Count Sort, Bucket Sort, Radix Sort

CSE 2320 – Algorithms and Data Structures
Vassilis Athitsos
University of Texas at Arlington

Overview

- Lower bounds on comparison-based sorting

- Count sort
- Bucket sort - briefly
- Radix sort

- The above sorting methods do not use comparisons
  - Exception: bucket sort when managing the buckets.

Lower-bounds on comparison-based sorting algorithms (Decision tree)

- Build the decision tree for an array with 3 elements
  - Ignore how the algorithm works (data movement, loops,…)
  - Focus only on comparisons needed to find the permutation. (We can look at the unsorted array as a permutation of the sorted data).

    => each permutation must be a leaf and must be reachable

    Number of permutations for n elements: n!

    => tree will have at least n! leaves. => height ≥ lg (n!) =>

    height = $\Omega(n \lg n)$ (b.c. $\lg(n!) = \Theta(n \lg n)$)
  - The decision tree for any comparison-based algorithm will have the above properties => cannot take less than $\Theta(n \lg n)$ time.

Count Sort

Based on counting occurrences, not on comparisons. See animation.

Stable?

Adaptive?

Extra memory?

Runtime?

Example 2: Sort an array of 10 English letters.

How big is the **Counts** array?

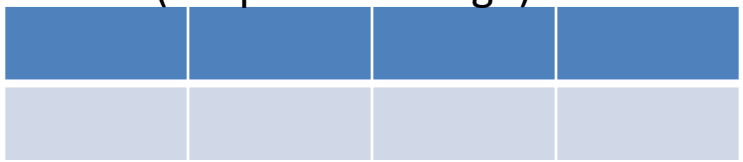
Θ(k)

(k = 26 possible key values letters)

Runtime: Θ(N+k)

D Rui	A Sam	C Mike	C Aaron	D Sam	C Tom	A Jane

Counts (=> position range)



Sorted data

0	1	2	3	4	5	6

Count Sort

Based on counting occurrences, not on comparisons. See animation.

Stable? Yes

Adaptive? No

Extra memory? Θ(N+k)

Runtime? Θ(N+k)

For sorting only grades (no names), just counting is enough.

Example 2: Sort an array of 10 English letters.

How big is the **Counts** array?

Θ(k)

(k = 26 possible key values letters)

Runtime: Θ(N+k)

D Rui	A Sam	C Mike	C Aaron	D Sam	C Tom	A Jane

1st count occurrences

A	B	C	D
2	0	3	2

2nd cumulative sum: gives 1+last index

A	B	C	D
2	~~0~~ 2 (=2+0)	~~3~~ 5 (=2+3)	~~2~~ 7 (=5+2)

Sorted data

0	1	2	3	4	5	6

Compare algorithms

- Compare the *time complexity* of Selection sort and Count sort for sorting
  - An array of 10 values in the range 1 to 10 vs
  - An array of 10 values in the range 501 to 1500.
  - An array of 1000 values in the range 1 to 10 vs
  - An array of 1000 values in the range 1 to 1000 vs

Algorithm/problem	**N = 10, k = ___ In range 1 to 10**	**N = 10, k = _____ In range 501 to 1500**	**N = 1000, k = __ In range 1 to 10**	**N = 1000, k = ____ In range 1 to 1000**
Selection sort	Θ(_____)	Θ(_____)	Θ(_____)	Θ(_____)
Count sort	Θ(_____)	Θ(_____)	Θ(_____)	Θ(_____)

Compare algorithms

- Compare the *time complexity* of Selection sort and Count sort for sorting
  - An array of 10 values in the range 1 to 10 vs
  - An array of 10 values in the range 501 to 1500.
  - An array of 1000 values in the range 1 to 10 vs
  - An array of 1000 values in the range 1 to 1000 vs

Algorithm/problem	**N = 10, k = 10 In range 1 to 10**	**N = 10, k = 1000 In range 501 to 1500**	**N = 1000, k = 10 In range 1 to 10**	**N = 1000, k = 1000 In range 1 to 10**
Selection sort	$\Theta(10^2)$	**$\Theta(10^2)$**	$\Theta(1000^2)$	$\Theta(1000^2)$
Count sort	**$\Theta(10+10)$ $=\Theta(10)$**	$\Theta(10+1000)$ $=\Theta(1000)$	**$\Theta(1000+10)$ $=\Theta(1000)$**	**$\Theta(1000+1000)$ $=\Theta(1000)$**

Best performing method is in **red**.
Note that this notation of Θ(number) is not correct.
I am showing it like this to highlight the difference in the values of N and k.

LSD Radix Sort

- Radix sort:
  - Addresses the problem count sort had with large range, k.
  - Sorts the data by repeatedly sorting by digits
  - Versions based on what it sorts first:
    - LSD = Least Significant Digit first.
    - MSD = Most Significant Digit first – We will not cover it.

- LSD radix sort
  - sorts the data based on individual digits, starting at the Least Significant Digit (LSD).
  - It is somewhat counterintuitive, but:
    - It works (requires a stable sort for sorting the digits)
    - It is simpler to implement than the MSD version.

LSD Radix sort

Algorithm:

for each digit i = 0 to d-1 *(0 is the least significant digit)*

count-sort A by digit I *(other STABLE sorting algs can be used)*

Example:

- Sort: {708, 512, 131, 24, 742, 810, 107, 634}

Using count-sort for the stable-sort by digit:

- Time complexity: _____
- Space complexity: _____

Bits and Radixes

- "radix" is used as a synonym for "base".
- A radix-k representation means a base-k representation.
- For example:
  - What is a radix-2 representation?
  - What is a radix-10 representation?
  - What is a radix-16 representation?

Bits and Radixes

- "radix" is used as a synonym for "base".
- A radix-k representation means a base-k 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.

LSD Radix Sort Complexity

- What are the quantities that affect the complexity?

- What is the time and space complexity?

LSD Radix Sort Complexity

- What are the quantities that affect the complexity?
  - n is the number of items
  - k is radix
  - d: the number of digits in the radix-k representation of each item.
- What is the time and space complexity?
- Θ(d*(k+n)) time. (Θ(nd+kd))
  - d * the time complexity of count sort
    - nd to put items into buckets, copy to scratch array and back
    - kd to initialize count and update the index where items from each bucket go.
  - See the visualization at: https://www.cs.usfca.edu/~galles/visualization/RadixSort.html
- Θ(n + k) space.
  - Θ(n) space for scratch array.
  - Θ(k) space for counters/indices array.

What base (radix) should you use?

- Do an example in base 2. Sort: {6,11,14,3,9} in base 2.
- Sort N = 20, 16-bit positive integers, with radixes:
  - 10, 2, 8, 16, 20
  - Compute the correct time complexity in each case $\Theta(d(N+k))$.
  - To compare the time complexity of the above methods, use the dominant term for (N+k)).

Radix sort Practice

Fill in the table below for count sort N = 20, 16-bit (b=16) positive integers using radixes (bases):10, 2, 8, 16, 32, 64. Assume all 16 bits are used (e.g. unsigned type).

Base $=2^r$	**N**	**k**	**d**	**Θ(d(N+k))**	**Dominant term**
10					
2					
8					
16					
32					
64					Expect larger or smaller than above?

Radix sort Practice

Fill in the table below for count sort N = 20, 16-bit (b=16) positive integers using radixes (bases):10, 2, 8, 16, 32, 64. Assume all 16 bits are used (e.g. unsigned type).

Base $=2^r$	N	k	d	Θ(d(N+k))	Dominant term
10	20	10	5	5(20+10)	100
2	20	2	16	16(20+2)	320
8 = 2^3	20	8	$\lceil 16/3 \rceil = 6$	6(20+8)	120
16 = 2^4	20	16	$\lceil 16/4 \rceil = 4$	4(20+16)	80
32 = 2^5	20	32	$\lceil 16/5 \rceil = 4$	4(20+32)	128
64 = 2^6					LARGER than 128

To compute d for bases that are powers of 2: $\mathrm{d} = \lceil b/r \rceil$

To compute d for base 10:

Largest positive int on 16 bits (uses all 16): 2^{16}-1 = 65536-1 = 65535 => d = 5

Tuning Radix Sort

Lemma 8.4 (CLRS): Given n b-bit numbers and any r ≤ b, LSD Radix-sort will correctly sort them in Θ((b/r)(n+2^r)) if the stable sort it uses takes Θ(n+k) to run for inputs in the range 0 to k.

(Here the radix (or base) is 2^r and each digit is represented on r bits)

How to choose r to optimize runtime:

- *r = min{b, floor(lg n)}* *(intuition: compare k with n and use the log of the smaller one)*
  - *If b ≤ lg n => r = b*
  - *If b > lg n => r = floor(lg n)*
- ***Use as base min(2^u, 2^b), where 2^u is the largest power of 2 smaller than n ($2^u \le n \le 2^{u+1}$)***

What is the extra space needed for each case above?

Θ(n+2^r) (assuming it uses count sort as the stable sorting algorithm for each digit)

What type of data can be sorted with radix sort?

For each type of data below, say if it can be sorted with Radix sort and how you would do it.

- Integers
  - Positive _____
  - Negative _____
  - Mixed _____
- Real numbers _____
- Strings _____
  - (If sorted according to the strcmp function, where "Dog" comes before "cat", because capital letters come before lowercase letters).
  - Consider “catapult” compared with “air”

Bucket Sort

Assume you need to sort an array of numbers in range [0,1):

E.g.: A = {0.58, 0.71, 0.23, 0.5, 0.12, 0.85, 0.29, 0.3, 0.21, 0.75}

Can we use count sort or radix sort to sort this type of data?

Bucket Sort

- Array, A, has *n* numbers in range [0,1) .
  - If they are not in range [0,1), bring them to that range.
  - See animation: https://www.cs.usfca.edu/~galles/visualization/BucketSort.html
- Idea:
  - Make as many buckets as number of items
  - Place items in buckets
  - Sort each bucket
  - Copy from each bucket into the original array
- Algorithm:

Create array, B, of size n. Each element will be a list (bucket).

For each list in B:

initialize it to be empty

For each elem in A,

*add elem to list B[floor(elem*n)]*

For each list in B:

sort it with insertion sort

For each list in B:

concatenate it (or copy back into A in this order).

Destroy the list (if needed).

Can you use count sort for this data?
A = {0.58, 0.71, 0.23, 0.5}

Some of the loops can be combined.
The given format makes the time complexity analysis easier.

Time complexity:
-Average: $\Theta(n)$
-Worst case : $\Theta(n^2)$
Worst case example:
.1,.11,.1001,.15,…

Bucket Sort Example

A has 10 elements:

A = {0.58, 0.71, 0.23, 0.5, 0.12, 0.85, 0.29, 0.3, 0.21, 0.75}

Give both an example of the data and the time complexity for:

- Best case: A=[_____] O(_____) Explanation: _____
- Worst case: A=[_____] O(_____) Explanation: _____

Bucket Sort Example

A has 8 elements:

A = {0.58, 0.71, 0.23, 0.5, 0.12, 0.85, 0.29, 0.3}

How would you repeat Bucket sort? E.g. if you wanted to apply bucket sort to the numbers from only one specific bucket, how could you do that? (If it helps, you can copy them back in A)

Range Transformations (Math review)

- Draw and show the mappings of the interval edges.

- [0,1) -> [0,n)

$$y = xn$$

- [a,b) -> [0,1) -> [0,n)

$$y = \frac{x-a}{b-a} \qquad z = yn$$

- [a,b) -> [0,1) -> [s,t)

$$z = y(t-s)+s$$

if $[a,b)$ -> $[0,n)$:

$$z = \frac{x-a}{b-a+1}n$$

As a check, see that a-> 0 and b-> $y < n$.

Direct formula for $[a,b)$ -> $[s,t)$:

$$z = \frac{x-a}{b-a}(t-s)+s$$

As a check, see that a-> s and b-> t.

A searching problem

- Money winning game:
  - There is an array, A, with 100 items.
  - The items are values in range [1,1000].
  - A is sorted.
  - Values in A are hidden (you cannot see them).
  - You will be given a value, val, to search for in the array and need to either find it (uncover it) or report that it is not there.
  - You start with $5000. For a $500 charge, you can ask the game host to flip (uncover) an item of A at a specific index (chosen by you). You win whatever money you have left after you give the correct answer. You have one free flip.

Value, val, you are searching for.	**What index will you flip?** **(this is a TV show so indexes starts from 1, not 0)**	
524		
100		
10		

Index	1	2	…	99	100
A					

- Money winning game – Version 2 only specific indexes can be flipped.
  - There is an array, A, with 100 items.
  - The items are values in range [1,100].
  - A is sorted.
  - Values in A are hidden (you cannot see them).
  - You will be given a value, val, to search for in the array and need to either find it (uncover it) or report that it is not there.
  - You start with $5000. For a $500 charge, you can ask the game host to flip (uncover) an item of A at a specific index (chosen by you). You win whatever money you have left after you give the correct answer. You have one free flip.

Value, val, you are searching for.	What index will you flip? 1?, 10?, 25?,50?, 75?, 90?, 100?	
524		
100		
10		

Index	1	2	…	99	100
A					

Interpolated Binary Search

- Similar to binary search, but I want an 'educated guess'.

- E.g. given sorted array, A, of 100 numbers in range [0,1000], if you search in A for the value below, what index will you look at first?

  Assume it is a money-winning game and that for each trial/question, you loose some of the prize money. Which indexes would you pick?

  – 524 Look at index: 1?, 10?, 25?, 50?, 75?, 90?, 100?
  – 100 Look at index: 1?, 10?, 25?, 50?, 75?, 90?, 100?
  – 10 Look at index: 1?, 10?, 25?, 50?, 75?, 90?, 100?

Direct formula for *[a,b)* -> *[s,t)*:

$$z = \frac{x-a}{b-a}(t-s)+s = (x-a)\frac{t-s}{b-a}+s$$

As a check, see that $a \to s$ and $b \to t$.

Here: value range [Mn, Mx], index range [s, t].

Use the [Mn, Mx] -> [s, t] transformation and use for x the value you are searching for.

The result, z, is the index, i, you are looking for.

Next level …

- Let’s assume you can play the actual game.
- Can you write a program to play this game instead of you?
  - What will be the program inputs?
  - Give an algorithm for it.
  - Check that the algorithm has the same behavior as the human (do you flip the same indexes?) for specific examples.
  - Check border cases
    - What border cases would you check for?
      - Value not in A / indexes cross over
      - Value at the edge (first or last in array)
    - Can you construct an example for each one of them?

EXTRA

- Example showing Radix-sort for base 2.

LSD Radix Sort Implementation

before radix sort:

0: 4

1: 93

2: 5

3: 104

4: 53

5: 90

6: 208

LSD Radix Sort Implementation

```
done with bit 0
 0:           4 00000000000000000000000000000100
 1:         104 00000000000000000000000001101000
 2:          90 00000000000000000000000001011010
 3:         208 00000000000000000000000011010000
 4:          93 00000000000000000000000001011101
 5:           5 00000000000000000000000000000101
 6:          53 00000000000000000000000000110101
```

LSD Radix Sort Implementation

```
done with bit 1
 0:           4 00000000000000000000000000000100
 1:         104 00000000000000000000000001101000
 2:         208 00000000000000000000000011010000
 3:          93 00000000000000000000000001011101
 4:           5 00000000000000000000000000000101
 5:          53 00000000000000000000000000110101
 6:          90 00000000000000000000000001011010
```

LSD Radix Sort Implementation

```
done with bit 2
 0:         104 00000000000000000000000001101000
 1:         208 00000000000000000000000011010000
 2:          90 00000000000000000000000001011010
 3:           4 00000000000000000000000000000100
 4:          93 00000000000000000000000001011101
 5:           5 00000000000000000000000000000101
 6:          53 00000000000000000000000000110101
```

LSD Radix Sort Implementation

```
done with bit 3
 0:         208 00000000000000000000000011010000
 1:           4 00000000000000000000000000000100
 2:           5 00000000000000000000000000000101
 3:          53 00000000000000000000000000110101
 4:         104 00000000000000000000000001101000
 5:          90 00000000000000000000000001011010
 6:          93 00000000000000000000000001011101
```

LSD Radix Sort Implementation

```
done with bit 4
 0:           4 00000000000000000000000000000100
 1:           5 00000000000000000000000000000101
 2:         104 00000000000000000000000001101000
 3:         208 00000000000000000000000011010000
 4:          53 00000000000000000000000000110101
 5:          90 00000000000000000000000001011010
 6:          93 00000000000000000000000001011101
```

LSD Radix Sort Implementation

```
done with bit 5
 0:           4 00000000000000000000000000000100
 1:           5 00000000000000000000000000000101
 2:         208 00000000000000000000000011010000
 3:          90 00000000000000000000000001011010
 4:          93 00000000000000000000000001011101
 5:         104 00000000000000000000000001101000
 6:          53 00000000000000000000000000110101
```

LSD Radix Sort Implementation

```
done with bit 6
 0:           4 00000000000000000000000000000100
 1:           5 00000000000000000000000000000101
 2:          53 00000000000000000000000000110101
 3:         208 00000000000000000000000011010000
 4:          90 00000000000000000000000001011010
 5:          93 00000000000000000000000001011101
 6:         104 00000000000000000000000001101000
```

LSD Radix Sort Implementation

```
done with bit 7
 0:           4 00000000000000000000000000000100
 1:           5 00000000000000000000000000000101
 2:          53 00000000000000000000000000110101
 3:          90 00000000000000000000000001011010
 4:          93 00000000000000000000000001011101
 5:         104 00000000000000000000000001101000
 6:         208 00000000000000000000000011010000
```

LSD Radix Sort Implementation

```
done with bit 8
 0:           4 00000000000000000000000000000100
 1:           5 00000000000000000000000000000101
 2:          53 00000000000000000000000000110101
 3:          90 00000000000000000000000001011010
 4:          93 00000000000000000000000001011101
 5:         104 00000000000000000000000001101000
 6:         208 00000000000000000000000011010000
```