

ECE2810J

Data Structures and Algorithms

Non-Comparison Sort

Learning Objective:

- Understand three non-comparison sorts, counting sort, bucket sort, and radix sort

Outline

- ▶ Non-comparison Sort
 - ▶ Counting Sort
 - ▶ Bucket Sort
 - ▶ Radix Sort

Counting Sort: A Simple Version

- ▶ Sort an array **A** of **integers** in the range $[0, k]$, where k is known.
 1. Allocate an array **count** $[k+1]$.
 2. Scan array **A**. For $i=1$ to N , increment **count** $[A[i]]$.
 3. Scan array **count**. For $i=0$ to k , print i for **count** $[i]$ times.
- ▶ Time complexity: $O(N + k)$.
- ▶ The algorithm can be converted to sort integers in some other known range $[a, b]$.
 - ▶ Minus each number by a , converting the range to $[0, b - a]$.

Example

2	5	3	0	2	3	0	3
---	---	---	---	---	---	---	---

Pseudo code:

Sort an array A of **integers** in the range $[0, k]$, where k is known.

1. Allocate an array `count[k+1]`.
2. Scan array A . For $i=1$ to N , increment `count[A[i]]`.
3. Scan array `count`. For $i=0$ to k , print i for `count[i]` times.

Counting Sort: A General Version

- ▶ In the previous version, we print i for $\text{count}[i]$ times
 - ▶ Simple but only works when sorting integer keys alone
 - ▶ How to sort items when there is “*additional*” information with each key? Furthermore, how to guarantee the stability?
- ▶ A general version:
 1. Allocate an array $C[k+1]$
 2. Scan array A . For $i=1$ to N , increment $C[A[i]]$
 3. For $i=1$ to k , $C[i]=C[i-1]+C[i]$
 - ▶ $C[i]$ now contains number of items less than or equal to i
 4. For $i=N$ downto 1 , put $A[i]$ in new position $C[A[i]]$ and decrement $C[A[i]]$

Counting Sort: Example

1. Allocate an array $C[k+1]$.
2. Scan array A . For $i=1$ to N , increment $C[A[i]]$.
3. For $i=1$ to k , $C[i]=C[i-1]+C[i]$
4. For $i=N$ downto 1 , put $A[i]$ in new position $C[A[i]]$ and decrement $C[A[i]]$.

$k=5$

	1	2	3	4	5	6	7	8
A	2	5	3	0	2	3	0	3

	0	1	2	3	4	5
C	2	0	2	3	0	1

	0	1	2	3	4	5
C	2	2	4	7	7	8

Counting Sort Example

1. Allocate an array $C[k+1]$.
2. Scan array A . For $i=1$ to N , increment $C[A[i]]$.
3. For $i=1$ to k , $C[i] = C[i-1] + C[i]$
4. For $i=N$ downto 1, put $A[i]$ in new position $C[A[i]]$ and decrement $C[A[i]]$.

$k=5$

	1	2	3	4	5	6	7	8
A	2	5	3	0	2	3	0	3

	0	1	2	3	4	5
C	2	2	4	7	7	8

	1	2	3	4	5	6	7	8
							3	

	0	1	2	3	4	5
C	2	2	4	6	7	8

Why putting 3 at location 7 is correct?

Counting Sort Example

1. Allocate an array $C[k+1]$.
2. Scan array A . For $i=1$ to N , increment $C[A[i]]$.
3. For $i=1$ to k , $C[i] = C[i-1] + C[i]$
4. For $i=N$ downto 1 , put $A[i]$ in new position $C[A[i]]$ and decrement $C[A[i]]$.

$k=5$

	1	2	3	4	5	6	7	8
A	2	5	3	0	2	3	0	3

	0	1	2	3	4	5
C	2	2	4	6	7	8

	1	2	3	4	5	6	7	8
		0					3	

	0	1	2	3	4	5
C	1	2	4	6	7	8

Counting Sort Example

1. Allocate an array $C[k+1]$.
2. Scan array A . For $i=1$ to N , increment $C[A[i]]$.
3. For $i=1$ to k , $C[i] = C[i-1] + C[i]$
4. For $i=N$ downto 1, put $A[i]$ in new position $C[A[i]]$ and decrement $C[A[i]]$.

$k=5$

	1	2	3	4	5	6	7	8
A	2	5	3	0	2	3	0	3

	0	1	2	3	4	5
C	1	2	4	6	7	8

	1	2	3	4	5	6	7	8
		0				3	3	

	0	1	2	3	4	5
C	1	2	4	5	7	8

Counting Sort Example

1. Allocate an array $C[k+1]$.
2. Scan array A . For $i=1$ to N , increment $C[A[i]]$.
3. For $i=1$ to k , $C[i] = C[i-1] + C[i]$
4. For $i=N$ downto 1, put $A[i]$ in new position $C[A[i]]$ and decrement $C[A[i]]$.

$k=5$

	1	2	3	4	5	6	7	8
A	2	5	3	0	2	3	0	3

	0	1	2	3	4	5
C	1	2	4	5	7	8

	1	2	3	4	5	6	7	8
		0		2		3	3	

	0	1	2	3	4	5
C	1	2	3	5	7	8

Counting Sort Example

1. Allocate an array $C[k+1]$.
2. Scan array A . For $i=1$ to N , increment $C[A[i]]$.
3. For $i=1$ to k , $C[i] = C[i-1] + C[i]$
4. For $i=N$ downto 1 , put $A[i]$ in new position $C[A[i]]$ and decrement $C[A[i]]$.

$k=5$

	1	2	3	4	5	6	7	8
A	2	5	3	0	2	3	0	3

	0	1	2	3	4	5
C	1	2	3	5	7	8

	1	2	3	4	5	6	7	8
	0	0		2		3	3	

	0	1	2	3	4	5
C	0	2	3	5	7	8

Counting Sort Example

1. Allocate an array $C[k+1]$.
2. Scan array A . For $i=1$ to N , increment $C[A[i]]$.
3. For $i=1$ to k , $C[i] = C[i-1] + C[i]$
4. For $i=N$ downto 1, put $A[i]$ in new position $C[A[i]]$ and decrement $C[A[i]]$.

$k=5$

	1	2	3	4	5	6	7	8
A	2	5	3	0	2	3	0	3

	0	1	2	3	4	5
C	0	2	3	5	7	8

1	2	3	4	5	6	7	8
0	0		2	3	3	3	

	0	1	2	3	4	5
C	0	2	3	4	7	8

Counting Sort Example

1. Allocate an array $C[k+1]$.
2. Scan array A . For $i=1$ to N , increment $C[A[i]]$.
3. For $i=1$ to k , $C[i] = C[i-1] + C[i]$
4. For $i=N$ downto 1 , put $A[i]$ in new position $C[A[i]]$ and decrement $C[A[i]]$.

$k=5$

	1	2	3	4	5	6	7	8
A	2	5	3	0	2	3	0	3

	0	1	2	3	4	5
C	0	2	3	4	7	8

	1	2	3	4	5	6	7	8
	0	0		2	3	3	3	5

	0	1	2	3	4	5
C	0	2	3	4	7	7

Counting Sort Example

1. Allocate an array $C[k+1]$.
2. Scan array A . For $i=1$ to N , increment $C[A[i]]$.
3. For $i=1$ to k , $C[i] = C[i-1] + C[i]$
4. For $i=N$ downto 1, put $A[i]$ in new position $C[A[i]]$ and decrement $C[A[i]]$.

$k=5$

	1	2	3	4	5	6	7	8
A	2	5	3	0	2	3	0	3

	0	1	2	3	4	5
C	0	2	3	4	7	7

	1	2	3	4	5	6	7	8
	0	0	2	2	3	3	3	5

Done!

	0	1	2	3	4	5
C	0	2	2	4	7	7

Is counting sort stable? Yes!

Code Exercise: Counting Sort (~10 mins)

- Goal: Implement your own Counting sort
- Code can be found in:
 - Canvas -> Non Comparison Sort -> Code Exercise -> Counting Sort

Outline

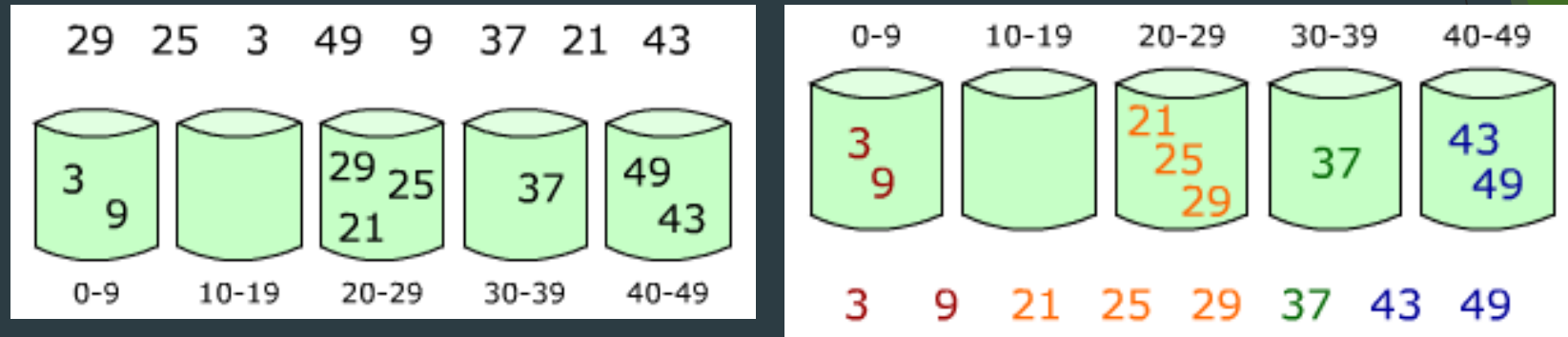
- ▶ Non-comparison Sort
 - ▶ Counting Sort
 - ▶ Bucket Sort
 - ▶ Radix Sort

Bucket Sort

- ▶ Instead of simple integer, each key can be a complicated record, such as a real value.
- ▶ Then instead of incrementing the count of each bucket, **distribute** the records **by their keys** into appropriate buckets.
- ▶ Algorithm:
 1. Set up an array of initially empty “buckets”.
 2. Scatter: Go over the original array, putting each object in its bucket.
 3. Sort each non-empty bucket by a comparison sort.
 4. Gather: Visit the buckets in order and put all elements back into the original array.

Bucket Sort

► Example



► Time complexity

- Suppose we are sorting cN items and we divide the entire range into N buckets.
- Assume that the items are **uniformly distributed** in the entire range.
- The average case time complexity is $O(N)$.

Code Exercise: Bucket Sort (~10 mins)

- Goal: Implement your own Bucket sort
- Code can be found in:
 - Canvas -> Non Comparison Sort -> Code Exercise -> Bucket Sort

Outline

- ▶ Non-comparison Sort
 - ▶ Counting Sort
 - ▶ Bucket Sort
 - ▶ Radix Sort

Radix Sort

- ▶ **Radix sort** sorts integers by looking at one digit at a time.
- ▶ Procedure: Given an array of integers, from the least significant bit (LSB) to the most significant bit (MSB), repeatedly do **stable** bucket sort according to the current bit.
- ▶ For sorting base- b numbers, bucket sort needs b buckets.
 - ▶ For example, for sorting decimal numbers, bucket sort needs 10 buckets.

Radix Sort

Example

- ▶ Sort 815, 906, 127, 913, 098, 632, 278.
- ▶ Bucket sort 815, 906, 127, 913, 098, 632, 278 according to the least significant bit:

0	1	2	3	4	5	6	7	8	9
		63 <u>2</u>	91 <u>3</u>		81 <u>5</u>	90 <u>6</u>	12 <u>7</u>	09 <u>8</u> 27 <u>8</u>	

- ▶ Bucket sort 632, 913, 815, 906, 127, 098, 278 according to the second bit.

Radix Sort

Example

- ▶ Bucket sort 632, 913, 815, 906, 127, 098, 278 according to the second bit.

0	1	2	3	4	5	6	7	8	9
9 <u>0</u> 6	9 <u>1</u> 3 8 <u>1</u> 5	1 <u>2</u> 7	6 <u>3</u> 2				2 <u>7</u> 8		0 <u>9</u> 8

- ▶ Bucket sort 906, 913, 815, 127, 632, 278, 098 according to the most significant bit.

Radix Sort

Example

- ▶ Bucket sort 906, 913, 815, 127, 632, 278, 098 according to the most significant bit.

0	1	2	3	4	5	6	7	8	9
<u>0</u> 98	<u>1</u> 27	<u>2</u> 78				<u>6</u> 32		<u>8</u> 15	<u>9</u> 06 <u>9</u> 13

- ▶ The final sorted order is: 098, 127, 278, 632, 815, 906, 913.

Radix Sort: Correctness

- ▶ Claim: after bucket sorting the i -th LSB, the numbers are sorted according to their last i digits
- ▶ Proof by mathematical induction
- ▶ Base case is obviously true
- ▶ Inductive step
 - ▶ Assume that according to the last i digits, order is $a_1 < \dots < a_n$
 - ▶ For two adjacent numbers a_k and a_{k+1} if they are not in the same bucket, they are sorted according to their last i^{th} digits
 - ▶ If they are in the same bucket, then $a_k < a_{k+1}$ for the last $(i - 1)$ bits. They are also sorted due to stability of bucket sort

Radix Sort

Time Complexity

- ▶ Let k be the maximum number of digits in the keys and N be the number of keys.
- ▶ We need to repeat bucket sort k times.
 - ▶ Time complexity for the bucket sort is $O(N)$.
- ▶ The total time complexity is $O(kN)$.




Radix Sort


- ▶ Radix sort can be applied to sort keys that are built on **positional notation**.
 - ▶ **Positional notation**: all positions uses the same set of symbols, but different positions have different weight.
 - ▶ Decimal representation and binary representation are examples of positional notation.
 - ▶ Strings can also be viewed as a type of positional notation. Thus, radix sort can be used to sort strings.
- ▶ We can also apply radix sort to sort records that contain multiple keys.
 - ▶ For example, sort records (year, month, day).



Code Exercise: Radix Sort (~10 mins)

- Goal: Implement your own Radix sort
- Code can be found in:
 - Canvas -> Non Comparison Sort -> Code Exercise -> Radix Sort


Code Exercise: Sorting in A Yelp-like Android APP








1. Izzy's Brooklyn Bagels
 300 Reviews
477 S California Ave, Palo Alto
Bagels, Kosher
 Order Pickup


\$




2. House of Bagels
 5 Reviews
2190 W Bayshore Rd, Palo Alto
Bagels
 Hot and New


\$




3. New York New York Sandwiches
 55 Reviews
125 University Ave, Palo Alto
Breakfast & Brunch, Sandwiches, Burgers


\$






4. Philz Coffee
 889 Reviews
3191 Middlefield Rd, Palo Alto
Coffee & Tea


\$\$





5. House of Bagels
 107 Reviews
1712 Miramonte Ave, Mountain View
Bagels, Breakfast & Brunch, Sandwiches


\$


**Izzy's Brooklyn Bagels**

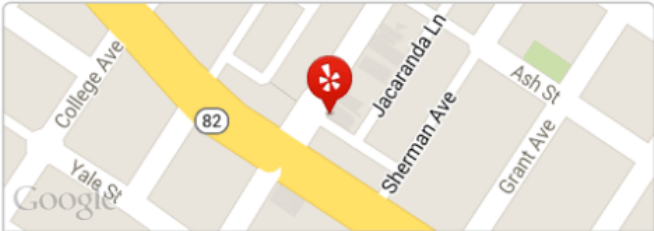
Izzy's Brooklyn Bagels 28.6 mi
 259 Reviews
Bagels, Kosher
Hours Today: 6:00 AM - 4:00 PM **Closed**

 Add Review


 Add Photo


 Check In

 Bookmark




477 S California Ave, Palo Alto, CA 94306

 Get Directions

 Call (650) 329-0700

More Info
Menu, Hours, Website, Attributes...

Code Exercise: LeetCode Problem 164

 Problem List < > 🔍

Description Editorial Solutions Submissions

164. Maximum Gap

Medium Topics Companies

Given an integer array `nums`, return the maximum difference between two successive elements in its sorted form. If the array contains less than two elements, return `0`.

You must write an algorithm that runs in linear time and uses linear extra space.

Example 1:

Input: `nums = [3,6,9,1]`
Output: `3`
Explanation: The sorted form of the array is `[1,3,6,9]`, either `(3,6)` or `(6,9)` has the maximum difference `3`.

Example 2:

Input: `nums = [10]`
Output: `0`
Explanation: The array contains less than 2 elements, therefore return `0`.

Constraints:


- `1 <= nums.length <= 105`
- `0 <= nums[i] <= 109`

That is All for today!


Any questions?





Today's 281 One More Thing


TXYZ

Beta

 Daily Digest

 My Documents

 Dark Mode

 Log In


[Terms](#) [Privacy](#)

Personalized paper feed tailored to your interests. [Artificial Intelligence](#) [+ Add Subjects](#)

Press **(fn) F** to exit full screen


Add keywords to filter...

Using AI Uncertainty Quantification to Improve Human Decision-Making

 Laura R. Marusich, Jonathan Z. Bakdash, Yan Zhou, Murat Kantarcioglu


AI Uncertainty Quantification (UQ) has the potential to improve human decision-making beyond AI predictions alone by providing additional useful probabilistic information to users. The majority of past research on AI and human decision-making has concentrated on model explainability and interpretability. We implemented instance-based UQ for three real datasets. To achieve this, we trained different AI models for classification for each dataset, and used random samples generated around the neighborhood of the given instance to create confidence intervals for UQ. The computed UQ was calibrated using a strictly proper scoring rule as a form of quality assurance for UQ. We then conducted two preregistered online behavioral experiments that compared objective human decision-making performance under different AI information conditions, including UQ. In Experiment 1, we compared decision-making for no AI (control), AI prediction alone, and AI prediction with a visualization of UQ. We found UQ significantly improved decision-making beyond the other two conditions. In Experiment 2, we focused on comparing different representations of UQ information: Point vs. distribution of uncertainty and visualization type (needle vs. dotplot). We did not find meaningful differences in decision-making performance among these different representations of UQ. Overall, our results indicate that human decision-making can be improved by providing UQ information along with AI predictions, and that this benefit generalizes across a variety of representations of UQ.

Believable Minecraft Settlements by Means of Decentralised Iterative Planning

 Arthur van der Staaij, Jelmer Prins, Vincent L. Prins, Julian Poelsma, Thera Smit, Matthias Müller-Brockhausen, Mike Preuss

Procedural city generation that focuses on believability and adaptability to random terrain is a difficult challenge in the field of Procedural Content Generation (PCG). Dozens of researchers compete for a realistic approach in challenges such as the Generative Settlement Design in Minecraft (GDMC), in which our method has won the 2022 competition. This was achieved through a decentralised, iterative planning process that is transferable to similar generation processes that aims to produce "organic" content procedurally.

Classifying Organizations for Food System Ontologies using Natural Language Processing

 Tianyu Jiang, Sonia Vinogradova, Nathan Stringham, E. Louise Earl, Allan D. Hollander, Patrick R. Huber, Ellen Riloff, R. Sandra Schillo, Giorgio A. Ubbiali, Matthew Lange

Our research explores the use of natural language processing (NLP) methods to automatically classify entities for the purpose of knowledge graph population and integration with food system ontologies. We have created NLP models that can automatically classify organizations with respect to categories associated with environmental issues as well as Standard Industrial Classification (SIC) codes, which are used by the U.S. government to characterize business activities. As input, the NLP models are provided with text snippets retrieved by the Google search engine for each organization, which covers a textual description of the organization that is used for indexing. Our experimental

Contact Us

Txyz.ai