Introduction to Algorithms and Data Structures: An example

Ana Echavarría (aechava3@eafit.edu.co)
Santiago Palacio (spalac24@eafit.edu.co)

EAFIT University

July 3, 2015

Overview

Problem Description

a_1	a_2	$\dots \quad \dots$	a_n

- Stream of numbers.
- Count how many times each a_i appears.
- $0 \leq a_i < 10^{12}$ and $n < 10^9$.

Naive Approach

Step 0

a =	4	6	3	0	7	0	6	8	6	8	2	1
count =	0	0	0	0	0	0	0	0	0	0	0	0

Naive Approach

Step 1

Lookup $= 4$

a =	4	6	3	0	7	0	6	8	6	8	2	1
count =	1	0	0	0	0	0	0	0	0	0	0	0

Element $= 4$

Naive Approach

Step 2

Lookup = 6

a =	4	6	3	0	7	0	6	8	6	8	2	1
count =	1	0	0	0	0	0	0	0	0	0	0	0

Element = 4

Naive Approach

Step 3

Lookup = 3

a =	4	6	3	0	7	0	6	8	6	8	2	1
count =	1	0	0	0	0	0	0	0	0	0	0	0

Element = 4

Naive Approach

Step 39

Lookup = 3

a =	4	6	3	0	7	0	6	8	6	8	2	1
count =	1	3	1	0	0	0	0	0	0	0	0	0

Element = 0

Naive Approach

Step 40

Lookup = 0

a =	4	6	3	0	7	0	6	8	6	8	2	1
count =	1	3	1	1	0	0	0	0	0	0	0	0

Element = 0

Naive Approach

Step 41

$$\text{Lookup} = 7$$

a =	4	6	3	0	7	0	6	8	6	8	2	1
count =	1	3	1	1	0	0	0	0	0	0	0	0

$$\text{Element} = 0$$

Naive Approach

Step 42

$$\text{Lookup} = 0$$

a =	4	6	3	0	7	0	6	8	6	8	2	1
count =	1	3	1	2	0	0	0	0	0	0	0	0

$$\text{Element} = 0$$

Naive Approach

Step 43

Lookup = 6

a =	4	6	3	0	7	0	6	8	6	8	2	1
count =	1	3	1	2	0	0	0	0	0	0	0	0

Element = 0

Naive Approach

Step 63

Lookup = 3

a =	4	6	3	0	7	0	6	8	6	8	2	1
count =	1	3	1	2	1	0	0	0	0	0	0	0

Element = 0

Naive Approach

Step 64

Lookup = 0

a =	4	6	3	0	7	0	6	8	6	8	2	1
count =	1	3	1	2	1	1	0	0	0	0	0	0

Element = 0

Naive Approach

Step 144

Lookup = 1

a =	4	6	3	0	7	0	6	8	6	8	2	1
count =	1	3	1	2	1	2	3	2	3	2	1	1

Element = 1

Naive Approach

- Time complexity: $O(n^2)$
- Space complexity: $O(n)$
- Lookup time complexity: $O(n)$

Dynamic Programming

- Similar approach to Naive Algorithm.
- let *count* be an array of $n+1$ slots, where $count[i]$ stores the number of occurrences of the i-th number to the right.
- How can we compute *count*?

Dynamic Programming

Step 0

a =	4	6	3	0	7	0	6	8	6	8	2	1
count =	0	0	0	0	0	0	0	0	0	0	0	0

Step 1

Lookup = End (Stop)

a =	4	6	3	0	7	0	6	8	6	8	2	1
count =	0	0	0	0	0	0	0	0	0	0	0	1

Element = 1

Dynamic Programming

Step 2

Lookup = 1

a =	4	6	3	0	7	0	6	8	6	8	2	1
count =	0	0	0	0	0	0	0	0	0	0	0	1

Element = 2

Step 3

Lookup = End (Stop)

a =	4	6	3	0	7	0	6	8	6	8	2	1
count =	0	0	0	0	0	0	0	0	0	0	1	1

Element = 2

Step 4

Lookup $= 2$

a =	4	6	3	0	7	0	6	8	6	8	2	1
count =	0	0	0	0	0	0	0	0	0	0	1	1

Element $= 8$

Step 5

$\color{blue}{\text{Lookup} = 1}$

a =	4	6	3	0	7	0	6	8	6	8	2	1
count =	0	0	0	0	0	0	0	0	0	0	1	1

$\color{red}{\text{Element} = 8}$

Step 6

Lookup = End (Stop)

a =	4	6	3	0	7	0	6	8	6	8	2	1
count =	0	0	0	0	0	0	0	0	0	1	1	1

Element = 8

Step 11

Lookup = 6

a =	4	6	3	0	7	0	6	8	6	8	2	1
count =	0	0	0	0	0	0	0	0	1	1	1	1

Element = 8

Dynamic Programming

Step 12

Lookup $= 8$ (Found, Stop)

a =	4	6	3	0	7	0	6	8	6	8	2	1
count =	0	0	0	0	0	0	0	2	1	1	1	1

Element $= 8$

Step 13

Lookup = 8

a =	4	6	3	0	7	0	6	8	6	8	2	1
count =	0	0	0	0	0	0	0	2	1	1	1	1

Element = 6

Dynamic Programming

Final Step

Lookup = End (Stop)

a =	4	6	3	0	7	0	6	8	6	8	2	1
count =	1	3	1	2	1	1	2	2	1	1	1	1

Element = 4

- Time complexity: $O(k * n)$ where k is the number of distinct elements in the array.
- Space complexity: $O(n)$
- Lookup time complexity: $O(n)$

Exercise

- Using this approach, is there a way to modify the algorithm so that it returns a list with all the elements of the array and their count?

Dynamic Programming- Solution

At each element, put a flag if it was used to get another answer (i.e. there's another element to the left).

a =	4	6	3	0	7	0	6	8	6	8	2	1
count =	1	3	1	2	1	1	2	2	1	1	1	1
						●	●		●	●		

This way, we can get the elements that are not flagged.

Counting Sort

- Have an array *count* of $\max\{a_i | 0 \leq i < n\}$ spaces.
- For each element e in the array, increment $count[e]$ by 1.

Counting Sort

Step 0

$$a = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|}\hline 4 & 6 & 13 & 0 & 6 & 0 & 7 & 8 & 6 & 8 & 2 & 1 \\ \hline\end{array}$$

Find maximum element $= 13$.

Counting Sort

Step 0

a =	4	6	13	0	6	0	7	8	6	8	2	1

Find maximum element = 13.

index =	0	1	2	3	4	5	6	7	8	9	10	11	12	13
count =	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Counting Sort

Step 1

Element = 4

a =

4	6	13	0	6	0	7	8	6	8	2	1

index =	0	1	2	3	4	5	6	7	8	9	10	11	12	13
count =	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Step 1

Element = 4

a =

4	6	13	0	6	0	7	8	6	8	2	1

index =	0	1	2	3	4	5	6	7	8	9	10	11	12	13
count =	0	0	0	0	1	0	0	0	0	0	0	0	0	0

Counting Sort

Step 2

Element = 6

a =

4	6	13	0	6	0	7	8	6	8	2	1

index =	0	1	2	3	4	5	6	7	8	9	10	11	12	13
count =	0	0	0	0	1	0	0	0	0	0	0	0	0	0

Counting Sort

Step 2

Element = 6

a =	4	6	13	0	6	0	7	8	6	8	2	1

index =	0	1	2	3	4	5	6	7	8	9	10	11	12	13
count =	0	0	0	0	1	0	1	0	0	0	0	0	0	0

Counting Sort

Step 3

Element = 13

a =

4	6	13	0	6	0	7	8	6	8	2	1

index =	0	1	2	3	4	5	6	7	8	9	10	11	12	13
count =	0	0	0	0	1	0	1	0	0	0	0	0	0	0

Counting Sort

Step 3

Element = 13

a =

4	6	13	0	6	0	7	8	6	8	2	1

index =	0	1	2	3	4	5	6	7	8	9	10	11	12	13
count =	0	0	0	0	1	0	1	0	0	0	0	0	0	1

Step 4

Element = 0

a =

4	6	13	0	6	0	7	8	6	8	2	1

index =	0	1	2	3	4	5	6	7	8	9	10	11	12	13
count =	0	0	0	0	1	0	1	0	0	0	0	0	0	1

Step 4

Element = 0

a =

4	6	13	0	6	0	7	8	6	8	2	1

index =	0	1	2	3	4	5	6	7	8	9	10	11	12	13
count =	1	0	0	0	1	0	1	0	0	0	0	0	0	1

Counting Sort

Step 5

Element = 6

a =

4	6	13	0	6	0	7	8	6	8	2	1

index =	0	1	2	3	4	5	6	7	8	9	10	11	12	13
count =	1	0	0	0	1	0	1	0	0	0	0	0	0	1

Step 5

Element = 6

a =

4	6	13	0	6	0	7	8	6	8	2	1

index =	0	1	2	3	4	5	6	7	8	9	10	11	12	13
count =	1	0	0	0	1	0	2	0	0	0	0	0	0	1

Counting Sort

Step 12

Element = 1

a =

4	6	13	0	6	0	7	8	6	8	2	1

index =	0	1	2	3	4	5	6	7	8	9	10	11	12	13
count =	2	0	1	0	1	0	3	1	2	0	0	0	0	1

Counting Sort

Step 12

Element = 1

a =

4	6	13	0	6	0	7	8	6	8	2	1

index =	0	1	2	3	4	5	6	7	8	9	10	11	12	13
count =	2	1	1	0	1	0	3	1	2	0	0	0	0	1

Counting Sort

Final result

a =

4	6	13	0	6	0	7	8	6	8	2	1

index =	0	1	2	3	4	5	6	7	8	9	10	11	12	13
count =	2	1	1	0	1	0	3	1	2	0	0	0	0	1

Counting Sort

- Time complexity: $O(n)$
- Space complexity: $O(\max a_i)$
- Lookup time complexity: $O(1)$

- With Counting Sort, there's a lot of unused space.
- Take a uniformly distributed hash function $f : A \mapsto B$ where A is our original space (e.g. $[1, 10^{12}]$) and B is a new smaller space (e.g. $[1, 10^8]$)

What is a good number for the size of B?

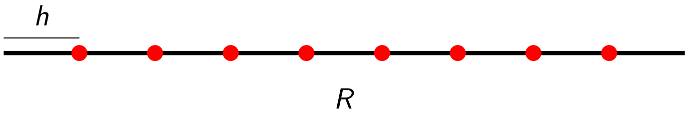
Hashing

What is a good number for the size of B?

- The size of the original array.

What is a good number for the size of B?

- The size of the original array.
- Get a random hash function (with large range R) and check for the minimum hash h. $|R|/h$ would be a good estimate.



Hashing

- Time complexity: $O(n)$
- Space complexity: $O(|B|)$
- Lookup time complexity: $O(1)$

But what about collitions?

- Birthday Paradox. With 23 people, there's 50.7% chance of a shared birthday.
- In general, for a space of size N and k random elements of N:

$$
\begin{aligned}
P &= 1 - \tfrac{N-1}{N} \times \tfrac{N-2}{N} \times \ldots \times \tfrac{N-(k-1)}{N} \\
&\approx 1 - e^{\frac{-k(k-1)}{2N}} \\
&\approx \frac{k^2}{2N}
\end{aligned}
$$

- Expected number of collitions:

$$k - N * (1 - ((N-1)/N)^k)$$

Traditional Sorting

- Use a traditional sorting technique, e.g. Quick Sort or Merge Sort.
- Count consecutive equal elements

Traditional Sorting

Step 0

4	6	13	0	6	0	7	8	6	8	2	1

a =

Step 1

Element = 0

a =

0	0	1	2	4	6	6	6	7	8	8	13

results = $\boxed{0,\ 1}$

Traditional Sorting

Step 2

Element = 0

a =	0	0	1	2	4	6	6	6	7	8	8	13

results = $\boxed{0,\ 2}$

Step 3

Element = 1

a =

0	0	1	2	4	6	6	6	7	8	8	13

results =

0, 2	1, 1

Traditional Sorting

Step 4

Element = 2

a =

0	0	1	2	4	6	6	6	7	8	8	13

results =

0, 2	1, 1	2, 1

Step 5

Element = 4

a =

0	0	1	2	4	6	6	6	7	8	8	13

results =

0, 2	1, 1	2, 1	4, 1

Traditional Sorting

Step 6

Element $= 6$

a =											
0	0	1	2	4	6	6	6	7	8	8	13

results =				
0, 2	1, 1	2, 1	4, 1	6, 1

Traditional Sorting

Step 7

Element = 6

a =

0	0	1	2	4	6	6	6	7	8	8	13

results =

0, 2	1, 1	2, 1	4, 1	6, 2

Traditional Sorting

Step 8

Element = 6

a =

0	0	1	2	4	6	6	6	7	8	8	13

results =

0, 2	1, 1	2, 1	4, 1	6, 3

Traditional Sorting

Step 12

Element = 13

a =

0	0	1	2	4	6	6	6	7	8	8	13

results =

0, 2	1, 1	2, 1	4, 1	6, 3	7, 1	8, 2	13, 1

Traditional Sorting

- Time complexity: $O(n * log(n))$
- Space complexity: $O(n)$
- Lookup time complexity: $O(log(n))$

Trees

- Build a binary search tree with the input.
- At each node, keep track of the times it has been seen.

Step 0

a =

4	6	13	0	6	0	7	8	6	8	2	1

Step 1

Element $= 4$

a $=$

4	6	13	0	6	0	7	8	6	8	2	1

el $= 4$ cnt $= 1$

Trees

Step 2

Element = 6

a =

4	6	13	0	6	0	7	8	6	8	2	1

el = 4 cnt = 1

el = 6 cnt = 1

Step 3

Element $= 13$

a $=$

4	6	13	0	6	0	7	8	6	8	2	1

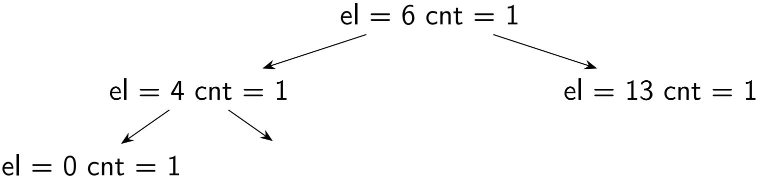
$$
\begin{array}{ccc}
 & \text{el} = 6 \ \text{cnt} = 1 & \\
\swarrow & & \searrow \\
\text{el} = 4 \ \text{cnt} = 1 & & \text{el} = 13 \ \text{cnt} = 1
\end{array}
$$

Step 4

Element = 0

a =

4	6	13	0	6	0	7	8	6	8	2	1



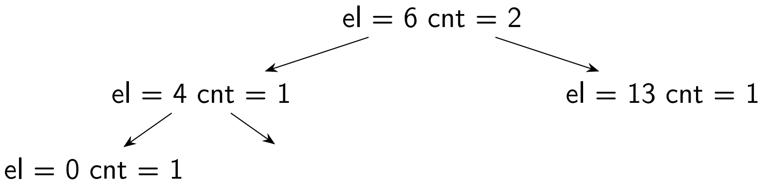
Trees

Step 5

Element = 6

a =

4	6	13	0	6	0	7	8	6	8	2	1

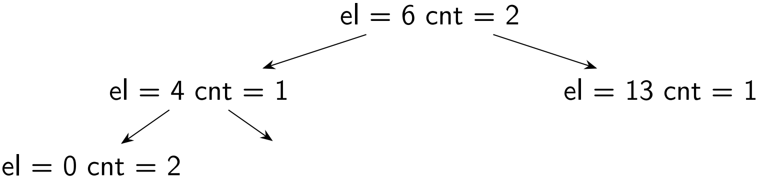


Step 6

Element = 0

a =

4	6	13	0	6	0	7	8	6	8	2	1

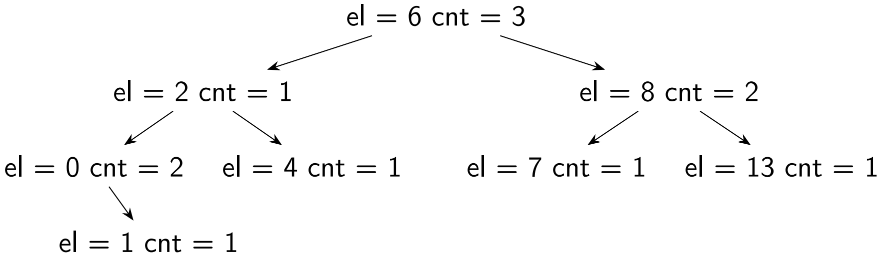


Final Step

Element = 1

a =

4	6	13	0	6	0	7	8	6	8	2	1



Trees

- Time complexity: $O(n * log(n))$
- Space complexity: $O(n)$
- Lookup time complexity: $O(log(n))$