Efficiency Analysis of Non-Recursive Algorithms

Teaching Talk

https://shorturl.at/yWT7P

Alberto Castro-Hernandez, 2024

Outline

- Motivation
- Summation formulas
- Basic complexity classes
- General plan for efficiency analysis
- Algorithm: Maximum value
- Algorithm: Sequential search
  - Best, Average, and Worst case scenarios
- Review
- Additional exercise

Motivation

- Why we analyze algorithms’ efficiency?
  - To identify its complexity (order of growth of its number of operations)
  - To determine how good an algorithm is in comparison to others
- Efficiency types
  - Time efficiency: how fast the algorithm runs
  - Space efficiency: how much extra memory uses
- Approaches
  - Experimental studies (run programs)
  - Asymptotic Algorithm analysis

Summation formulas
(1)

- Summation

$$1 + 1 + 1 + 1 + 1 + 1 + 1 = 7 \qquad \sum_{i=1}^{7} 1 = 7 \qquad \sum_{i=0}^{6} 1 = 7 \qquad \sum_{i=13}^{19} 1 = 7$$

- Formula

$$\boxed{\sum_{i=l}^{u} 1 = u - l + 1}$$

- Application

$$\sum_{i=1}^{7} 1 = 7 - 1 + 1 = 7 \qquad \sum_{i=2}^{12} 1 = 12 - 2 + 1 = 11$$

Summation formulas
(2)

- Summation

$$1 + 2 + 3 + 4 + 5 + 6 = 21 \qquad \sum_{i=1}^{6} i = 21 \qquad 1 + 2 + 3 + 4 = 10 \qquad \sum_{i=1}^{4} i = 10$$

- Formula

$$\boxed{\sum_{i=1}^{n} i = \frac{n(n+1)}{2}}$$

- Application

$$\sum_{i=1}^{6} i = \frac{6(6+1)}{2} = 21 \qquad \sum_{i=1}^{8} i = \frac{8(8+1)}{2} = 36$$

Basic Complexity Classes

List

Lower

Order of Growth

Higher

Class	Name
1	Constant
n	Linear
log n	Logarithmic
n log n	Linearithmic
n^2	Quadratic
n^3	Cubic
2^n	Exponential
n!	Factorial

Faster

Algorithm Efficiency

Slower

Basic Complexity Classes

Asymptotic Order of Growth

- A way of comparing functions that ignores constant factors and small input sizes
- $\Theta(g(n))$: Class of functions that grow at the same rate as $g(n)$

$4n^3$ $\qquad$ $4n^3 \in \Theta(n^3)$

$11n^2 + 18$ $\qquad$ $11n^2 + 18 \in \Theta(n^2)$

$2^n + 3n^3 - 5$ $\qquad$ $2^n + 3n^3 - 5 \in \Theta(2^n)$

General Plan for Efficiency Analysis

- Decide on a parameter (or parameters) indicating an input's size.
- Identify the algorithm's basic operation (As a rule, it is indicated in the inner cycle).
- Check whether the number of times the basic operation is executed depends on the size of an input. If it also depends on some additional property, the worst-case, average-case, and, if necessary, best-case efficiencies have to be investigated separately.
- Set up a sum expressing the number of times the algorithm's basic operation is executed.
- Using standard formulas and rules of sum manipulation, either find a closed-form formula for the count or, at the very least, establish its order of growth.

Algorithm: Maximum value
Problem and algorithm

- Problem
  - Consider the problem of finding the value of the largest element in a list of n numbers. For simplicity, we assume the list is implemented as an array.

ALGORITHM *MaxElement*($A[0..n-1]$)
　//Determines the value of the largest element in a given array
　//Input: An array $A[0..n-1]$ of real numbers
　//Output: The value of the largest element in A
　$maxval \leftarrow A[0]$
　for $i \leftarrow 1$ **to** $n-1$ **do**
　　if $A[i] > maxval$
　　　$maxval \leftarrow A[i]$
　return $maxval$

Algorithm: Maximum value

Parameter

ALGORITHM *MaxElement*($A[0..n-1]$)

//Determines the value of the largest element in a given array
//Input: An array $A[0..n-1]$ of real numbers
//Output: The value of the largest element in A

$maxval \leftarrow A[0]$
for $i \leftarrow 1$ **to** $n-1$ **do**
  if $A[i] > maxval$
    $maxval \leftarrow A[i]$
return $maxval$

n: number of real numbers

in the list

Algorithm: Maximum value

Basic operation

ALGORITHM $MaxElement(A[0..n-1])$

//Determines the value of the largest element in a given array
//Input: An array $A[0..n-1]$ of real numbers
//Output: The value of the largest element in A

$maxval \leftarrow A[0]$
for $i \leftarrow 1$ **to** $n-1$ **do**

Inner loop
  if $A[i] > maxval$ Basic operation
    $maxval \leftarrow A[i]$

return $maxval$

Algorithm: Maximum value

Worst, Average, and Best cases (if necessary)

- The number of comparisons will be the same for all arrays of size n
- Therefore, in terms in this metric, there is no need to distinguish among the worst, average, and best cases

Algorithm: Maximum value

Sum

ALGORITHM *MaxElement*$(A[0..n-1])$

//Determines the value of the largest element in a given array
//Input: An array $A[0..n-1]$ of real numbers
//Output: The value of the largest element in A
$maxval \leftarrow A[0]$
for $i \leftarrow 1$ **to** $n-1$ **do**
  if $A[i] > maxval$
    $maxval \leftarrow A[i]$
return $maxval$

Inner loop

Basic operation

$$C(n) = \sum_{i=1}^{n-1} 1$$

Algorithm: Maximum value
Sum manipulation and classification

- Formula $$\sum_{i=l}^{u} 1 = u - l + 1$$
- Sum manipulation

$$C(n) = \sum_{i=1}^{n-1} 1 = (n-1) - 1 + 1 = n - 1$$

$$C(n) = n - 1$$

$$n - 1 \in \Theta(n)$$

Class	Name
1	Constant
n	Linear
log n	Logarithmic
n log n	Linearithmic
n^2	Quadratic
n^3	Cubic
2^n	Exponential
n!	Factorial

Algorithm: Sequential search
Problem and algorithm

- Problem
  - Given a value K, search for it in a given array of n elements.
- Algorithm

ALGORITHM *SequentialSearch(A[0..n-1], K)*
    // Searches for a given value in a given array by sequential search
    // Input: An array A[0..n-1] and a search key K
    // Output: The index of the first element in A that matches K
    //   or -1 if there are no matching elements
    $i \leftarrow 0$
    while $i < n$ ***and*** $A[i] \neq K$ ***do***
        $i \leftarrow i + 1$
    if $i < n$ ***return*** i
    else return *-1*

Algorithm: Sequential search

Parameter

ALGORITHM *SequentialSearch(A[0..n-1], K)*

// Searches for a given value in a given array by sequential search
// Input: An array A[0..n-1] and a search key K
// Output: The index of the first element in A that matches K
//   or -1 if there are no matching elements

i $\leftarrow$ *0*
while *i* < *n* ***and*** *A[i]* $\neq$ *K* ***do***
    i $\leftarrow$ *i* + *1*
if *i* < *n* ***return*** *i*
else return *-1*

n: number of real numbers

in the list

Algorithm: Sequential search

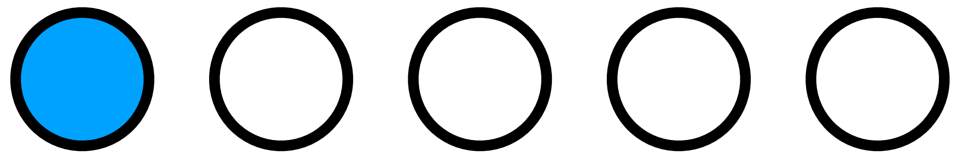
Basic operation

ALGORITHM *SequentialSearch(A[0..n-1], K)*
// Searches for a given value in a given array by sequential search
// Input: An array A[0..n-1] and a search key K
// Output: The index of the first element in A that matches K
//   or -1 if there are no matching elements
$i \leftarrow 0$
Inner loop ***while*** $i < n$ ***and*** $A[i] \neq K$ ***do*** Basic operation
    $i \leftarrow i + 1$
if $i < n$ ***return*** i
else return -1

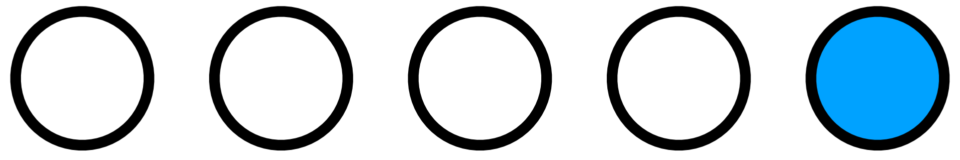
Algorithm: Sequential search

Scenarios

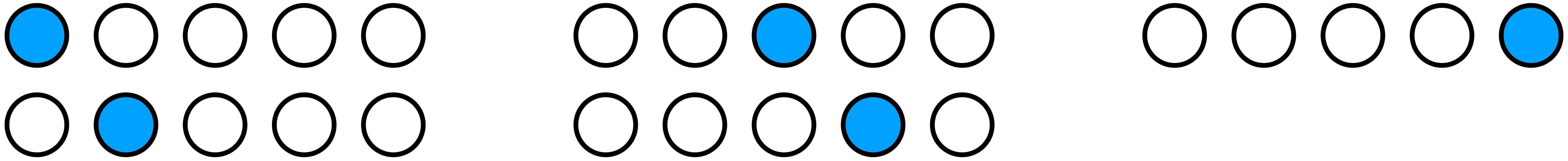
- Best case scenario
  - K (blue circle) is in the first position in the array



- Worst case scenario
  - K (blue circle) is in the last position in the array



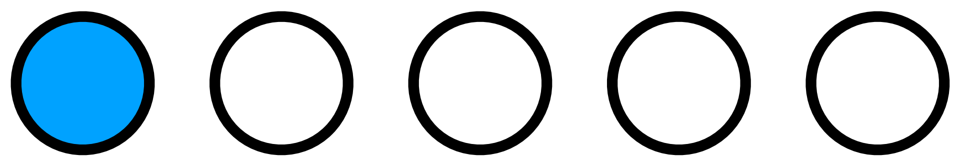
- Average case scenario
  - The average of comparisons in all the possible cases



Algorithm: Sequential search

Best case's sum and manipulation

- Best case scenario
  - K (blue circle) is in the first position in the array



- If K is in the first position, it only takes 1 comparison to find it.

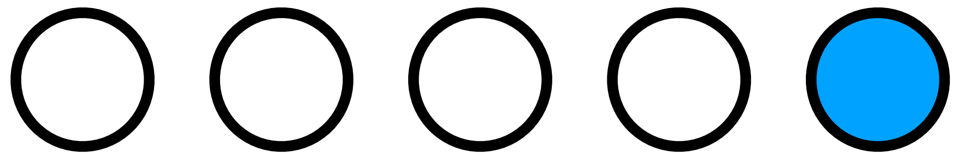
$$C(n)_{best} = 1$$

$$C(n)_{best} \in \Theta(1)$$

Algorithm: Sequential search

Worst case's sum

- Worst case scenario
  - K (blue circle) is in the last position in the array



ALGORITHM *SequentialSearch(A[0..n-1], K)*

// Searches for a given value in a given array by sequential search

// Input: An array A[0..n-1] and a search key K

// Output: The index of the first element in A that matches K

//   or -1 if there are no matching elements

i ← *0*

Inner loop **while** $i < n$ **and** $A[i] \neq K$ **do** Basic operation

    i ← *i + 1*

if $i < n$ **return** *i*

else return *-1*

$$\sum_{i=0}^{n-1} 1$$

Algorithm: Sequential search

Worst case's sum manipulation

- Formula

$$\sum_{i=l}^{u} 1 = u - l + 1$$

- Sum manipulation

$$C(n)_{worst} = \sum_{i=0}^{n-1} 1 = (n-1) - 0 + 1 = n$$

$$C(n)_{worst} = n$$

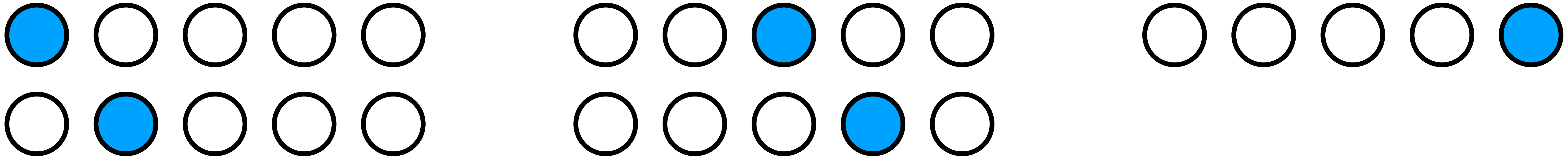
$$n \in \Theta(n)$$

Class	Name
1	Constant
n	Linear
log n	Logarithmic
n log n	Linearithmic
n^2	Quadratic
n^3	Cubic
2^n	Exponential
n!	Factorial

Algorithm: Sequential search

Average case's sum

- Average case scenario
  - The average of comparisons in all the possible cases



In this example,

$$C(5)_{average} = \frac{1+2+3+4+5}{5}$$

In general,

$$C(n)_{average} = \frac{1+2+3+\ldots+n}{n}$$

Algorithm: Sequential search

Average case's sum and manipulation

- Formula

$$\sum_{i=1}^{n} i = 1 + 2 + 3 + \ldots + n = \frac{n(n+1)}{2}$$

- Sum manipulation

$$C(n)_{average} = \frac{1 + 2 + 3 + \ldots + n}{n} = \frac{\sum_{i=1}^{n} i}{n}$$

$$C(n)_{average} = \frac{\frac{n(n+1)}{2}}{n} = \frac{n(n+1)}{2n} = \frac{n+1}{2} = \frac{1}{2}n + \frac{1}{2}$$

$$C(n)_{average} \in \Theta(n)$$

Class	Name
1	Constant
n	Linear
log n	Logarithmic
n log n	Linearithmic
n^2	Quadratic
n^3	Cubic
2^n	Exponential
n!	Factorial

Algorithm: Sequential search

Summary

$$C(n)_{best} \in \Theta(1)$$

$$C(n)_{worst} \in \Theta(n)$$

$$C(n)_{average} \in \Theta(n)$$

Review

General Plan for Efficiency Analysis

- Decide on a parameter indicating an input's size
- Identify the algorithm's basic operation
- Check whether the number of times the basic operation is executed depends also on some additional property. If so, worst-case, average-case, and best-case efficiencies have to be investigated
- Set up a sum expressing the number of times the algorithm's basic operation is executed
- Find a closed-form formula for the count or, at the very least, establish its order of growth

Additional exercise

Matrix multiplication

https://shorturl.at/yWT7P

Thank you!