Complexity of Algorithms

Bengt J. Nilsson

Malmö University, Sweden

How many operations does the following code perform?

A is an array with n elements, *element* is a variable with a *key* item

for $i \leftarrow 0$ **to** $|A| - 1$ **do**
  if $element.key = A[i].key$ **then return** $A[i]$ **endif**
endfor

How many unit operations does the following code perform?

A is an array with n elements, *element* is a variable with a *key* item

```
for i ← 0 to |A| − 1 do
    if element.key = A[i].key then return A[i] endif
endfor
```

What is a unit operation?

How many operations does the following code perform?

A is an array with n elements, *element* is a variable with a *key* item

> **for** $i \leftarrow 0$ **to** $|A| - 1$ **do**
>   **if** $element.key = A[i].key$ **then return** $A[i]$ **endif**
> **endfor**

What is a unit operation? Any operation that (roughly) corresponds to one machine instruction.

How many operations does the following code perform?

A is an array with n elements, *element* is a variable with a *key* item

for $i \leftarrow 0$ **to** $|A| - 1$ **do**
  if $element.key = A[i].key$ **then return** $A[i]$ **endif**
endfor

What is a unit operation? Any operation that (roughly) corresponds to one machine instruction.

for statement: value accesses, increment, subtraction, test, jump

$$\leq 5 + 1 + 1 + 1 + 1 = 9 \text{ unit operations}$$

How many operations does the following code perform?

A is an array with n elements, *element* is a variable with a *key* item

> **for** $i \leftarrow 0$ **to** $|A| - 1$ **do**
>
>   **if** $element.key = A[i].key$ **then return** $A[i]$ **endif**
>
> **endfor**

What is a unit operation? Any operation that (roughly) corresponds to one machine instruction.

for statement: value accesses, increment, subtraction, test, jump

$$\leq 5 + 1 + 1 + 1 + 1 = 9 \text{ unit operations}$$

if statement: accesses, indexing, test, accesses, indexing, return

$$\leq 5 + 2 + 1 + 2 + 1 + 1 = 12 \text{ unit operations}$$

How many operations does the following code perform?

A is an array with n elements, *element* is a variable with a *key* item

```
for i ← 0 to |A| − 1 do
    if element.key = A[i].key then return A[i] endif
endfor
```

What is a unit operation? Any operation that (roughly) corresponds to one machine instruction.

for statement: value accesses, increment, subtraction, test, jump
$$\leq 5+1+1+1+1 = 9 \text{ unit operations}$$

if statement: accesses, indexing, test, accesses, indexing, return
$$\leq 5+2+1+2+1+1 = 12 \text{ unit operations}$$

How many times is the **if** statement run?

How many operations does the following code perform?

A is an array with n elements, *element* is a variable with a *key* item

```
for i ← 0 to |A| − 1 do
    if element.key = A[i].key then return A[i] endif
endfor
```

What is a unit operation? Any operation that (roughly) corresponds to one machine instruction.

for statement: value accesses, increment, subtraction, test, jump
$\leq 5 + 1 + 1 + 1 + 1 = 9$ unit operations

if statement: accesses, indexing, test, accesses, indexing, return
$\leq 5 + 2 + 1 + 2 + 1 + 1 = 12$ unit operations

How many times is the **if** statement run?
Assume worst case, equality never occurs $\Rightarrow$ n times

How many operations does the following code perform?

A is an array with n elements, *element* is a variable with a *key* item

for $i \leftarrow 0$ **to** $|A| - 1$ **do**
  if $element.key = A[i].key$ **then return** $A[i]$ **endif**
endfor

What is a unit operation? Any operation that (roughly) corresponds to one machine instruction.

for statement: value accesses, increment, subtraction, test, jump
$$\leq 5 + 1 + 1 + 1 + 1 = 9 \text{ unit operations}$$

if statement: accesses, indexing, test, accesses, indexing, return
$$\leq 5 + 2 + 1 + 2 + 1 + 1 = 12 \text{ unit operations}$$

How many times is the **if** statement run?
Assume worst case, equality never occurs $\Rightarrow$ n times

At most $(9 + 12) \cdot n = 21n$ unit operations are performed

Why do we care about this?

BTW: the code does *linear search* through an (unordered) array

Why do we care about this?

BTW: the code does *linear search* through an (unordered) array

We want to *estimate* the running time of our algorithms to ensure that they are as fast as possible. $\Rightarrow$ code is efficient!

Why do we care about this?

BTW: the code does *linear search* through an (unordered) array

We want to *estimate* the running time of our algorithms to ensure that they are as fast as possible. $\Rightarrow$ code is efficient!

Assumptions:
Each unit operation takes the same time on a machine!

Why do we care about this?

BTW: the code does *linear search* through an (unordered) array

We want to *estimate* the running time of our algorithms to ensure that they are as fast as possible. $\Rightarrow$ code is efficient!

Assumptions:
Each unit operation takes the same time on a machine!
Not True, but reasonable approximation to be useful!

Speed of unit operations differs between machines (might not even have the same machine instructions).

Why do we care about this?

BTW: the code does *linear search* through an (unordered) array

We want to *estimate* the running time of our algorithms to ensure that they are as fast as possible. $\Rightarrow$ code is efficient!

Assumptions:
Each unit operation takes the same time on a machine!
Not True, but reasonable approximation to be useful!

Speed of unit operations differs between machines (might not even have the same machine instructions).

Imagine two machines, one with unit operation speed 50μsec, one with unit operation speed 1μsec
n varies from 100 elements to 1 000 000 000. Running time is:

n	machine 1	machine 2	ratio
100	105ms	2ms	50
1 000	1s	21ms	50
100 000	105s	2s	50
100 000 000	$\approx$ 30h	$\approx$ 35m	50
1 000 000 000	$\approx$ 12d	$\approx$ 6h	50

How about this code?

A is a 2D array with $n \times n$ integers,

```
for m ← 0 to n − 1 do
    A[m, 0] ← 1, A[m, m] ← 1
    for k ← 1 to m − 1 do
        A[m, k] ← A[m − 1, k] + A[m − 1, k − 1]
    endfor
endfor
```

Unit operations:

First **for** statement: value accesses, increment, subtraction, test, jump
$\leq 4 + 1 + 1 + 1 + 1 = 8$ unit operations

Second statement: value accesses, indexing, assignment
$\leq 6 + 4 + 2 = 12$ unit operations

Next **for** statement: value accesses, increment, subtraction, test, jump
$\leq 4 + 1 + 1 + 1 + 1 = 8$ unit operations

Innermost statement: value accesses, indexing, addition, assignment
$\leq 12 + 6 + 1 + 1 = 20$ unit operations

Total number of unit operations:

$$T(n) \leq \sum_{m=0}^{n-1} \Big((8+12)+(8+20)(m-1)\Big) = \sum_{m=0}^{n-1} 28m-8 = 14n(n-1)-8n = 14n^2 - 22n$$

Running times for different n

BTW: the code computes the number of combinations you can get by picking k items out of m, for all $0 \leq k \leq m \leq n$.

What is the run time estimate on our two machines for different n? (50μsec/op and 1μsec/op)

n	machine 1	machine 2	ratio
100	7s	0.14s	50
1 000	$\approx$ 12m	$\approx$ 14s	50
100 000	$\approx$ 81d	$\approx$ 39h	50
100 000 000	$\approx$ 222 000y	$\approx$ 4 440y	50
1 000 000 000	$\approx$ 22 200 000y	$\approx$ 444 000y	50

Seems it is good to know these estimates as n increases!

Running times for different n

BTW: the code computes the number of combinations you can get by picking k items out of m, for all $0 \leq k \leq m \leq n$.

What is the run time estimate on our two machines for different n? (50μsec/op and 1μsec/op)

n	machine 1	machine 2	ratio
100	7s	0.14s	50
1 000	$\approx$ 12m	$\approx$ 14s	50
100 000	$\approx$ 81d	$\approx$ 39h	50
100 000 000	$\approx$ 222 000y	$\approx$ 4 440y	50
1 000 000 000	$\approx$ 22 200 000y	$\approx$ 444 000y	50

Seems it is good to know these estimates as n increases!

However, these are cumbersome calculations to do.

Can we not simplify?

Complexity Analysis and O-notation

Remembering runtime functions like $21n$ and $14n^2 - 22n$ is not easy!

What if I made a mistake when estimating number of unit operations on a line?
Maybe the functions are actually $25n$ and $36n^2 - 2n$, respectively!

The constants 21, 14, -22, etc, seem to be offset by the speed of the machine operations. We don't care about constants! We also don't care about lower order terms!

We would like to just say that running time first algorithm grows *proportionally to* n, n being the number of items in the array, and the second one
grows *proportionally to* n^2, where n is the maximum number of possible items to choose from.

Complexity Analysis and O-notation

Objective: a simple way to provide information about magnitudes of functions.

These can represent running times based on input size or other parameter.

Complexity Analysis and O-notation

Objective: a simple way to provide information about magnitudes of functions.

These can represent running times based on input size or other parameter.

Definition: $O(g(n))$ is the set of all functions $f(n)$ such that

$$\lim_{n\to\infty} \left| \frac{f(n)}{g(n)} \right| \leq c,$$

where c is any fixed finite constant (c does not dependent on n).

$f(n)$ is at most of the order of magnitude of $g(n)$.

Examples

$21n \in O(n)$, since $\lim_{n \to \infty} \left| \frac{21n}{n} \right| = 21$ fulfills the requirement.

Examples

$21n \in O(n)$, since $\lim_{n\to\infty} \left|\frac{21n}{n}\right| = 21$ fulfills the requirement.

$14n^2 - 22n \in O(n^2)$, since $\lim_{n\to\infty} \left|\frac{14n^2 - 22n}{n^2}\right| \leq \lim_{n\to\infty} \frac{14n^2}{n^2} = 14$ fulfills the requirement.

Examples

$21n \in O(n)$, since $\lim_{n\to\infty} \left|\frac{21n}{n}\right| = 21$ fulfills the requirement.

$14n^2 - 22n \in O(n^2)$, since $\lim_{n\to\infty} \left|\frac{14n^2 - 22n}{n^2}\right| \leq \lim_{n\to\infty} \frac{14n^2}{n^2} = 14$ fulfills the requirement.

$21n \in O(n^2)$, since $\lim_{n\to\infty} \left|\frac{21n}{n^2}\right| = \lim_{n\to\infty} \frac{21}{n} \leq 21$ fulfills the requirement.

Examples

$21n \in O(n)$, since $\lim_{n\to\infty} \left|\frac{21n}{n}\right| = 21$ fulfills the requirement.

$14n^2 - 22n \in O(n^2)$, since $\lim_{n\to\infty} \left|\frac{14n^2 - 22n}{n^2}\right| \leq \lim_{n\to\infty} \frac{14n^2}{n^2} = 14$ fulfills the requirement.

$21n \in O(n^2)$,since $\lim_{n\to\infty} \left|\frac{21n}{n^2}\right| = \lim_{n\to\infty} \frac{21}{n} \leq 21$ fulfills the requirement.

$14n^2 - 22n \notin O(n)$, since

$$\lim_{n\to\infty} \left|\frac{14n^2 - 22n}{n}\right| \geq \lim_{n\to\infty} \frac{13n^2 + n^2 - 22n}{n} \geq \lim_{n\to\infty} \frac{13n^2}{n}$$
$$= \lim_{n\to\infty} 13n = \infty$$

does not fulfill the requirement.

More Examples

$n \in O(21n)$, since $\lim_{n\to\infty} \left| \frac{n}{21n} \right| = \frac{1}{21}$ fulfills the requirement.

More Examples

$n \in O(21n)$, since $\lim_{n\to\infty} \left|\frac{n}{21n}\right| = \frac{1}{21}$ fulfills the requirement.

$n^2 \in O(14n^2 - 22n)$, since

$$\lim_{n\to\infty} \left|\frac{n^2}{14n^2 - 22n}\right| = \lim_{n\to\infty} \frac{n^2}{13n^2 + n^2 - 22n} \leq \lim_{n\to\infty} \frac{n^2}{13n^2} = \frac{1}{13}$$

fulfills the requirement.

More Examples

$n \in O(21n)$, since $\lim_{n\to\infty} \left|\frac{n}{21n}\right| = \frac{1}{21}$ fulfills the requirement.

$n^2 \in O(14n^2 - 22n)$, since

$$\lim_{n\to\infty} \left|\frac{n^2}{14n^2 - 22n}\right| = \lim_{n\to\infty} \frac{n^2}{13n^2 + n^2 - 22n} \leq \lim_{n\to\infty} \frac{n^2}{13n^2} = \frac{1}{13}$$

fulfills the requirement.

Any fixed finite constant $c \in O(1)$, since $\lim_{n\to\infty} \left|\frac{c}{1}\right| = |c|$ fulfills the requirement.

A Useful Result

$\sum_{k=0}^{p} a_k \cdot n^k \in O(n^p)$, where a_k, $0 \leq k \leq p$, are constants, since

$$\lim_{n\to\infty} \left| \frac{\sum_{k=0}^{p} a_k n^k}{n^p} \right| \leq \lim_{n\to\infty} \frac{\sum_{k=0}^{p} |a_k| n^k}{n^p} \leq \lim_{n\to\infty} \frac{\sum_{k=0}^{p} |a_k| n^p}{n^p} = \sum_{k=0}^{p} |a_k|$$

fulfills the requirement.

Quick estimations of Ordo *for polynomial functions*:

1. remove all lower order terms,
2. remove constant factors.

More Notation

Definition: $\Omega\big(g(n)\big)$ is the set of all functions $f(n)$ such that

$$\lim_{n\to\infty}\left|\frac{g(n)}{f(n)}\right| \le c,$$

where c is any fixed finite constant (c does not dependent on n).

Implies: $f(n) \in \Omega\big(g(n)\big)$ if and only if $g(n) \in O\big(f(n)\big)$.

$f(n)$ is at least of the order of magnitude of $g(n)$.

Definition: $\Theta\big(g(n)\big)$ is the set of all functions $f(n)$ such that

$$f(n) \in O\big(g(n)\big) \text{ and } f(n) \in \Omega\big(g(n)\big).$$

The two functions are of the same magnitude.

Let us look at this code again!

A is a 2D array with $n \times n$ integers,

```
for m ← 0 to n − 1 do
    A[m, 0] ← 1, A[m, m] ← 1
    for k ← 1 to m − 1 do
        A[m, k] ← A[m − 1, k] + A[m − 1, k − 1]
    endfor
endfor
```

Unit operations:

First **for** statement: $O(1)$ unit operations

Second statement: $O(1)$ unit operations

Next **for** statement: $O(1)$ unit operations

Innermost statement: $O(1)$ unit operations **Much easier, right!**

Total number of unit operations:
inner loop is performed $m - 1$ times, outer loop is performed n times

$$T(n) \in \sum_{m=0}^{n-1} O(m) = O\left(\sum_{m=0}^{n-1} m\right) = O\big(n(n-1)/2\big) = O(n^2)$$

How about this code?

A is an array of n values

```
Algorithm findMin
Input: An array A, lower index i, upper index j
if i = j then
    return A[i]
else
    return min { findMin(A, i, (i+j)//2), findMin(A, (i+j)//2+1, j) }
endif
End findMin
```

The algorithm returns the smallest value in A between indices i and j.
External call: $\mathit{findMin}(A, 0, n-1)$

How about this code?

A is an array of n values

Algorithm *findMin*
Input: An array A, lower index i, upper index j

if $i = j$ **then**
  return $A[i]$
else
  return $\min\left\{ \mathit{findMin}\big(A, i, (i+j) /\!\!/ 2\big), \mathit{findMin}\big(A, (i+j) /\!\!/ 2+1, j\big) \right\}$
endif
End *findMin*

The algorithm returns the smallest value in A between indices i and j.
External call: $\mathit{findMin}(A, 0, n-1)$

Complexity in terms of n, the size of the interval in A becomes a recurrence:

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ 2 \cdot T(n/2) + O(1) & \text{if } n > 1 \end{cases}$$

How do we solve this?

Solving Recurrence Equations

The Master Theorem *If* $T(n) = a \cdot T(n/b) + O(n^d)$ $\big(T(1) \in O(1)$ *is always assumed*$\big)$, *for constants* $a > 0$, $b > 1$, *and* $d \geq 0$, *then*

$$T(n) \in \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a \end{cases}$$

Solving Recurrence Equations

The Master Theorem *If* $T(n) = a \cdot T(n/b) + O(n^d)$ $\big(T(1) \in O(1)$ *is always assumed*$\big)$, *for constants* $a > 0$, $b > 1$, *and* $d \geq 0$, *then*

$$T(n) \in \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a \end{cases}$$

In our case:

$$T(n) = 2 \cdot T(n/2) + O(1)$$

so $a = 2$, $b = 2$, and $d = 0$. $O(1) = O(n^0)$
$d = 0 < 1 = \log_2 2$, hence the third case applies

$$T(n) \in O(n^{\log_2 2}) = O(n^1) = O(n)$$

Thank you for your attention.

Questions? Comments?