Hash Table

CS143: lecture 16

Sorting

Recap

- Three $O(n^2)$ algorithms: Selection, Insertion, Bubble
- Four $O(n \log n)$ algorithms: Tree, Merge, Quick, Heap
- There are a lot more sorting algorithms...

Sorting Recap

- Three $O(n^2)$ algorithms: Selection
- Four $O(n \log n)$ algorithms: T
- There are a lot more sorting a

Name +	Best +	Average +	Worst +	
Quicksort	$n \log n$	$n \log n$	n^2	
Merge sort	$n \log n$	$n \log n$	$n \log n$	
In-place merge sort	_	_	$n\log^2 n$	
Introsort	$n \log n$	$n \log n$	$n \log n$	
Heapsort	$n \log n$	$n \log n$	$n \log n$	
Insertion sort	n	n^2	n^2)la la l
Block sort	n	$n \log n$	$n \log n$	Subble
Timsort	n	$n \log n$	$n \log n$, Hoor
Selection sort	n^2	n^2	n^2	к, неар
Cubesort	n	$n \log n$	$n \log n$	
Shellsort	$n \log n$	$n^{4/3}$	$n^{3/2}$	
Bubble sort	n	n^2	n^2	
Exchange sort	n^2	n^2	n^2	
Tree sort	$n \log n$	$n \log n$	$n\log n$ (balanced)	
Cycle sort	n^2	n^2	n^2	
Library sort	$n \log n$	$n \log n$	n^2	
Patience sorting	n	$n \log n$	$n \log n$	
Smoothsort	n	$n \log n$	$n \log n$	
Strand sort	n	n^2	n^2	
Tournament sort	$n \log n$	$n \log n$	$n \log n$	
Cocktail shaker		2	2	

Article Talk Read Edit View history Tools >

From Wikipedia, the free encyclopedia

In computer science, bogosort^{[1][2]} (also known as permutation sort and stupid sort^[3]) is a sorting algorithm based on the generate and test paradigm. The function successively generates permutations of its input until it finds one that is sorted. It is not considered useful for sorting, but may be used for educational purposes, to contrast it with more efficient algorithms.

Two versions of this algorithm exist: a deterministic version that enumerates all permutations until it hits a sorted one, [2][4] and a randomized version that randomly permutes its input. An analogy for the working of the latter version is to sort a deck of cards by throwing the deck into the air, picking the cards up at random, and repeating the process until the deck is sorted. In a worst-case scenario with this version, the random source is of low quality and happens to make the sorted permutation unboundedly unlikely to occur. The algorithm's name is a portmanteau of the words *bogus* and *sort*.^[5]

Bogosort

Class Sorting Data Array structure Unbounded (randomized **Worst-case** version), $O(n \times n!)$ performance (deterministic version) $\Omega(n)^{[1]}$ **Best-case** performance $\Theta(n \times n!)^{[1]}$ **Average** performance O(1)**Worst-case** space complexity

Description of the algorithm [edit]

Pseudocode [edit]

The following is a description of the randomized algorithm in pseudocode:

```
while not sorted(deck):
shuffle(deck)
```

Sorting

Recap

- Three $O(n^2)$ algorithms: Selection, Insertion, Bubble
- Four $O(n \log n)$ algorithms: Tree, Merge, Quick, Heap
- There are a lot more sorting algorithms...
- ... we have time for one more weird one

- Count the occurrences of every number
- Output each number as many times as it occurs in the original list

Input

4	8	4	2	9	9	6	2	9
		l			I	I	1	1

0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0

Input

4 8 4 2 9 9 6 2

0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	1	0	0	0	0	0	0

Input

4	8	4	2	9	9	6	2	9
---	---	---	---	---	---	---	---	---

0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	1	0	0	0	1	0	0

Input

4	8	4	2	9	9	6	2	9
							1	ı

0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	2	0	0	0	1	0	0

Input

4	8	4	2	9	9	6	2	9
---	---	---	---	---	---	---	---	---

0	1	2	3	4	5	6	7	8	9	10
0	0	1	0	2	0	0	0	1	0	0

Input

4	8	4	2	9	9	6	2	9
---	---	---	---	---	---	---	---	---

0	1	2	3	4	5	6	7	8	9	10
0	0	1	0	2	0	0	0	1	1	0

Input

4 8 4 2 9 9 6 2

L											10
	0	0	1	0	2	0	0	0	1	2	0

Input

4	8	4	2	9	9	6	2	9
---	---	---	---	---	---	---	---	---

0	1	2	3	4	5	6	7	8	9	10
0	0	1	0	2	0	1	0	1	2	0

Input

4	8	4	2	9	9	6	2	9
---	---	---	---	---	---	---	---	---

0	1	2	3	4	5	6	7	8	9	10
0	0	2	0	2	0	1	0	1	2	0

Input

4 8 4 2 9 9 6 2 9

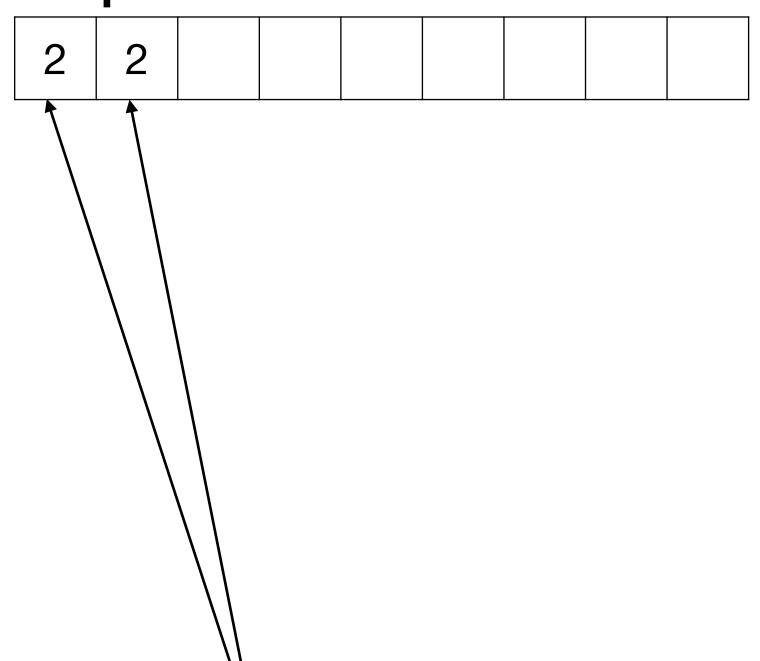
0	1	2	3	4	5	6	7	8	9	10	
0	0	2	0	2	0	1	0	1	3	0	

Output

1 1 1 1 1 1 1 1 1 1					
1 1 1 1 1 1 1 1 1 1					

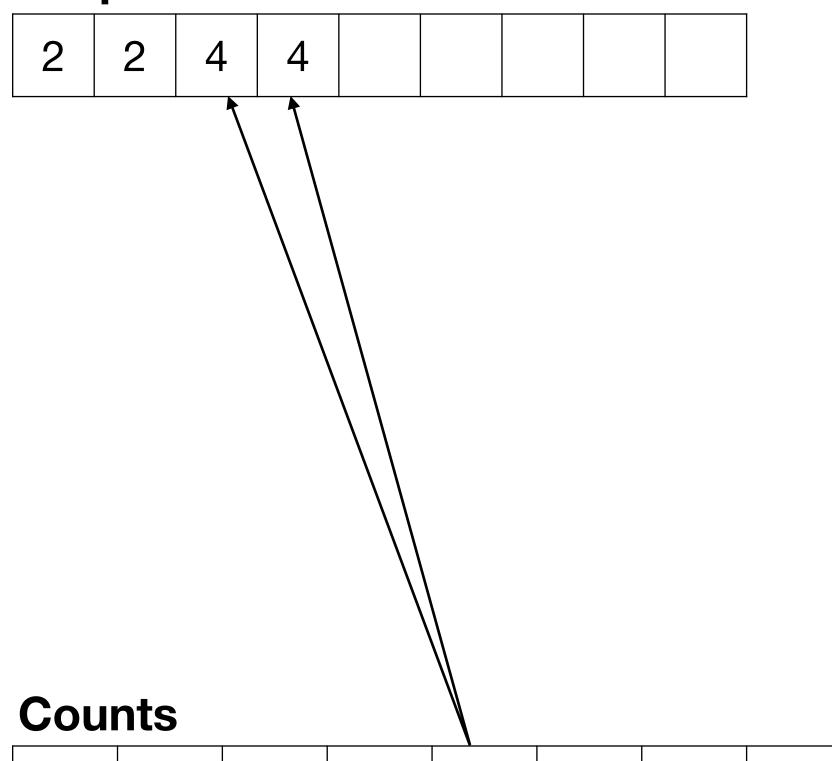
0	1	2	3	4	5	6	7	8	9	10	
0	0	2	0	2	0	1	0	1	3	0	

Output



0	1	2	3	4	5	6	7	8	9	10
0	0	2	0	2	0	1	0	1	3	0

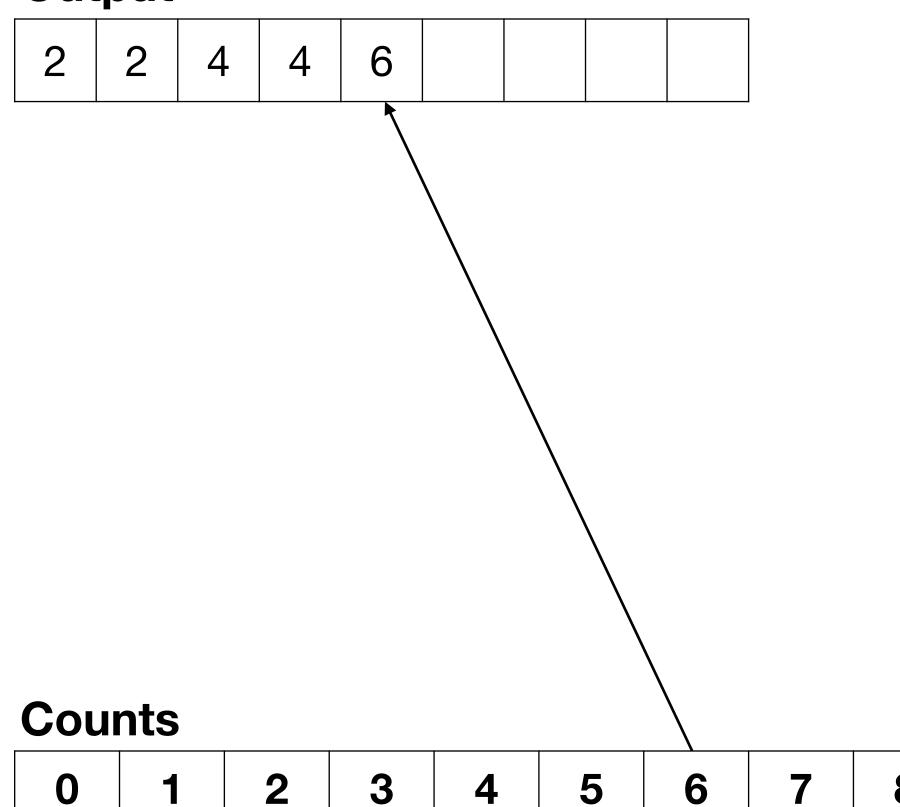
Output



6

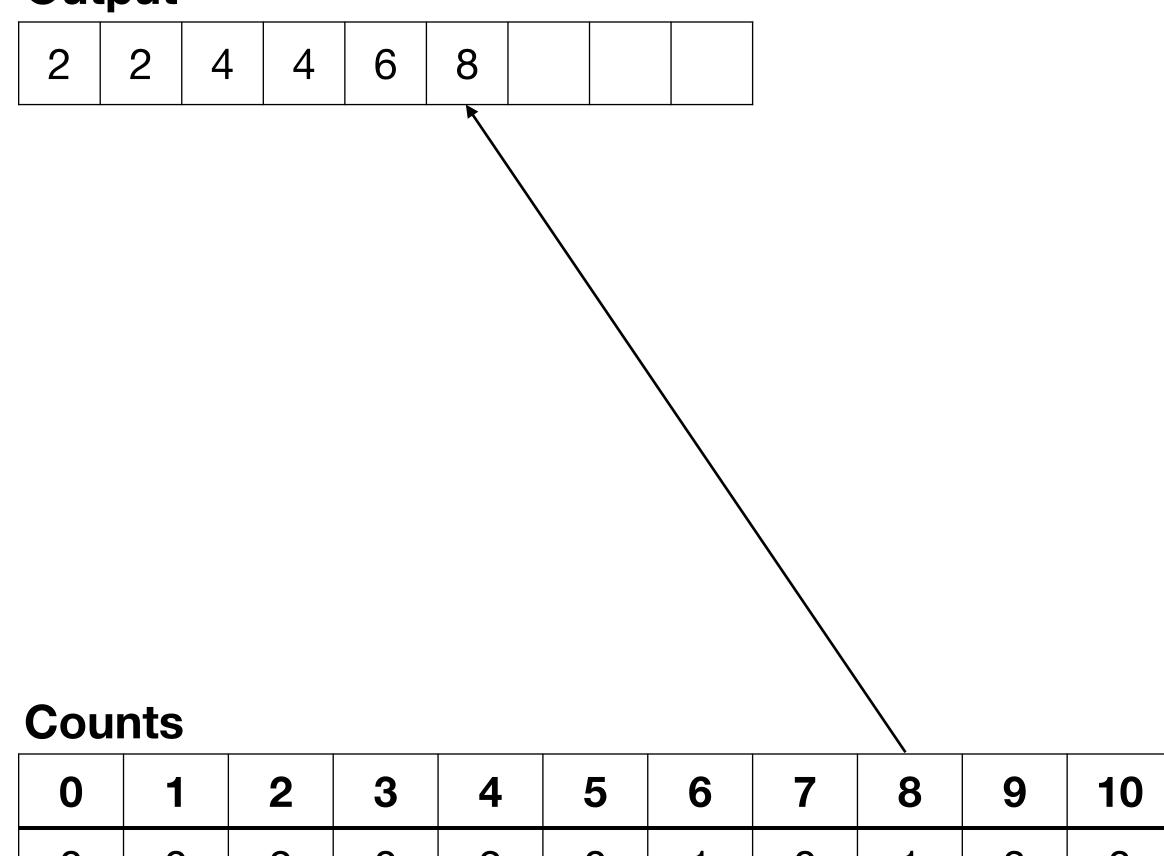
10

Output

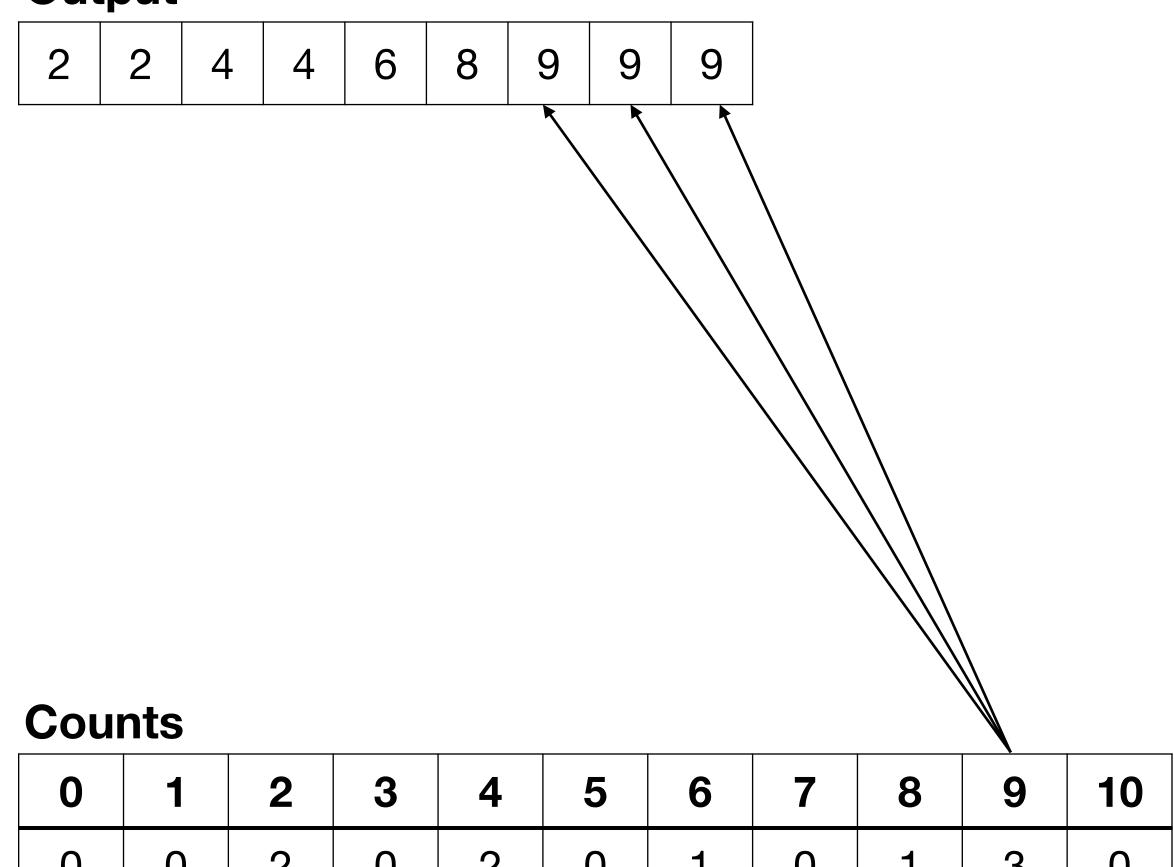


10

Output



Output



Complexity

- 1. Find the range of values: O(n)
- 2. Initialize array: O(n)
- 3. Scan the list to count: O(n)
- 4. Scan the counts to output: O(n)

Overall complexity: O(n)

Limitations?

- Only apply to integers -- need to use the value as array indices
- Need extra space:
 - Counts: O(Range) -- if the input is sparse, this can be a lot
 - Output: O(n)
- This is almost a Map!
 - Key: Integer
 - Value: Counts

Limitations?

- Can we make this work with any value?
 - Sure, instead of having an array of integers, we can have an array of whatever values
- Can we make this work with any key?
 - Turn any key into an integer
 - Make the range of the integer reasonable

Turning any value into an integer

- A hash function maps a key to an integer deterministically:
 - I.e. the same key is always turned into the same integer
 - Hash functions should run in O(1) time
- There are good/bad choices for hash functions

Example: 2-letter word dictionary

- Map 2-letter words to definitions:
 - Key: 2-letter words (string)
 - Value: definitions (string)

ah: used to express delight, relief, regret, or contempt

as: to the same degree or amount

at: used as a function word to indicate presence or occurrence in, on, or near

do: to bring to pass

go: to move on a course

ha: used especially to express surprise, joy, or triumph

he: that male one who is neither speaker nor hearer

hi: used especially as a greeting

• • •

What hash function could we use to map keys to ints?

Example: 2-letter word dictionary

- How many 2-letter words are there?
 - 26 * 26 = 676
- How to map words into [0, 676)?
 - Idea: map a-z: 0-25
 - then, first letter's number * 26 + second letter's number

а	b	С	d	е	f	g	h	i	j	k	I	m	n	0	р	q	r	S	t	u	V	w	X	У	Z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

- $hash(\alpha\beta) = 26\alpha + \beta$
- $hash(go) = 26 \cdot 6 + 14 = 170$

Example: 2-letter word dictionary

Example!

Problem

- Can we extend this function to work for all words?
- https://en.wikipedia.org/wiki/Longest_word_in_English

Word	Letters
Longest chemical	189,819
Longest word in Merriam-Webster	45
Supercalifragilisticexpialidocious	34
Longest word in Shakespeare's works	27

• $26^{27} = 160059109085386090080713531498405298176$

Problem

- $26^{27} = 160059109085386090080713531498405298176$
- Too big for an array!
- Also, English has ~700,000 words; we only need a tiny fraction of these.
- Solution: Compress

Compression

- Generally, hash functions do not care about its output range.
- We use a compression function to put the integer in the reasonable range [0,size)
- Common choice: modulus
 - a % b calculates the remainder of a divided by b
 - a % b always returns an int in the range [0, b)

Compression example

Keys: integer

Table size: 10

hash: itself

• compress: hash % 10

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Compression example

Keys: integer

Table size: 10

hash: itself

• compress: hash % 10

0	
1	
2	
3	
4	
5	
6	
7	7
8	
9	

Compression example

Keys: integer

Table size: 10

hash: itself

• compress: hash % 10

0	
1	
2	
3	
4	
5	
6	
7	7
8	18
9	

Compression example

Keys: integer

Table size: 10

hash: itself

• compress: hash % 10

0	
1	41
2	
3	
4	
5	
6	
7	7
8	18
9	

Compression example

Keys: integer

Table size: 10

hash: itself

• compress: hash % 10

• insert: 7, 18, 41, 35

•

0	
1	41
2	
3	
4	
5	35
6	
7	7
8	18
9	

Compression example

- Keys: integer
- Table size: 10
- hash: itself
- compress: hash % 10
- insert: 7, 18, 41, 35
- What if we try to insert 75?

0	
1	41
2	
3	
4	
5	35
6	
7	7
8	18
9	

Compression example

Keys: integer

• Table size: 10

hash: itself

• compress: hash % 10

• insert: 7, 18, 41, 35

• What if we try to insert 75?

0	
1	41
2	
3	
4	
5	35
6	
7	7
8	18
9	

75

Collision

- Two different keys sometimes end up in the same slot
 - This is called a collision
- Collision has to happen if we have smaller array than the range of hash function
 - Hash function could produce the same integer for two different keys
 - Compression merges different hashes together
- All tables need to handle collision

Handling Collision

- 1. Avoid collisions when possible:
 - 1. Pick a good hash function (e.g. strlen is a terrible hash function)
 - 2. Pick a good table size
- 2. When they arise (inevitably):
 - 1. Have a way to put collisions in a table.

Picking a good hash function

- Minimize collision:
 - What is the worst possible hash function?
 - hash(k) = 1
 - What is the best possible hash function?
 - Every input maps to a distinct output, $f(x) = f(y) \implies x = y$
 - This is called perfect hashing. The two-letter hash function is a perfect hash function.

Picking a good hash function (Example)

- If we want to hash UChicago students:
 - Use their birthdays
 - Month (Jan, Feb, Mar, ...)?
 - Age (0, 1, 2, ..., 100)?
 - Day of month (1, 2, 3, ..., 31)?
 - Use their first name
 - Use their last name
 - Use their student ID

Picking a good hash function

- A good hash function should be:
 - fast
 - collision with (extremely) low probability
 - spreads out the keys
- CS284: Cryptography