COMP9020 Week 5
Term 3, 2019
Algorithmic analysis

Summary of topics

- Motivation
- Standard approach
- Examples
- Simplifying with worst-case and big-O
- Recursive examples

Summary of topics

- Motivation
- Standard approach
- Examples
- Simplifying with worst-case and big-O
- Recursive examples

Algorithmic analysis: motivation

Want to compare algorithms – particularly ones that can solve *arbitrarily large* instances.

We would like to be able to talk about the resources (running time, memory, energy consumption) required by a program/algorithm as a function $f(n)$ of some parameter n (e.g. the size) of its input.

Example

How long does a given sorting algorithm take to run on a list of n elements?

Issues

Problems

- The exact resources required for an algorithm are difficult to pin down. Heavily dependent on:
    - Environment the program is run in (hardware, software, choice of language, external factors, etc)
    - Choice of inputs used

Issues

Problems

- The exact resources required for an algorithm are difficult to pin down. Heavily dependent on:
  - Environment the program is run in (hardware, software, choice of language, external factors, etc)
  - Choice of inputs used
- Cost functions can be complex, e.g.

$$2n\log(n) + (n-100)\log(n)^2 + \frac{1}{2^n}\log(\log(n))$$

Need to identify the "important" aspects of the function.

Order of growth

Example

Consider two time-cost functions:

- $f_1(n) = \frac{1}{10}n^2$ milliseconds, and
- $f_2(n) = 10n \log n$ milliseconds

Input size	$f_1(n)$	$f_2(n)$
100	0.01s	2s
1000	1s	30s
10000	1m40s	6m40s
100000	2h47m	1h23m
1000000	11d14h	16h40h
10000000	3y3m	8d2h

Summary of topics

- Motivation
- Standard approach
- Examples
- Simplifying with worst-case and big-O
- Recursive examples

Algorithmic analysis

Asymptotic analysis is about how costs **scale** as the input increases.

Standard (default) approach:

- Consider **asymptotic growth** of cost functions
- Consider **worst-case** (highest cost) inputs
- Consider **running time** cost: number of **elementary operations**

NB

Other common analyses include:

- *Average-case analysis*
- *Space (memory) cost*

Elementary operations

Informally: A single computational "step"; something that takes a constant number of computation cycles.

Examples:

- Arithmetic operations
- Comparison of two values
- Assignment of a value to a variable
- Accessing an element of an array
- Calling a function
- Returning a value
- Printing a single character

NB

Count operations up to a constant factor, $O(1)$, rather than an exact number.

Summary of topics

- Motivation
- Standard approach
- Examples
- Simplifying with worst-case and big-O
- Recursive examples

Examples

Example

Squaring a number (First version):

$$\texttt{square}(n):$$
$$\quad \text{return } n * n$$

Examples

Example

Squaring a number (First version):

$$\texttt{square}(n):$$
$$\quad \text{return } n * n \qquad\qquad O(1)$$

Examples

Example

Squaring a number (First version):

$$\texttt{square}(n):$$
$$\quad \text{return } n * n \qquad\qquad O(1)$$

Running time: $O(1)$

Running time vs Execution time

Previous example shows one difference between running time and execution time.

In general, running time only *approximates* execution time:

- Simplifying assumptions about elementary operations
- Hidden constants in big-O
- Big-O only looks at limiting performance as n gets large.

Examples

- Implementations of `square`(n) will take longer as n gets bigger
- A program that "solves chess" will run in $O(1)$ time.

Examples

Example

Squaring a number (Second version):

$\texttt{square}(n)$:
  $r := 0$
  for $i = 1$ to n :
    $r := r + n$
  return r

Examples

Example

Squaring a number (Second version):

$\texttt{square}(n)$:
  $r := 0$ $\qquad O(1)$
  for $i = 1$ to n :
    $r := r + n$
  return r

Examples

Example

Squaring a number (Second version):

$\texttt{square}(n)$:
    $r := 0$ $\quad O(1)$
    for $i = 1$ to n : $\quad O(1)$
        $r := r + n$
    return r

Examples

Example

Squaring a number (Second version):

$\texttt{square}(n)$:
  $r := 0$ $O(1)$
  for $i = 1$ to n : $O(1)$
    $r := r + n$ $O(1)$
  return r

Examples

Example

Squaring a number (Second version):

$$
\begin{array}{llll}
\texttt{square}(n): & & & \\
\quad r := 0 & & & O(1) \\
\quad \text{for } i = 1 \text{ to } n: & O(1) & \multirow{2}{*}{$\Big|\ n$ times} & \\
\qquad r := r + n & O(1) & & \\
\quad \text{return } r & & &
\end{array}
$$

Examples

Example

Squaring a number (Second version):

$\texttt{square}(n)$:			
$r := 0$			$O(1)$
for $i = 1$ to n :	$O(1)$	n times	$O(n)$
$r := r + n$	$O(1)$		
return r			

Examples

Example

Squaring a number (Second version):

$$
\begin{array}{lllll}
\texttt{square}(n): & & & & \\
\quad r := 0 & & & & O(1) \\
\quad \text{for } i = 1 \text{ to } n: & O(1) & \Big| & n \text{ times} & O(n) \\
\quad\quad r := r + n & O(1) & \Big| & & \\
\quad \text{return } r & & & & O(1)
\end{array}
$$

Examples

Example

Squaring a number (Second version):

$$
\begin{array}{llll}
\texttt{square}(n): & & & \\
\quad r := 0 & & & O(1) \\
\quad \text{for } i = 1 \text{ to } n: & O(1) & \left| \; n \text{ times} \right. & O(n) \\
\quad\quad r := r + n & O(1) & & \\
\quad \text{return } r & & & O(1)
\end{array}
$$

Running time: $O(1) + O(n) + O(1) = O(n)$

Examples

Example

Cubing a number (using second squaring program):

$\texttt{cube}(n)$:
$\quad r := 0$
$\quad$ for $i = 1$ to n :
$\qquad r := r + \texttt{square}(n)$
$\quad$ return r

Examples

Example

Cubing a number (using second squaring program):

$\texttt{cube}(n)$:
  $r := 0$ $\qquad$ $O(1)$
  for $i = 1$ to n : $\qquad$ $O(1)$
    $r := r + \texttt{square}(n)$
  return r $\qquad$ $O(1)$

Examples

Example

Cubing a number (using second squaring program):

$\texttt{cube}(n)$:

$r := 0$ $\quad O(1)$

for $i = 1$ to n : $\quad O(1)$

$r := r + \texttt{square}(n)$ $\quad O(1) + O(n)$

return r $\quad O(1)$

Examples

Example

Cubing a number (using second squaring program):

$\texttt{cube}(n)$:			
$r := 0$			$O(1)$
for $i = 1$ to n :	$O(1)$	n times	$O(n^2)$
$r := r + \texttt{square}(n)$	$O(1) + O(n)$		
return r			$O(1)$

Examples

Example

Cubing a number (using second squaring program):

$\texttt{cube}(n)$:			
$r := 0$			$O(1)$
for $i = 1$ to n :	$O(1)$	n times	$O(n^2)$
$r := r + \texttt{square}(n)$	$O(1) + O(n)$		
return r			$O(1)$

Running time: $O(1) + O(n^2) + O(1) = O(n^2)$

Summary of topics

- Motivation
- Standard approach
- Examples
- Simplifying with worst-case and big-O
- Recursive examples

Worst-case and big-O

Worst-case input assumption and big-O combine to *simplify* the analysis:

Example

Sum of squares (Using second squaring program):

$\texttt{sumOfSquares}(n)$:
  $r := 0$
  for $i = 1$ to n :
    $r := r + \texttt{square}(i)$
  return r

Worst-case and big-O

Worst-case input assumption and big-O combine to *simplify* the analysis:

Example

Sum of squares (Using second squaring program):

$\texttt{sumOfSquares}(n)$:
 $r := 0$ $\quad O(1)$
 for $i = 1$ to n : $\quad O(1)$
  $r := r + \texttt{square}(i)$
 return r $\quad O(1)$

Worst-case and big-O

Worst-case input assumption and big-O combine to *simplify* the analysis:

Example

Sum of squares (Using second squaring program):

$\texttt{sumOfSquares}(n)$:		
$r := 0$		$O(1)$
for $i = 1$ to n :	$O(1)$	$O(?)$
$r := r + \texttt{square}(i)$	$O(i)$	
return r		$O(1)$

Worst-case and big-O

Worst-case input assumption and big-O combine to *simplify* the analysis:

Example

Sum of squares (Using second squaring program):

$\texttt{sumOfSquares}(n)$:

$r := 0$ $\quad O(1)$

for $i = 1$ to n : $\quad O(1)$ | n times $\quad O(?)$

$r := r + \texttt{square}(i)$ $\quad O(n)$ |

return r $\quad O(1)$

Worst-case and big-O

Worst-case input assumption and big-O combine to *simplify* the analysis:

Example

Sum of squares (Using second squaring program):

$\texttt{sumOfSquares}(n)$:
$r := 0$ $\qquad O(1)$
for $i = 1$ to n : $\qquad O(1)$ $\quad$ | n times $\qquad O(n^2)$
$r := r + \texttt{square}(i)$ $\qquad O(n)$ $\quad$ |
return r $\qquad O(1)$

Running time: $O(1) + O(n^2) + O(1) = O(n^2)$

Worst-case and big-O

Worst-case input assumption and big-O combine to *simplify* the analysis:

Example

Finding an element (x) in an array (L) of length n:

```
find(x, L) :
  for i = 0 to n − 1 :
    if L[i] == x :
      return i
  return − 1
```

Worst-case and big-O

Worst-case input assumption and big-O combine to *simplify* the analysis:

Example

Finding an element (x) in an array (L) of length n:

$$
\begin{array}{lll}
\texttt{find}(x, L): & & \\
\quad \text{for } i = 0 \text{ to } n-1: & O(1) & \\
\qquad \text{if } L[i] == x: & O(1) & \\
\qquad\quad \text{return } i & O(1) & \\
\quad \text{return } -1 & & O(1)
\end{array}
$$

Worst-case and big-O

Worst-case input assumption and big-O combine to *simplify* the analysis:

Example

Finding an element (x) in an array (L) of length n:

$$
\begin{array}{llll}
\texttt{find}(x, L): & & & \\
\quad \text{for } i = 0 \text{ to } n-1: & O(1) & & \\
\quad\quad \text{if } L[i] == x: & O(1) & \text{? times} & O(?) \\
\quad\quad\quad \text{return } i & O(1) & & \\
\quad \text{return } -1 & & & O(1)
\end{array}
$$

Worst-case and big-O

Worst-case input assumption and big-O combine to *simplify* the analysis:

Example

Finding an element (x) in an array (L) of length n:

$\texttt{find}(x, L)$:			
for $i = 0$ to $n - 1$:	$O(1)$		
if $L[i] == x$:	$O(1)$	$O(n)$ times	$O(?)$
return i	$O(1)$		
return -1			$O(1)$

Worst-case and big-O

Worst-case input assumption and big-O combine to *simplify* the analysis:

Example

Finding an element (x) in an array (L) of length n:

$$
\begin{array}{llll}
\texttt{find}(x, L): & & & \\
\quad \text{for } i = 0 \text{ to } n-1: & O(1) & & \\
\quad\quad \text{if } L[i] == x: & O(1) & O(n) \text{ times} & O(n) \\
\quad\quad\quad \text{return } i & O(1) & & \\
\quad \text{return } -1 & & & O(1)
\end{array}
$$

Running time: $O(n) + O(1) = O(n)$

Worst-case and big-O

Worst-case input assumption and big-O combine to *simplify* the analysis:

NB

Simplifications might lead to sub-optimal bounds, may have to do a better analysis to get best bounds:

- *Finer-grained upper bound analysis*
- *Analyse specific cases to find a matching lower bound (big-Ω)*

NB

Big-Ω is a **lower bound** *analysis of the worst-case; NOT a "best-case" analysis.*

Worst-case and big-O

Analyse specific cases to find a matching lower bound (big-Ω)

Example

Let L_n be an n-element array of 0's.
Finding an element (x) in an array (L) of length n:

$\texttt{find}(x, L)$:
  for $i = 0$ to $n - 1$:
    if $L[i] == x$:
      return i
  return -1

Worst-case and big-O

Analyse specific cases to find a matching lower bound (big-Ω)

Example

Let L_n be an n-element array of 0's.
Finding an element (x) in an array (L) of length n:

```
find(x, L) :
    for i = 0 to n − 1 :        Ω(1)
        if L[i] == x :          Ω(1)
            return i            Ω(1)
    return −1                                      Ω(1)
```

Worst-case and big-O

Analyse specific cases to find a matching lower bound (big-Ω)

Example

Let L_n be an n-element array of 0's.
Finding an element (x) in an array (L) of length n:

$$
\begin{array}{lll}
\texttt{find}(x, L): & & \\
\quad \text{for } i = 0 \text{ to } n-1: & \Omega(1) & \\
\quad\quad \text{if } L[i] == x: & \Omega(1) & \Omega(n) \text{ times} \\
\quad\quad\quad \text{return } i & \Omega(1) & \\
\quad \text{return } -1 & & \Omega(1)
\end{array}
$$

Worst-case and big-O

Analyse specific cases to find a matching lower bound (big-Ω)

Example

Let L_n be an n-element array of 0's.
Finding an element (x) in an array (L) of length n:

$$
\begin{array}{lll}
\texttt{find}(x, L): & & \\
\quad \text{for } i = 0 \text{ to } n-1: & \Omega(1) & \\
\quad\quad \text{if } L[i] == x: & \Omega(1) \;\Big|\; \Omega(n) \text{ times} & \Omega(n) \\
\quad\quad\quad \text{return } i & \Omega(1) & \\
\quad \text{return } -1 & & \Omega(1)
\end{array}
$$

Worst-case and big-O

Analyse specific cases to find a matching lower bound (big-Ω)

Example

Let L_n be an n-element array of 0's.
Finding an element (x) in an array (L) of length n:

$$
\begin{array}{llll}
\texttt{find}(x, L): & & & \\
\quad \text{for } i = 0 \text{ to } n-1: & \Omega(1) & & \\
\quad\quad \text{if } L[i] == x: & \Omega(1) & \Omega(n) \text{ times} & \Omega(n) \\
\quad\quad\quad \text{return } i & \Omega(1) & & \\
\quad \text{return } -1 & & & \Omega(1)
\end{array}
$$

Running time of $\texttt{find}(1, L_n)$: $\Omega(n)$

Worst-case and big-O

Analyse specific cases to find a matching lower bound (big-Ω)

Example

Let L_n be an n-element array of 0's.
Finding an element (x) in an array (L) of length n:

$$
\begin{array}{llll}
\texttt{find}(x, L): & & & \\
\quad \text{for } i = 0 \text{ to } n-1: & \Omega(1) & & \\
\quad\quad \text{if } L[i] == x: & \Omega(1) & \Omega(n) \text{ times} & \Omega(n) \\
\quad\quad\quad \text{return } i & \Omega(1) & & \\
\quad \text{return } -1 & & & \Omega(1)
\end{array}
$$

Running time of $\texttt{find}(1, L_n)$: $\Omega(n)$
Therefore, running time of $\texttt{find}(x, L)$: $\Theta(n)$

Summary of topics

- Motivation
- Standard approach
- Examples
- Simplifying with worst-case and big-O
- Recursive examples

Recursive examples

Example

Factorial:

$$
\begin{array}{l}
\texttt{fact}(n): \\
\quad \text{if } n == 0: \\
\qquad \text{return } 1 \\
\quad \text{else}: \\
\qquad \text{return } n * \texttt{fact}(n-1)
\end{array}
$$

Recursive examples

Example

Factorial:

$\texttt{fact}(n)$:

if $n == 0$: $\quad O(1)$

return 1 $\quad O(1)$

else :

return $n * \texttt{fact}(n-1)$ $\quad O(1) + ?$

Recursive examples

Example

Factorial:

$\texttt{fact}(n)$:
  if $n == 0$: $O(1)$
    return 1 $O(1)$
  else :
    return $n * \texttt{fact}(n-1)$ $O(1) + ?$

Running time for $\texttt{fact}(n)$: $T(n)$

Recursive examples

Example

Factorial:

$$
\begin{array}{ll}
\texttt{fact}(n): & \\
\quad \text{if } n == 0: & O(1) \\
\quad\quad \text{return } 1 & O(1) \\
\quad \text{else}: & \\
\quad\quad \text{return } n * \texttt{fact}(n-1) & O(1) + T(n-1)
\end{array}
$$

Running time for $\texttt{fact}(n)$: $T(n)$

Recursive examples

Example

Factorial:

$$
\begin{array}{ll}
\texttt{fact}(n): & \\
\quad \text{if } n == 0: & O(1) \\
\quad\quad \text{return } 1 & O(1) \\
\quad \text{else}: & \\
\quad\quad \text{return } n * \texttt{fact}(n-1) & O(1) + T(n-1)
\end{array}
$$

Running time for $\texttt{fact}(n)$: $T(n)$, where:

$$
\begin{aligned}
T(0) &\in O(1) + O(1) = O(1) \\
T(n) &= T(n-1) + O(1)
\end{aligned}
$$

Recursive examples

Example

Factorial:

$$\begin{array}{ll}
\texttt{fact}(n): & \\
\quad \text{if } n == 0: & O(1) \\
\quad\quad \text{return } 1 & O(1) \\
\quad \text{else}: & \\
\quad\quad \text{return } n * \texttt{fact}(n-1) & O(1) + T(n-1)
\end{array}$$

Running time for $\texttt{fact}(n)$: $T(n)$, where:

$$\begin{aligned}
T(0) &\in O(1) + O(1) = O(1) \\
T(n) &= T(n-1) + O(1) \\
&\in O(n)
\end{aligned}$$

Recursive examples

Example

Factorial:

$$
\begin{array}{ll}
\texttt{fact}(n): & \\
\quad \text{if } n == 0: & \color{red}{O(1)} \\
\qquad \text{return } 1 & \color{red}{O(1)} \\
\quad \text{else}: & \\
\qquad \text{return } n * \texttt{fact}(n-1) & \color{red}{O(1) + T(n-1)}
\end{array}
$$

Running time for $\texttt{fact}(n)$: $T(n)$, where:

$$
\begin{array}{rcl}
T(0) & \in & O(1) + O(1) = O(1) \\
T(n) & = & T(n-1) + O(1) \\
 & \in & O(n)
\end{array}
$$

Running time: $T(n) \in O(n)$

Recursive examples

Example

Summing elements of a linked list (length n):

```
sum(L) :
  if L.isEmpty() :
    return 0
  else :
    return L.data + sum(L.next)
```

Recursive examples

Example

Summing elements of a linked list (length n):

$\texttt{sum(L)}$:
  if $\texttt{L.isEmpty()}$: $O(1)$
    return 0 $O(1)$
  else :
    return $\texttt{L.data} + \texttt{sum(L.next)}$ $O(1) +$

Recursive examples

Example

Summing elements of a linked list (length n):

$\texttt{sum(L)}$:
    if $\texttt{L.isEmpty()}$: $\quad\quad$ $O(1)$
        return 0 $\quad\quad$ $O(1)$
    else :
        return $\texttt{L.data} + \texttt{sum(L.next)}$ $\quad\quad$ $O(1) +$

Running time for `sum(L)`: $T(n)$

Recursive examples

Example

Summing elements of a linked list (length n):

```
sum(L) :
  if L.isEmpty() :                          O(1)
    return 0                                O(1)
  else :
    return L.data + sum(L.next)             O(1) + T(n − 1)
```

Running time for `sum(L)`: $T(n)$

Recursive examples

Example

Summing elements of a linked list (length n):

```
sum(L) :
  if L.isEmpty() :                              O(1)
    return 0                                    O(1)
  else :
    return L.data + sum(L.next)                 O(1) + T(n − 1)
```

Running time for `sum(L)`: $T(n)$, where:

$$\begin{aligned} T(0) &\in O(1) + O(1) = O(1) \\ T(n) &= T(n-1) + O(1) \end{aligned}$$

Recursive examples

Example

Summing elements of a linked list (length n):

```
sum(L) :
  if L.isEmpty() :                         O(1)
    return 0                               O(1)
  else :
    return L.data + sum(L.next)            O(1) + T(n − 1)
```

Running time for `sum(L)`: $T(n)$, where:

$$\begin{aligned} T(0) &\in O(1) + O(1) = O(1) \\ T(n) &= T(n-1) + O(1) \\ &\in O(n) \end{aligned}$$

Recursive examples

Example

Insertion sort (L has n elements):

```
sort(L) :
  if L.isEmpty() :
    return L
  else :
    L2 := sort(L.next)
    insert L.data into L2
    return L2
```

Recursive examples

Example

Insertion sort (L has n elements):

```
sort(L) :
  if L.isEmpty() :          O(1)
    return L                O(1)
  else :
    L2 := sort(L.next)
    insert L.data into L2
    return L2               O(1)
```

Recursive examples

Example

Insertion sort (`L` has n elements):

```
sort(L) :
  if L.isEmpty() :               O(1)
    return L                     O(1)
  else :
    L2 := sort(L.next)
    insert L.data into L2        O(n)
    return L2                    O(1)
```

Recursive examples

Example

Insertion sort (`L` has n elements):

`sort(L)` :	
if `L.isEmpty()` :	$O(1)$
return `L`	$O(1)$
else :	
`L2` := `sort(L.next)`	$T(n-1)$
insert `L.data` into `L2`	$O(n)$
return `L2`	$O(1)$

Running time for `sort(L)`: $T(n)$

Recursive examples

Example

Insertion sort (`L` has n elements):

`sort(L)` :	
if `L.isEmpty()` :	$O(1)$
return `L`	$O(1)$
else :	
`L2` := `sort(L.next)`	$T(n-1)$
insert `L.data` into `L2`	$O(n)$
return `L2`	$O(1)$

Running time for `sort(L)`: $T(n)$, where:

$$
\begin{aligned}
T(0) &\in O(1) + O(1) = O(1) \\
T(n) &= T(n-1) + O(n) + O(1)
\end{aligned}
$$

Recursive examples

Example

Insertion sort (L has n elements):

```
sort(L) :
  if L.isEmpty() :               O(1)
    return L                     O(1)
  else :
    L2 := sort(L.next)           T(n-1)
    insert L.data into L2        O(n)
    return L2                    O(1)
```

Running time for sort(L): $T(n)$, where:

$$\begin{aligned} T(0) &\in O(1) + O(1) = O(1) \\ T(n) &= T(n-1) + O(n) + O(1) \\ &\in O(n^2) \end{aligned}$$

Recursive examples

Example

Euclidean algorithm for $\text{gcd}(m, n)$ $(N = m + n)$:

$$
\begin{array}{l}
\text{gcd}(m, n) : \\
\quad \text{if } m > n : \\
\qquad \text{return } \text{gcd}(m - n, n) \\
\quad \text{else if } n > m : \\
\qquad \text{return } \text{gcd}(m, n - m) \\
\quad \text{else} : \qquad \text{return } m
\end{array}
$$

Recursive examples

Example

Euclidean algorithm for $\mathrm{gcd}(m, n)$ $(N = m + n)$:

$$
\begin{array}{ll}
\mathrm{gcd}(m,n): & \\
\quad \text{if } m > n: & O(1) \\
\quad\quad \text{return } \mathrm{gcd}(m-n, n) & \\
\quad \text{else if } n > m: \qquad O(1) & \\
\quad\quad \text{return } \mathrm{gcd}(m, n-m) & \\
\quad \text{else}: \qquad \text{return } m & O(1)
\end{array}
$$

Recursive examples

Example

Euclidean algorithm for $\gcd(m, n)$ $(N = m + n)$:

$$
\begin{array}{ll}
\gcd(m, n): & \\
\quad \text{if } m > n: & \color{red}{O(1)} \\
\quad\quad \text{return } \gcd(m-n, n) & \\
\quad \text{else if } n > m: \quad \color{red}{O(1)} & \\
\quad\quad \text{return } \gcd(m, n-m) & \\
\quad \text{else}: \quad \text{return } m & \color{red}{O(1)}
\end{array}
$$

Running time for $\gcd(m, n)$: $T(N)$

Recursive examples

Example

Euclidean algorithm for $\text{gcd}(m, n)$ ($N = m + n$):

$$
\begin{array}{lll}
\text{gcd}(m,n): & & \\
\quad \text{if } m > n: & & O(1) \\
\qquad \text{return } \text{gcd}(m-n, n) & & \leq T(N-1) \\
\quad \text{else if } n > m: & O(1) & \\
\qquad \text{return } \text{gcd}(m, n-m) & & \leq T(N-1) \\
\quad \text{else}: \quad \text{return } m & & O(1)
\end{array}
$$

Running time for $\text{gcd}(m, n)$: $T(N)$

Recursive examples

Example

Euclidean algorithm for $\gcd(m, n)$ ($N = m + n$):

$$
\begin{array}{lr}
\gcd(m, n): & \\
\quad \text{if } m > n: & O(1) \\
\quad\quad \text{return } \gcd(m - n, n) & \leq T(N - 1) \\
\quad \text{else if } n > m: \qquad O(1) & \\
\quad\quad \text{return } \gcd(m, n - m) & \leq T(N - 1) \\
\quad \text{else}: \qquad \text{return } m & O(1)
\end{array}
$$

Running time for $\gcd(m, n)$: $T(N)$, where:

$$
\begin{aligned}
T(1) &\in O(1) \\
T(N) &\leq T(N - 1) + O(1)
\end{aligned}
$$

Recursive examples

Example

Euclidean algorithm for $\gcd(m, n)$ $(N = m + n)$:

$$
\begin{array}{ll}
\gcd(m, n): & \\
\quad \text{if } m > n: & O(1) \\
\qquad \text{return } \gcd(m - n, n) & \leq T(N - 1) \\
\quad \text{else if } n > m: \qquad O(1) & \\
\qquad \text{return } \gcd(m, n - m) & \leq T(N - 1) \\
\quad \text{else}: \quad \text{return } m & O(1)
\end{array}
$$

Running time for $\gcd(m, n)$: $T(N)$, where:

$$
\begin{array}{rcl}
T(1) & \in & O(1) \\
T(N) & \leq & T(N - 1) + O(1) \\
 & \in & O(N)
\end{array}
$$

Recursive examples

Example

Euclidean algorithm for $\mathrm{gcd}(m, n)$ ($N = m + n$):

Running time: $O(N)$

NB

N is not the input **size**. *Input size is* $\log(m) + \log(n)$

Recursive examples

Example

Faster Euclidean algorithm for $\mathrm{gcd}(m, n)$ $(N = m + n)$:

$\mathrm{gcd}(m, n)$:
  if $m > n > 0$:
    return $\mathrm{gcd}(m \,\%\, n, n)$
  else if $n > m > 0$:
    return $\mathrm{gcd}(m, n \,\%\, m)$
  else :   return $\max(m, n)$

Recursive examples

Example

Faster Euclidean algorithm for $\gcd(m, n)$ $(N = m + n)$:

$\gcd(m, n)$:
  if $m > n > 0$: $O(1)$
    return $\gcd(m \,\%\, n, n)$
  else if $n > m > 0$: $O(1)$
    return $\gcd(m, n \,\%\, m)$
  else : return $\max(m, n)$ $O(1)$

Recursive examples

Example

Faster Euclidean algorithm for $\text{gcd}(m, n)$ $(N = m + n)$:

$$
\begin{array}{ll}
\text{gcd}(m, n): & \\
\quad \text{if } m > n > 0: & {\color{red}O(1)} \\
\quad\quad \text{return } \text{gcd}(m \,\%\, n, n) & \\
\quad \text{else if } n > m > 0: \qquad {\color{red}O(1)} & \\
\quad\quad \text{return } \text{gcd}(m, n \,\%\, m) & \\
\quad \text{else}: \qquad \text{return } \max(m, n) & {\color{red}O(1)}
\end{array}
$$

Running time for $\text{gcd}(m, n)$: $T(N)$

Recursive examples

Example

Faster Euclidean algorithm for $\gcd(m, n)$ $(N = m + n)$:

$$
\begin{array}{ll}
\gcd(m, n): & \\
\quad \text{if } m > n > 0: & O(1) \\
\quad\quad \text{return } \gcd(m \% n, n) & \leq T(N/1.5) \\
\quad \text{else if } n > m > 0: \qquad O(1) & \\
\quad\quad \text{return } \gcd(m, n \% m) & \leq T(N/1.5) \\
\quad \text{else}: \quad \text{return } \max(m, n) & O(1)
\end{array}
$$

Running time for $\gcd(m, n)$: $T(N)$

Recursive examples

Example

Faster Euclidean algorithm for $\gcd(m, n)$ $(N = m + n)$:

$$
\begin{array}{lll}
\gcd(m, n): & & \\
\quad \text{if } m > n > 0: & & O(1) \\
\quad\quad \text{return } \gcd(m \,\%\, n, n) & & \leq T(N/1.5) \\
\quad \text{else if } n > m > 0: & O(1) & \\
\quad\quad \text{return } \gcd(m, n \,\%\, m) & & \leq T(N/1.5) \\
\quad \text{else}: \quad \text{return } \max(m, n) & & O(1)
\end{array}
$$

Running time for $\gcd(m, n)$: $T(N)$, where:

Recursive examples

Example

Faster Euclidean algorithm for $\mathrm{gcd}(m,n)$ ($N = m + n$):

$$
\begin{array}{llr}
\mathrm{gcd}(m,n): & & \\
\quad \text{if } m > n > 0: & & O(1) \\
\quad\quad \text{return } \mathrm{gcd}(m \,\%\, n, n) & & \leq T(N/1.5) \\
\quad \text{else if } n > m > 0: & O(1) & \\
\quad\quad \text{return } \mathrm{gcd}(m, n \,\%\, m) & & \leq T(N/1.5) \\
\quad \text{else}: \quad \text{return } \max(m,n) & & O(1)
\end{array}
$$

Running time for $\mathrm{gcd}(m,n)$: $T(N)$, where:

$$
\begin{aligned}
T(1) &\in O(1) \\
T(N) &\leq T(N/1.5) + O(1)
\end{aligned}
$$

Recursive examples

Example

Faster Euclidean algorithm for $\text{gcd}(m, n)$ $(N = m + n)$:

$$
\begin{array}{lr}
\text{gcd}(m, n): & \\
\quad \text{if } m > n > 0: & O(1) \\
\quad\quad \text{return } \text{gcd}(m \,\%\, n, n) & \leq T(N/1.5) \\
\quad \text{else if } n > m > 0: \qquad O(1) & \\
\quad\quad \text{return } \text{gcd}(m, n \,\%\, m) & \leq T(N/1.5) \\
\quad \text{else}: \qquad \text{return } \max(m, n) & O(1)
\end{array}
$$

Running time for $\text{gcd}(m, n)$: $T(N)$, where:

$$
\begin{aligned}
T(1) &\in O(1) \\
T(N) &\leq T(N/1.5) + O(1) \\
&\in O(\log N)
\end{aligned}
$$

Recursive examples

Example

Faster Euclidean algorithm for $\gcd(m, n)$ ($N = m + n$):

What about lower bounds?

Recursive examples

Example

Faster Euclidean algorithm for $\gcd(m, n)$ ($N = m + n$):

What about lower bounds?

- Can show algorithm takes k steps to compute $\gcd(F_k, F_{k-1})$ where F_k is the k-th Fibonacci number
- Can show $1.5^k \leq F_k \leq 2^k$, so $k \in \Theta(\log F_k)$
- Therefore $\gcd(F_k, F_{k-1}) \in \Omega(\log(F_k + F_{k-1}))$

Exercise

Exercise

$\boxed{4.3.22}$ The following algorithm raises a number a to a power n.

$$
\begin{array}{l}
\text{exp}(a, n): \\
\quad p = 1 \\
\quad i = n \\
\quad \text{while } i > 0: \\
\quad\quad p = p * a \\
\quad\quad i = i - 1 \\
\quad \text{return } p
\end{array}
$$

Determine the running time of this algorithm.

Exercise

Exercise

$\boxed{4.3.21}$ The following algorithm gives a fast method for raising a number a to a power n.

$$
\begin{array}{l}
\texttt{fast-exp}(a, n): \\
\quad p = 1 \\
\quad q = a \\
\quad i = n \\
\quad \text{while } i > 0: \\
\qquad \text{if } i \text{ is odd}: \\
\qquad\quad p = p * q \\
\qquad q = q * q \\
\qquad i = \left\lfloor \frac{i}{2} \right\rfloor \\
\quad \text{return } p
\end{array}
$$

Determine the running time of this algorithm.