD

- Idea: repeatedly sort by digit—perform multiple bucket sorts on S starting with the rightmost digit
- If maximum value is 999999, only ten buckets (not 1 million) will be necessary
- Use this strategy when the keys are integers, and there is a reasonable limit on their values
  - Number of passes (bucket sort stages) will depend on the number of digits in the maximum value

Example: 1st pass



Example: 2nd pass



Example: 1st and 2nd passes

12	58	37	64	52	36	99	63	18	9	20	88	47

sort by rightmost digit

20	12	52	63	64	36	37	47	58	18	88	9	99

sort by leftmost digit

9	12	18	20	36	37	47	52	58	63	64	88	99

Radix sort and stability

- Radix sort works as long as the bucket sort stages are stable sorts

Radix sort and stability

- Stable sort: in case of ties, relative order of elements are preserved in the resulting array
  - Suppose there are two elements whose first digit is the same; for example, 52 & 58
  - If 52 occurs before 58 in the array prior to the sorting stage, 52 should occur before 58 in the resulting array
- This way, the work carried out in the previous bucket sort stages is preserved

Algorithm

```
Algorithm radixSort(a, first, last, maxDigits)
// Sorts the array of positive decimal integers a[first..last] into ascending order;
// maxDigits is the number of digits in the longest integer.

for (i = 0 to maxDigits - 1)
{
    Clear bucket[0], bucket[1], . . . , bucket[9]
    for (index = first to last)
    {
        digit = digit i of a[index]
        Place a[index] at end of bucket[digit]
    }
    Place contents of bucket[0], bucket[1], . . . , bucket[9] into the array a
}
```

Time complexity

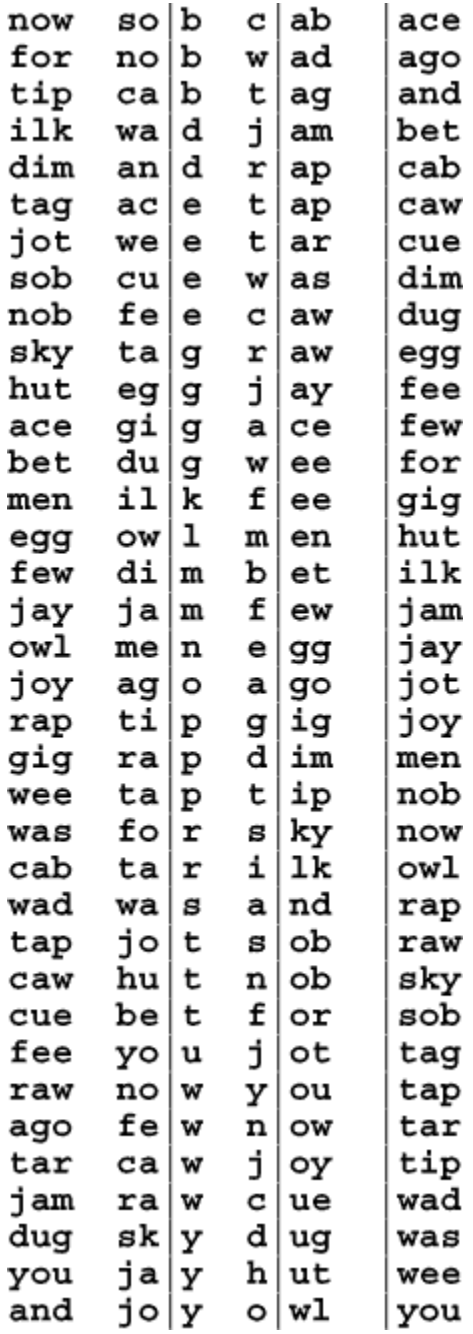
- If there is a fixed number p of bucket sort stages (six stages in the case where the maximum value is 999999), then radix sort is O(n)
  - There are p bucket sort stages, each taking O(n) time
- Strictly speaking, time complexity is O(pn), where p is the number of digits

About Radix sort

- Note that only 10 buckets are needed regardless of number of stages since the buckets are reused at each stage
- Radix sort can apply to words
  - Set a limit to the number of letters in a word
  - Use 27 buckets (or more, depending on the letters/characters allowed), one for each letter plus a “blank” character
  - The word-length limit is exactly the number of bucket sort stages needed

LSD Radix Sort

- Like LSD Binary sort –
  but on bigger chunks of the key.
- We still need to count the number of each type to figure out where to move items in one pass
- Runtime is still O(n).



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

Induction to the rescue!!!

- base case:
  - i=1. 0 digits are sorted (that wasn't hard!)

Induction

- Induction step
  - assume for i, prove for $i+1$.
  - consider two numbers: X, Y. Say X_i is i^{th} digit of X (from the right)
    - $X_{i+1} < Y_{i+1}$ then $i+1^{th}$ BinSort will put them in order
    - $X_{i+1} > Y_{i+1}$, same thing
    - $X_{i+1} = Y_{i+1}$, order depends on last i digits. Induction hypothesis says already sorted for these digits. (Careful about ensuring that your BinSort preserves order aka "stable"…)

RadixSorting Strings

- 1 Character can be BinSorted
- Break strings into characters
- Need to know length of biggest string (or calculate this on the fly).

Sort using Radix LSD

- FAT, FIT, SIT, CAT, BAT, MET, COT, DOT, BIT