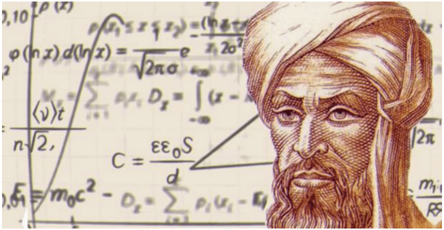

THE UNIVERSITY OF
AUCKLAND
Te Whare Wānanga o Tāmaki Makaurau
NEW ZEALAND

Algorithms and Data Structures

Lecture 3 Estimating Running Time

Jiamou Liu
The University of Auckland



Recap

Recall

The running time of an algorithm `algo` running on input `inp` is defined as T(`inp`) which is the number of elementary operations used when `inp` is fed into `algo`.

Question. Given the description of an algorithm (say, in pseudocode), how can we estimate its running time?

Case 1: No Loop

Example 1. Algorithm **SWAP**: Swapping two elements in an array

Require: $0 \le i \le j \le n-1$

function SWAP(array $a[0..n-1]$, integer i, integer j)

$t \leftarrow a[i]$

$a[i] \leftarrow a[j]$

$a[j] \leftarrow t$

return a

[1] This is strictly speaking not right, but it will not be important (as will be discussed in the future)

Case 1: No Loop

Example 1. Algorithm **SWAP**: Swapping two elements in an array

Require: $0 \le i \le j \le n - 1$

  function SWAP(array $a[0..n-1]$, integer i, integer j)

    $t \leftarrow a[i]$

    $a[i] \leftarrow a[j]$

    $a[j] \leftarrow t$

    return a

Note.

- Input: n-element array $a[0..n-1]$, two indices i, j.
- Input size: n[1].
- **SWAP** is a constant time algorithm. The running time is $f(n) = c$ for some constant c.
- We usually do not care about the exact value of c.

[1] This is strictly speaking not right, but it will not be important (as will be discussed in the future)

Case 1: No Loop

Example 1. Algorithm **SWAP**: Swapping two elements in an array

Require: $0 \le i \le j \le n-1$
  function SWAP(array $a[0..n-1]$, integer i, integer j)
    $t \leftarrow a[i]$
    $a[i] \leftarrow a[j]$
    $a[j] \leftarrow t$
    return a

Note.

- Input: n-element array $a[0..n-1]$, two indices i, j.
- Input size: n[1].
- **SWAP** is a constant time algorithm. The running time is $f(n) = c$ for some constant c.
- We usually do not care about the exact value of c.

Rule 1: A fixed number of statements take constant time.

[1] This is strictly speaking not right, but it will not be important (as will be discussed in the future)

Case 2: A Single Loop

Example 2. Algorithm **FINDMAX**: Finding the maximum in an array

function FINDMAX(array $a[0..n-1]$)
  $k \leftarrow 0$ ▹ location of maximum so far
  for $j \leftarrow 1$ to $n-1$ **do**
    if $a[k] < a[j]$ **then**
      $k = j$
  return k

Case 2: A Single Loop

Example 2. Algorithm **FINDMAX**: Finding the maximum in an array

```
function FINDMAX(array a[0..n − 1])
    k ← 0                          ▹ location of maximum so far
    for j ← 1 to n − 1 do
        if a[k] < a[j] then
            k = j
    return k
```

Note.

- **FINDMAX** contains a **for**-loop, repeating $n - 1$ iterations.
- In each iteration, a fixed number of operations are executed.
- There is a constant number of operations outside of the **for**-loop.
- Thus the running time of the algorithm is $f(n) = c(n - 1) + d$ for some constants c and d. This is a linear function

Case 2: A Single Loop

Example 2. Algorithm **FINDMAX**: Finding the maximum in an array

```
function FINDMAX(array a[0..n − 1])
    k ← 0                                ▹ location of maximum so far
    for j ← 1 to n − 1 do
        if a[k] < a[j] then
            k = j
    return k
```

Note.

- **FINDMAX** contains a **for**-loop, repeating $n - 1$ iterations.
- In each iteration, a fixed number of operations are executed.
- There is a constant number of operations outside of the **for**-loop.
- Thus the running time of the algorithm is $f(n) = c(n - 1) + d$ for some constants c and d. This is a linear function

Rule 2: The running time of a loop multiplies by the number of iterations.

Example 3. Exponential change of variable in loop.

$i \leftarrow 1$
while $i < n$ **do**
    $i \leftarrow 2i$
    print i

Example 3. Exponential change of variable in loop.

$i \leftarrow 1$
while $i < n$ **do**
  $i \leftarrow 2i$
  print i

Note. The algorithm involves a loop.

- What is the print out of the algorithm when $n = 50$?

- How many iterations does the algorithm perform?

Example 3. Exponential change of variable in loop.

$i \leftarrow 1$
while $i < n$ **do**
  $i \leftarrow 2i$
  print i

Note. The algorithm involves a loop.

- What is the print out of the algorithm when $n = 50$?
  Answer. "2 4 8 16 32 64".
- How many iterations does the algorithm perform?

Example 3. Exponential change of variable in loop.

$i \leftarrow 1$
while $i < n$ **do**
    $i \leftarrow 2i$
    print i

Note. The algorithm involves a loop.

- What is the print out of the algorithm when $n = 50$?

  Answer. "2 4 8 16 32 64".

- How many iterations does the algorithm perform?

  Answer. The number of iterations is the number of times we can multiply 2 starting from 1 before reaching n.

  E.g., If $n = 50$, then i grows in $1, 2, 4, 8, 16, 32, 64$. There are 6 iterations.

  In general, the number of iterations is $\lceil \lg n \rceil$, the smallest k such that $2^k > n$.

- Thus the overall running time of the algorithm is $c\lceil \lg n \rceil$.

Example 4.

for $i \leftarrow 1$ to n **do**
    print i

$j \leftarrow 1$
while $j < n$ **do**
    $j \leftarrow 2j$
    print j

Note. The code contains two blocks:

- Block 1 is a **for**-loop with running time $f(n) = cn$ for constant c.
- Block 2 is a **while**-loop with running time $g(n) = d\lceil \lg n \rceil$ for constant d.
- The running time of the entire algorithm is $f(n) + g(n) = cn + d\lceil \lg n \rceil$.

Rule 3. Running time of disjoint blocks adds.

Case 3: Nested Loops

Example 5.

for $i \leftarrow 1$ to n **do**
  for $j \leftarrow 1$ to n **do**
    print $i + j$

Note.

- There is a **for**-loop inside a **for**-loop.
- Each loop executes n iterations.
- In the inner-most iteration, a constant number of elementary operations.
- Therefore the running time is quadratic cn^2 for constant c.

Rule 4. Running time of nested loops with non-interacting variables multiplies.

Example 6.

for $i \leftarrow 1$ to n **do**
    for $j \leftarrow 1$ to n **do**
        for $k \leftarrow 1$ to n **do**
            print $i + j + k$

Example 6.

for $i \leftarrow 1$ to n **do**
  for $j \leftarrow 1$ to n **do**
    for $k \leftarrow 1$ to n **do**
      print $i + j + k$

Note.

- Three nested **for**-loops.
- Each loop executes n iterations.
- In the inner-most iteration, a constant number of elementary operations.
- Therefore the running time is cubic cn^3 for constant c.

Example 6.

for $i \leftarrow 1$ to n **do**
    for $j \leftarrow 1$ to n **do**
        for $k \leftarrow 1$ to n **do**
            print $i + j + k$

Note.

- Three nested **for**-loops.
- Each loop executes n iterations.
- In the inner-most iteration, a constant number of elementary operations.
- Therefore the running time is cubic cn^3 for constant c.

Rule 5. Single, double, and triple loops with fixed number of elementary operations inside the inner loop yield linear, quadratic, and cubic running time.

Example 7. Nested loops with interacting variables.

for $i \leftarrow 1$ to n **do**
  for $j \leftarrow i$ to n **do**
    print (i, j)

Questions.

- What is the output of the algorithm for $n = 3$?
- What is the running time of the inner **for**-loop?
- What is the running time of the algorithm?

Example 7. Nested loops with interacting variables.

for $i \leftarrow 1$ to n **do**
  for $j \leftarrow i$ to n **do**
    print (i, j)

Questions.

- What is the output of the algorithm for $n = 3$?

  Answer. $(1,1),(1,2),(1,3),(2,2),(2,3),(3,3)$

- What is the running time of the inner **for**-loop?

- What is the running time of the algorithm?

Example 7. Nested loops with interacting variables.

for $i \leftarrow 1$ to n **do**
  for $j \leftarrow i$ to n **do**
    print (i, j)

Questions.

- What is the output of the algorithm for $n = 3$?
  Answer. $(1,1),(1,2),(1,3),(2,2),(2,3),(3,3)$
- What is the running time of the inner **for**-loop?
  Answer. $n - i + 1$
- What is the running time of the algorithm?

Example 7. Nested loops with interacting variables.

for $i \leftarrow 1$ to n **do**
  for $j \leftarrow i$ to n **do**
    print (i, j)

Questions.

- What is the output of the algorithm for $n = 3$?

  Answer. $(1, 1), (1, 2), (1, 3), (2, 2), (2, 3), (3, 3)$

- What is the running time of the inner **for**-loop?

  Answer. $n - i + 1$

- What is the running time of the algorithm?

  Answer. $(n - 1 + 1) + (n - 2 + 1) + (n - 3 + 1) + \cdots + (n - n + 1) = 1 + 2 + \cdots + n = n(n + 1)/2$

More Complex Algorithms

Example 8. Snippet: If statements.

for $i = 1; i < n; i \leftarrow 2i$ **do**
  for $j = 1; j < n; j \leftarrow 2j$ **do**
    if $j = 2i$ **then**
      for $k = 0; k < n; k \leftarrow k + 1$ **do**
        { constant number of elementary operations }
    else
      for $k = 1; k < n; k \leftarrow 3k$ **do**
        { constant number of elementary operations }

Note.

- The nested **for**-loops will contribute $(\lceil \lg n \rceil)^2$ towards running time.
- We let m be the number of iterations executed by the **for**-loop, i.e., $m = \lceil \lg n \rceil$ (so $2^m \geq n$).
- For every $i \in \{1, 2, 2^2, \ldots, 2^{m-2}\}$, there is exactly 1 j that satisfies $j = 2i$.
- Therefore, the **if**-statement will go through $m - 1$ times, and will not go through $m^2 - (m - 1)$ times.
- The overall running time is:

$$d\left(\lceil \lg n \rceil^2 - \lceil \lg n \rceil + 1\right)\lceil \log_3 n \rceil + c(\lceil \lg n \rceil - 1)n$$

for constants c and c.

Example 9.

$m \leftarrow 2$
for $j \leftarrow 1$ **to** n **do**
  if $j = m$ **then**
    $m \leftarrow 2m$
    for $i \leftarrow 1$ **to** n **do**
      { constant number of elementary operations }

Note. We roughly estimate the running time.

- Let k be $\lfloor \lg n \rfloor$.
- Since m doubles each time, the **if**-statement will go through only when $j = 2, 4, 8, \ldots, 2^k$ (k times).
- The total running time is thus $ckn = cn\lfloor \lg n \rfloor$ for some constant c.

Example 10.

$m \leftarrow 1$
for $j \leftarrow 1$ **step** $j \leftarrow j + 1$ **to** n **do**
    if $j = m$ **then**
        $m \leftarrow m\,(n - 1)$
        for $i \leftarrow 0$ **step** $i \leftarrow i + 1$ **to** $n - 1$ **do**
            { constant number of elementary operations }

Note.

- The inner **for**-loop will only be executed when $j = 1$ or $j = n - 1$ (assuming $n > 2$).
- Thus the running time of the algorithm is $2c \times n$ (linear) for constant c.

Summary

In this lecture, we study the problem of estimating the running time of an algorithm: *Given an algorithm description, state the running time as a function (defined on the input size)*

Simple Rules

- A fixed number of statements take constant time.
- The running time of a loop multiplies by the number of iterations.
- Running time of disjoint blocks adds.
- Running time of nested loops with non-interacting variables multiplies.

In general, there is no fixed procedure to estimate the running time:

- Simple rules only apply in limited situations.
- Complex scenarios require detailed, ingenious, and sometimes creative ways to analyse.


THE UNIVERSITY OF
AUCKLAND
Te Whare Wānanga o Tāmaki Makaurau
NEW ZEALAND