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

1. Algorithms overview
2. Algorithms real time performance
3. Algorithms number of operations performance
4. Finding elements with a certain frequency
   - Finding a majority
   - Generalized problem

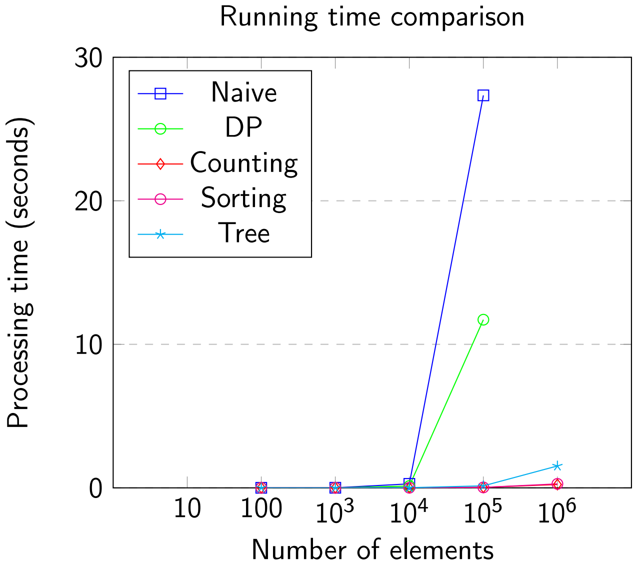
Naive Run through the whole array looking for each element. $O(n^2)$ time.

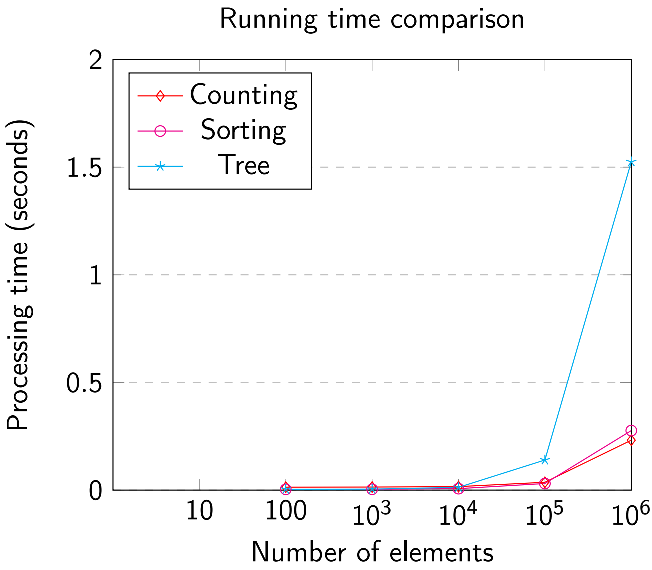
DP Remember already seen elements when looking for them. $O(n * k)$ time.

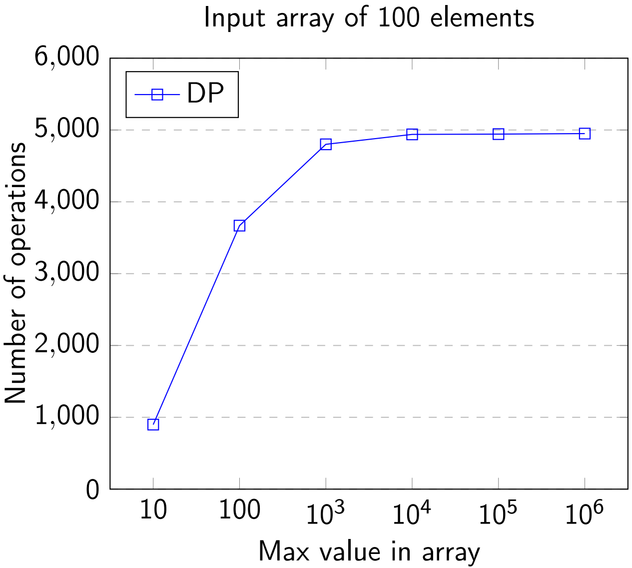
Counting Keep an array of the whole range for counters. $O(n)$ time.

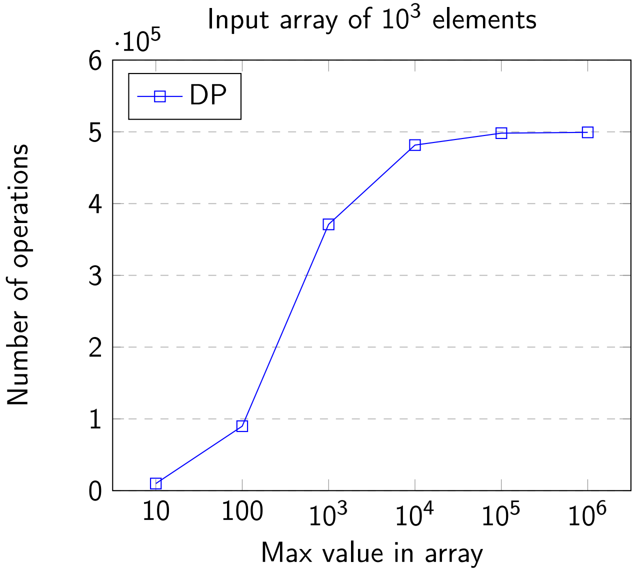
Sorting Sort the elements and count consecutive numbers. $O(n * \log(n))$ time.

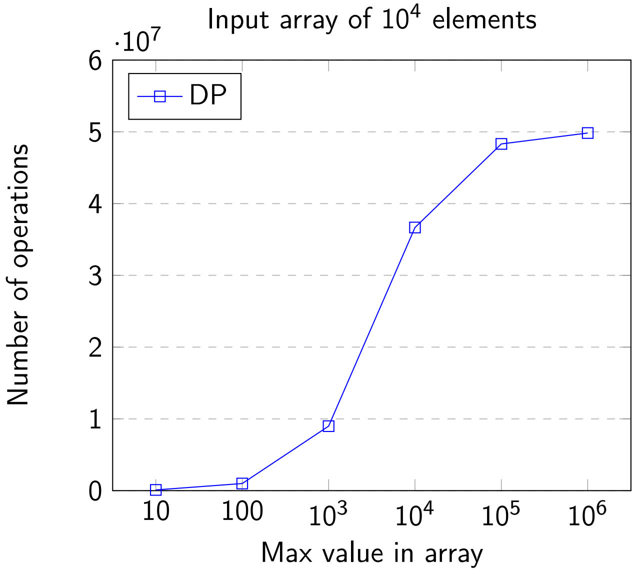
Tree Store the elements in a tree and keep the counter at each node. $O(n * \log(n))$ time.

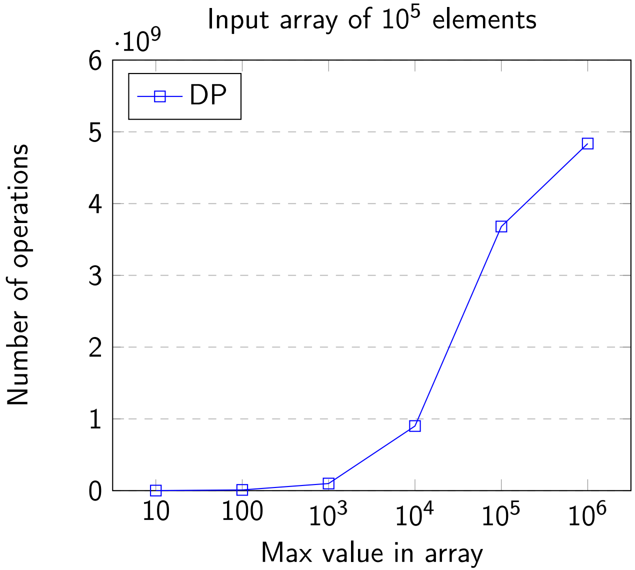

Running time comparison
Processing time (seconds)
30
20
10
0
Naive
DP
Counting
Sorting
Tree
10
100
10^3
10^4
10^5
10^6
Number of elements


Running time comparison
Counting
Sorting
Tree
Processing time (seconds)
Number of elements
0
0.5
1
1.5
2
10
100
10^3
10^4
10^5
10^6


Input array of 100 elements
DP
Number of operations
0
1,000
2,000
3,000
4,000
5,000
6,000
10
100
10^3
10^4
10^5
10^6
Max value in array


Input array of 10^3 elements
$\cdot 10^5$
6
5
4
3
2
1
0
DP
Number of operations
10
100
10^3
10^4
10^5
10^6
Max value in array


Input array of 10^4 elements
$\cdot 10^7$
6
5
4
3
2
1
0
DP
Number of operations
10
100
10^3
10^4
10^5
10^6
Max value in array


Input array of 10^5 elements
$\cdot 10^9$
6
5
4
3
2
1
0
DP
Number of operations
10
100
10^3
10^4
10^5
10^6
Max value in array

1. Algorithms overview
2. Algorithms real time performance
3. Algorithms number of operations performance
4. Finding elements with a certain frequency
   - Finding a majority
   - Generalized problem

Finding a majority

Given a *stream* of numbers, find which element, if any, is repeated more than 50% of the time.
Note: If there is no such element, any output is allowed.

Sorting solution

- Sort the complete array.
- The element in the middle of the array must be the number we're looking for.

  Disadvantages

- $O(n)$ space, $O(n * log(n))$ time.

Constant space solution

- Keep two variables, *count* and *element*.
- Let $count = 0$ and $element = null$.
- For each element e in the *stream*
  - If $e = element$, increase *count* by 1.
  - Otherwise:
    - If $count > 0$, decrease it by 1.
    - Otherwise, let $element = e$ and $count = 1$.

Example

stream =

6	3	4	3	3	3	3	2	1

element = null

count = 0

Example

stream =

6	3	4	3	3	3	3	2	1

element = 6
count = 1

Example

stream = | 6 | 3 | 4 | 3 | 3 | 3 | 3 | 2 | 1 |

element = 6

count = 0

Example

stream =

6	3	4	3	3	3	3	2	1

element = 4
count = 1

Example

6	3	4	3	3	3	3	2	1

stream = (above)

element = 4 *

count = 0

Example

stream =

6	3	4	3	3	3	3	2	1

element = 3

count = 1

Example

stream =	6	3	4	3	3	3	3	2	1

element = 3

count = 2

Example

stream = | 6 | 3 | 4 | 3 | 3 | 3 | 3 | 2 | 1 |

$p = 3$

element = 3
count = 3

Example

stream = | 6 | 3 | 4 | 3 | 3 | 3 | 3 | 2 | 1 |

okup = 2

element = 3
count = 2

Example

stream = | 6 | 3 | 4 | 3 | 3 | 3 | 3 | 2 | 1 |

Lookup = 1

element = 3
count = 1

Complexity

- $O(n)$ time.
- $O(\log(\max\{stream\}) + \log(n))$ space.

Generalized problem

Given a *stream* of numbers, find which elements, if any, appear more than $\lfloor n/(k+1) \rfloor$ times.
Note: If there are no such elements, any output is allowed.

k counters soulution

- Very similar idea as previous algorithm. Instead of a single counter use k counters.
- Initialize k counters with value 0.
- For every element e in the stream:
  - If a counter for e exists, increase its value by one.
  - If no such counter exists:
    - If there is a counter set to 0, assign it to e with value 1.
    - Otherwise decrease all counters by one.

Example with $k = 2$

stream =

1	3	4	3	3	1	3	2	1	1

element =	null	null
count =	0	0

Example with $k = 2$

stream =

1	3	4	3	3	1	3	2	1	1

element =	1	null
count =	1	0

Example with $k = 2$

stream =									
1	3	4	3	3	1	3	2	1	1

element =	1	3
count =	1	1

Example with $k = 2$

stream =

1	3	4	3	3	1	3	2	1	1

element =	1	3
count =	0	0

Example with $k = 2$

stream =

1	3	4	3	3	1	3	2	1	1

element =	1	3
count =	0	1

Example with $k = 2$

stream =

1	3	4	3	3	1	3	2	1	1

element =	1	3
count =	0	2

Example with $k = 2$

stream =

1	3	4	3	3	1	3	2	1	1

element =	1	3
count =	1	2

Example with $k = 2$

stream = | 1 | 3 | 4 | 3 | 3 | 1 | 3 | 2 | 1 | 1 |

element =	1	3
count =	1	3

Example with $k = 2$

stream =

1	3	4	3	3	1	3	2	1	1

element =	1	3
count =	0	2

$p = 2$

Example with $k = 2$

stream =	1	3	4	3	3	1	3	2	1	1

okup = 1

element =	1	3
count =	1	2

Example with $k = 2$

stream =

1	3	4	3	3	1	3	2	1	1

Lookup = 1

element =	1	3
count =	2	2

Complexity

- $O(k * n)$ time.
- $O(k * (\log(\max\{stream\}) + \log(n)))$ space.