The RAM Model and Runtime Analysis

COMS10018 - Algorithms

Dr Christian Konrad

What is an Algorithm?

Muhammad ibn Musa **al-Khwarizmi**
$\sim$ 780 - $\sim$ 850
($\approx$ Algorithms)

Algorithms

What is an Algorithm?

- Computational procedure to solve a computational problem

Muhammad ibn Musa **al-Khwarizmi**
$\sim$ 780 - $\sim$ 850
($\approx$ Algorithms)

Algorithms

What is an Algorithm?

- Computational procedure to solve a computational problem
- Mathematical abstraction of a computer programme

Muhammad ibn Musa **al-Khwarizmi**
$\sim$ 780 - $\sim$ 850
($\approx$ Algorithms)

Algorithms

What is an Algorithm?

- Computational procedure to solve a computational problem
- Mathematical abstraction of a computer programme

Discussion Points?

Muhammad ibn Musa **al-Khwarizmi**
$\sim$ 780 - $\sim$ 850
($\approx$ Algorithms)

Algorithms

What is an Algorithm?

- Computational procedure to solve a computational problem
- Mathematical abstraction of a computer programme

Discussion Points?

- Which individual steps can an algorithm do?

Muhammad ibn Musa **al-Khwarizmi**
$\sim$ 780 - $\sim$ 850
($\approx$ Algorithms)

Algorithms

What is an Algorithm?

- Computational procedure to solve a computational problem
- Mathematical abstraction of a computer programme

Discussion Points?

- Which individual steps can an algorithm do?
  Depends on computer, programming language, ...

Muhammad ibn Musa **al-Khwarizmi**
$\sim$ 780 - $\sim$ 850
($\approx$ Algorithms)

Algorithms

What is an Algorithm?

- Computational procedure to solve a computational problem
- Mathematical abstraction of a computer programme

Muhammad ibn Musa **al-Khwarizmi**
$\sim$ 780 - $\sim$ 850
($\approx$ Algorithms)

Discussion Points?

- Which individual steps can an algorithm do?
  Depends on computer, programming language, ...
- How long do these steps take?

Algorithms

What is an Algorithm?

- Computational procedure to solve a computational problem
- Mathematical abstraction of a computer programme

Muhammad ibn Musa **al-Khwarizmi**
~ 780 - ~ 850
($\approx$ Algorithms)

Discussion Points?

- Which individual steps can an algorithm do?
  Depends on computer, programming language, ...
- How long do these steps take?
  Depends on computer, compiler optimization, ...

Models of Computation

Real Computers are complicated

Models of Computation

Real Computers are complicated

Memory hierachy

Models of Computation

Real Computers are complicated

Memory hierachy, floating point operations

Models of Computation

Real Computers are complicated

Memory hierachy, floating point operations, garbage collector

Models of Computation

Real Computers are complicated

Memory hierachy, floating point operations, garbage collector, how long does x^y take?

Models of Computation

Real Computers are complicated
Memory hierachy, floating point operations, garbage collector, how long does x^y take?, compiler optimizations

Models of Computation

Real Computers are complicated

Memory hierachy, floating point operations, garbage collector, how long does x^y take?, compiler optimizations, different programming languages,

Models of Computation

Real Computers are complicated

Memory hierachy, floating point operations, garbage collector, how long does x^y take?, compiler optimizations, different programming languages, ...

Models of Computation

Real Computers are complicated
Memory hierachy, floating point operations, garbage collector, how long does x^y take?, compiler optimizations, different programming languages, ...

Models of Computation:

Models of Computation

Real Computers are complicated
Memory hierachy, floating point operations, garbage collector, how long does x^y take?, compiler optimizations, different programming languages, ...

Models of Computation:

- Simple abstraction of a Computer

Models of Computation

Real Computers are complicated

Memory hierachy, floating point operations, garbage collector, how long does x^y take?, compiler optimizations, different programming languages, . . .

Models of Computation:

- Simple abstraction of a Computer
- Defines the “Rules of the Game”:

Models of Computation

Real Computers are complicated

Memory hierachy, floating point operations, garbage collector, how long does x^y take?, compiler optimizations, different programming languages, ...

Models of Computation:

- Simple abstraction of a Computer
- Defines the "Rules of the Game":
  - Which operations is an algorithm allowed to do?

Models of Computation

Real Computers are complicated

Memory hierachy, floating point operations, garbage collector, how long does x^y take?, compiler optimizations, different programming languages, ...

Models of Computation:

- Simple abstraction of a Computer
- Defines the "Rules of the Game":
  - Which operations is an algorithm allowed to do?
  - What is the cost of each operation?

Models of Computation

Real Computers are complicated

Memory hierachy, floating point operations, garbage collector, how long does x^y take?, compiler optimizations, different programming languages, ...

Models of Computation:

- Simple abstraction of a Computer
- Defines the "Rules of the Game":
  - Which operations is an algorithm allowed to do?
  - What is the cost of each operation?
  - Cost of an algorithm $= \sum$ cost of all its operations

Models of Computation

Real Computers are complicated

Memory hierachy, floating point operations, garbage collector, how long does x^y take?, compiler optimizations, different programming languages, . . .

Models of Computation:

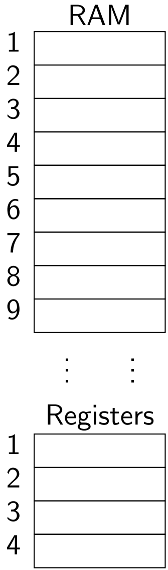
- Simple abstraction of a Computer
- Defines the "Rules of the Game":
  - Which operations is an algorithm allowed to do?
  - What is the cost of each operation?
  - Cost of an algorithm $= \sum$ cost of all its operations

See also:

COMS20007: Programming Languages and Computation

RAM Model

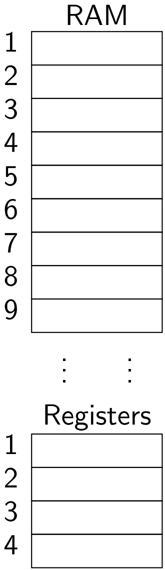
RAM Model: Random Access Machine Model



RAM Model

RAM Model: Random Access Machine Model

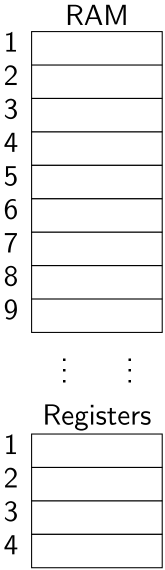
- Infinite Random Access Memory (an array), each cell has a unique address



RAM Model

RAM Model: Random Access Machine Model

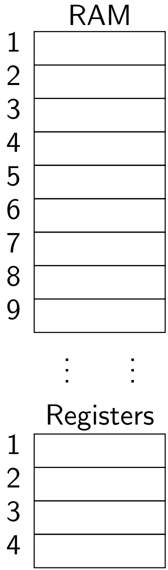
- Infinite Random Access Memory (an array), each cell has a unique address
- Each cell stores one *word*, e.g., an integer, a character, an address, etc.



RAM Model

RAM Model: Random Access Machine Model

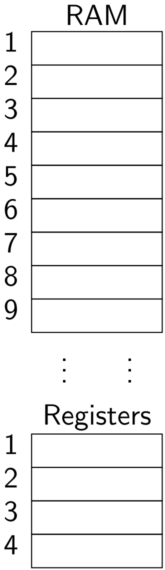
- Infinite Random Access Memory (an array), each cell has a unique address
- Each cell stores one *word*, e.g., an integer, a character, an address, etc.
- **Input:** Stored in RAM



RAM Model

RAM Model: Random Access Machine Model

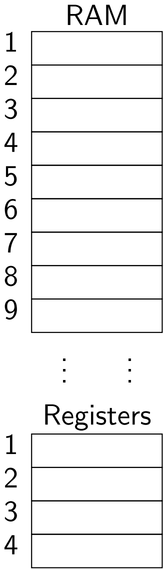
- Infinite Random Access Memory (an array), each cell has a unique address
- Each cell stores one *word*, e.g., an integer, a character, an address, etc.
- **Input:** Stored in RAM
- **Output:** To be written into RAM



RAM Model

RAM Model: Random Access Machine Model

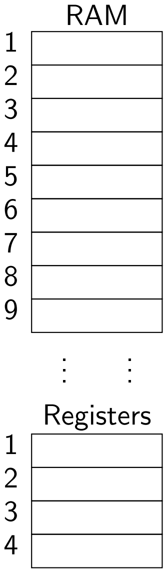
- Infinite Random Access Memory (an array), each cell has a unique address
- Each cell stores one *word*, e.g., an integer, a character, an address, etc.
- **Input:** Stored in RAM
- **Output:** To be written into RAM
- A finite (constant) number of registers (e.g., 4)



RAM Model

RAM Model: Random Access Machine Model

- Infinite Random Access Memory (an array), each cell has a unique address
- Each cell stores one *word*, e.g., an integer, a character, an address, etc.
- **Input:** Stored in RAM
- **Output:** To be written into RAM
- A finite (constant) number of registers (e.g., 4)

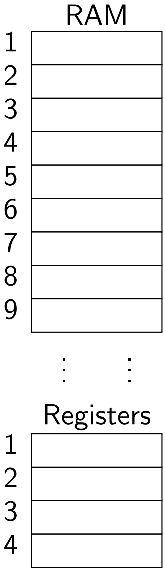


In a single Time Step we can:

RAM Model

RAM Model: Random Access Machine Model

- Infinite Random Access Memory (an array), each cell has a unique address
- Each cell stores one *word*, e.g., an integer, a character, an address, etc.
- **Input:** Stored in RAM
- **Output:** To be written into RAM
- A finite (constant) number of registers (e.g., 4)



In a single Time Step we can:

- Load a word from memory into a register

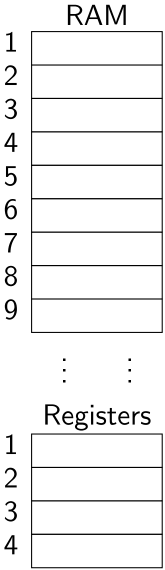
RAM Model

RAM Model: Random Access Machine Model

- Infinite Random Access Memory (an array), each cell has a unique address
- Each cell stores one *word*, e.g., an integer, a character, an address, etc.
- **Input:** Stored in RAM
- **Output:** To be written into RAM
- A finite (constant) number of registers (e.g., 4)

In a single Time Step we can:

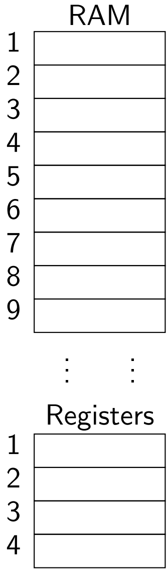
- Load a word from memory into a register
- Compute $(+, -, *, /)$, bit operations, comparisons, etc. on registers



RAM Model

RAM Model: Random Access Machine Model

- Infinite Random Access Memory (an array), each cell has a unique address
- Each cell stores one *word*, e.g., an integer, a character, an address, etc.
- **Input:** Stored in RAM
- **Output:** To be written into RAM
- A finite (constant) number of registers (e.g., 4)



In a single Time Step we can:

- Load a word from memory into a register
- Compute $(+, -, *, /)$, bit operations, comparisons, etc. on registers
- Move a word from register to memory

RAM Model (2)

Algorithm in the RAM Model

RAM Model (2)

Algorithm in the RAM Model

Sequence of elementary operations (similar to assembler code)

RAM Model (2)

Algorithm in the RAM Model
Sequence of elementary operations (similar to assembler code)

Example: Compute the sum of two integers

- Assume that $M[0]$ and $M[1]$ contain the integers
- Write output to position $M[2]$

RAM Model (2)

Algorithm in the RAM Model

Sequence of elementary operations (similar to assembler code)

Example: Compute the sum of two integers

- Assume that $M[0]$ and $M[1]$ contain the integers
- Write output to position $M[2]$

Cost of an Algorithm:

RAM Model (2)

Algorithm in the RAM Model
Sequence of elementary operations (similar to assembler code)

Example: Compute the sum of two integers

- Assume that $M[0]$ and $M[1]$ contain the integers
- Write output to position $M[2]$

Cost of an Algorithm:

- **Runtime:** Total number of elementary operations

RAM Model (2)

Algorithm in the RAM Model
Sequence of elementary operations (similar to assembler code)

Example: Compute the sum of two integers

- Assume that $M[0]$ and $M[1]$ contain the integers
- Write output to position $M[2]$

Cost of an Algorithm:

- **Runtime:** Total number of elementary operations
- **Space:** Total number of memory cells used (excluding the cells that contain the input)

RAM Model (2)

Algorithm in the RAM Model
Sequence of elementary operations (similar to assembler code)

Example: Compute the sum of two integers

- Assume that $M[0]$ and $M[1]$ contain the integers
- Write output to position $M[2]$

Cost of an Algorithm:

- **Runtime:** Total number of elementary operations
- **Space:** Total number of memory cells used (excluding the cells that contain the input)

Assumption:

RAM Model (2)

Algorithm in the RAM Model
Sequence of elementary operations (similar to assembler code)

Example: Compute the sum of two integers

- Assume that $M[0]$ and $M[1]$ contain the integers
- Write output to position $M[2]$

Cost of an Algorithm:

- **Runtime:** Total number of elementary operations
- **Space:** Total number of memory cells used (excluding the cells that contain the input)

Assumption:

- Input for algorithm is stored on read-only cells

RAM Model (2)

Algorithm in the RAM Model
Sequence of elementary operations (similar to assembler code)

Example: Compute the sum of two integers

- Assume that $M[0]$ and $M[1]$ contain the integers
- Write output to position $M[2]$

Cost of an Algorithm:

- **Runtime:** Total number of elementary operations
- **Space:** Total number of memory cells used (excluding the cells that contain the input)

Assumption:

- Input for algorithm is stored on read-only cells
- This space is not accounted for

Specifying an Algorithm

How to specify an Algorithm

- We specify algorithms using pseudo code or English language

Specifying an Algorithm

How to specify an Algorithm

- We specify algorithms using pseudo code or English language
- **We however always bear in mind that every operation of our algorithm can be implemented in $O(1)$ elementary operations in the RAM model**

Specifying an Algorithm

How to specify an Algorithm

- We specify algorithms using pseudo code or English language
- **We however always bear in mind that every operation of our algorithm can be implemented in $O(1)$ elementary operations in the RAM model**
- O-notation gives us the necessary flexibility for a meaningful definition of runtime

Specifying an Algorithm

How to specify an Algorithm

- We specify algorithms using pseudo code or English language
- **We however always bear in mind that every operation of our algorithm can be implemented in $O(1)$ elementary operations in the RAM model**
- O-notation gives us the necessary flexibility for a meaningful definition of runtime

Exercise: How to implement in RAM model?

Require: Array of n integers A
$S \leftarrow 0$
for $i = 0, \ldots, n-1$ **do**
$S \leftarrow S + A[i]$
return S

Notions of Runtime

Runtime on Specific Input
Given a specific input X, what is the number of elementary operations of the algorithm on X?

Notions of Runtime

Runtime on Specific Input
Given a specific input X, what is the number of elementary operations of the algorithm on X?

Worst-case
Consider the set of all inputs of length n. What is the maximum number of elementary operations executed by the algorithm when run on every input of this set?

Notions of Runtime

Runtime on Specific Input
Given a specific input X, what is the number of elementary operations of the algorithm on X?

Worst-case
Consider the set of all inputs of length n. What is the maximum number of elementary operations executed by the algorithm when run on every input of this set?

Best-case
Consider the set of all inputs of length n. What is the minimum number of elementary operations executed by the algorithm when run on every input of this set?

Notions of Runtime

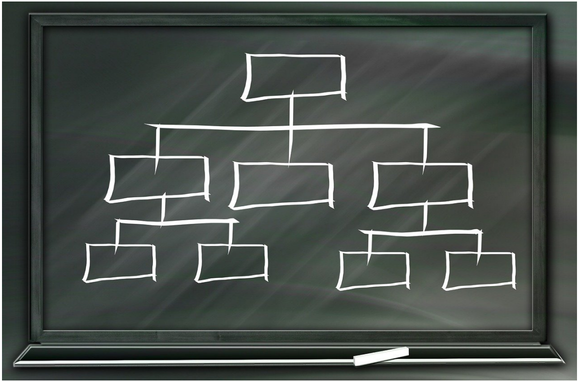
Runtime on Specific Input
Given a specific input X, what is the number of elementary operations of the algorithm on X?

Worst-case
Consider the set of all inputs of length n. What is the maximum number of elementary operations executed by the algorithm when run on every input of this set?

Best-case
Consider the set of all inputs of length n. What is the minimum number of elementary operations executed by the algorithm when run on every input of this set?

Average-case
Consider a set of inputs (e.g. the set of all inputs of length n). What is the average number of elementary operations executed by the algorithm when run on every input of this set?

Runtime Hierachy:



$$\textbf{Best-case} = O(\textbf{Average-case}) = O(\textbf{Worst-case})$$

Runtime/Space Analysis of Algorithms

Runtime

Goals:

Runtime

Goals:

- **Runtime:** Count number of elementary operations when implemented in RAM model

Runtime

Goals:

- **Runtime:** Count number of elementary operations when implemented in RAM model
- **Space:** Count number of cells used when implemented in RAM model

Runtime

Goals:

- **Runtime:** Count number of elementary operations when implemented in RAM model
- **Space:** Count number of cells used when implemented in RAM model

However...

Runtime

Goals:

- **Runtime:** Count number of elementary operations when implemented in RAM model
- **Space:** Count number of cells used when implemented in RAM model

However...

- Algorithms are usually not stated to run in RAM model

Runtime

Goals:

- **Runtime:** Count number of elementary operations when implemented in RAM model
- **Space:** Count number of cells used when implemented in RAM model

However...

- Algorithms are usually not stated to run in RAM model
- We would like to state and analyze our algorithms in pseudo code (or a programming language, natural language, ...)

Runtime

Goals:

- **Runtime:** Count number of elementary operations when implemented in RAM model
- **Space:** Count number of cells used when implemented in RAM model

However...

- Algorithms are usually not stated to run in RAM model
- We would like to state and analyze our algorithms in pseudo code (or a programming language, natural language, ...)

Solution:

Runtime

Goals:

- **Runtime:** Count number of elementary operations when implemented in RAM model
- **Space:** Count number of cells used when implemented in RAM model

However...

- Algorithms are usually not stated to run in RAM model
- We would like to state and analyze our algorithms in pseudo code (or a programming language, natural language, . . .)

Solution:

- Analyze algorithm as specified in pseudo code directly

Runtime

Goals:

- **Runtime:** Count number of elementary operations when implemented in RAM model
- **Space:** Count number of cells used when implemented in RAM model

However...

- Algorithms are usually not stated to run in RAM model
- We would like to state and analyze our algorithms in pseudo code (or a programming language, natural language, . . .)

Solution:

- Analyze algorithm as specified in pseudo code directly
- Make sure that every instruction can be implemented in the RAM model using $O(1)$ elementary operations

Example

Require: Integer array A of length n

```
s ← 0
for i ← 0 ... n − 1 do
    s ← s + A[i]
return s
```

Example

Require: Integer array A of length n

$s \leftarrow 0$
for $i \leftarrow 0 \ldots n-1$ **do**
  $s \leftarrow s + A[i]$
return s

Example

Require: Integer array A of length n

$s \leftarrow 0$ **O(1)**

for $i \leftarrow 0 \dots n-1$ **do**

  $s \leftarrow s + A[i]$

return s

Example

Require: Integer array A of length n

$s \leftarrow 0$ $\mathbf{O(1)}$

for $i \leftarrow 0 \ldots n-1$ **do**

  $s \leftarrow s + A[i]$

return s

Example

Require: Integer array A of length n

$s \leftarrow 0$ $\quad$ $\mathbf{O(1)}$

for $i \leftarrow 0 \ldots n-1$ **do**

$\quad s \leftarrow s + A[i]$ $\quad$ $\mathbf{O(1)}$

return s

Example

Require: Integer array A of length n

```
s ← 0                                   O(1)
for i ← 0 ... n − 1 do
    s ← s + A[i]                        O(1)
return s
```

Example

Require: Integer array A of length n

$s \leftarrow 0$ **O(1)**

for $i \leftarrow 0 \dots n-1$ **do**

  $s \leftarrow s + A[i]$ **O(1)**

return s **O(1)**

Example

Require: Integer array A of length n
  $s \leftarrow 0$ $\mathbf{O(1)}$
  for $i \leftarrow 0 \dots n-1$ **do**
    $s \leftarrow s + A[i]$ $\mathbf{O(1)}$
  return s $\mathbf{O(1)}$

Example

Require: Integer array A of length n

$s \leftarrow 0$	$\mathbf{O(1)}$
for $i \leftarrow 0 \ldots n-1$ **do**	**n times**
$\quad s \leftarrow s + A[i]$	$\mathbf{O(1)}$
return s	$\mathbf{O(1)}$

Example

Require: Integer array A of length n	
$s \leftarrow 0$	$\mathbf{O(1)}$
for $i \leftarrow 0 \ldots n-1$ **do**	**n times**
$s \leftarrow s + A[i]$	$\mathbf{O(1)}$
return s	$\mathbf{O(1)}$

Runtime:

Example

Require: Integer array A of length n	
$s \leftarrow 0$	$\mathbf{O(1)}$
for $i \leftarrow 0 \ldots n-1$ **do**	**n times**
$s \leftarrow s + A[i]$	$\mathbf{O(1)}$
return s	$\mathbf{O(1)}$

Runtime: $O(1) + n \cdot O(1) + O(1) =$

Example

Require: Integer array A of length n

$s \leftarrow 0$	$\mathbf{O(1)}$
for $i \leftarrow 0 \dots n-1$ **do**	**n times**
$s \leftarrow s + A[i]$	$\mathbf{O(1)}$
return s	$\mathbf{O(1)}$

Runtime: $O(1) + n \cdot O(1) + O(1) = O(1) + O(n) + O(1) =$

Example

Require: Integer array A of length n

$s \leftarrow 0$	$\mathbf{O(1)}$
for $i \leftarrow 0 \ldots n-1$ **do**	**n times**
$\quad s \leftarrow s + A[i]$	$\mathbf{O(1)}$
return s	$\mathbf{O(1)}$

Runtime: $O(1) + n \cdot O(1) + O(1) = O(1) + O(n) + O(1) = O(n)$.

Example 2

Require: Integer array A of length n

```
s ← 0
for i ← 0 ... n − 1 do
    for j ← i ... 2i do
        s ← s + A[i]
return s
```

Example 2

Require: Integer array A of length n

$s \leftarrow 0$	$\mathbf{O(1)}$
for $i \leftarrow 0 \ldots n-1$ **do**	
for $j \leftarrow i \ldots 2i$ **do**	
$s \leftarrow s + A[i]$	$\mathbf{O(1)}$
return s	$\mathbf{O(1)}$

Example 2

Require: Integer array A of length n

$s \leftarrow 0$	$\mathbf{O(1)}$
for $i \leftarrow 0 \ldots n-1$ **do**	
for $j \leftarrow i \ldots 2i$ **do**	
$s \leftarrow s + A[i]$	$\mathbf{O(1)}$
return s	$\mathbf{O(1)}$

Example 2

Require: Integer array A of length n

$s \leftarrow 0$	$\mathbf{O(1)}$
for $i \leftarrow 0 \ldots n-1$ **do**	
for $j \leftarrow i \ldots 2i$ **do**	$\mathbf{i + 1}$ **times**
$s \leftarrow s + A[i]$	$\mathbf{O(1)}$
return s	$\mathbf{O(1)}$

Example 2

Require: Integer array A of length n	
$s \leftarrow 0$	$\mathbf{O(1)}$
for $i \leftarrow 0 \ldots n-1$ **do**	
for $j \leftarrow i \ldots 2i$ **do**	$\mathbf{i+1}$ **times**
$s \leftarrow s + A[i]$	$\mathbf{O(1)}$
return s	$\mathbf{O(1)}$

Example 2

Require: Integer array A of length n	
$s \leftarrow 0$	$\mathbf{O(1)}$
for $i \leftarrow 0 \ldots n-1$ **do**	**n times**
for $j \leftarrow i \ldots 2i$ **do**	$\mathbf{i+1}$ **times**
$s \leftarrow s + A[i]$	$\mathbf{O(1)}$
return s	$\mathbf{O(1)}$

Example 2

Require: Integer array A of length n

$s \leftarrow 0$	$\mathbf{O(1)}$
for $i \leftarrow 0 \ldots n-1$ **do**	**n times**
for $j \leftarrow i \ldots 2i$ **do**	$\mathbf{i+1}$ **times**
$s \leftarrow s + A[i]$	$\mathbf{O(1)}$
return s	$\mathbf{O(1)}$

Runtime:

Example 2

Require: Integer array A of length n	
$s \leftarrow 0$	$\mathbf{O(1)}$
for $i \leftarrow 0 \ldots n-1$ **do**	**n times**
for $j \leftarrow i \ldots 2i$ **do**	$\mathbf{i+1}$ **times**
$s \leftarrow s + A[i]$	$\mathbf{O(1)}$
return s	$\mathbf{O(1)}$

Runtime:

$$O(1) + \sum_{i=0}^{n-1} ((i+1) \cdot O(1)) + O(1)$$

Example 2

```
Require: Integer array A of length n
  s ← 0                                O(1)
  for i ← 0 ... n − 1 do               n times
    for j ← i ... 2i do                i + 1 times
      s ← s + A[i]                     O(1)
  return s                             O(1)
```

Runtime:

$$O(1) + \sum_{i=0}^{n-1} ((i+1) \cdot O(1)) + O(1) = O(1) + O(1) \sum_{i=0}^{n-1} (i+1)$$

Example 2

```
Require: Integer array A of length n
  s ← 0                                  O(1)
  for i ← 0 ... n − 1 do                 n times
    for j ← i ... 2i do                  i + 1 times
      s ← s + A[i]                       O(1)
  return s                               O(1)
```

Runtime:

$$O(1) + \sum_{i=0}^{n-1} ((i+1) \cdot O(1)) + O(1) = O(1) + O(1) \sum_{i=0}^{n-1} (i+1)$$

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

Example 2

Require: Integer array A of length n

$s \leftarrow 0$	$\mathbf{O(1)}$
for $i \leftarrow 0 \ldots n-1$ **do**	**n times**
for $j \leftarrow i \ldots 2i$ **do**	$\mathbf{i+1}$ **times**
$s \leftarrow s + A[i]$	$\mathbf{O(1)}$
return s	$\mathbf{O(1)}$

Runtime:

$$O(1) + \sum_{i=0}^{n-1}((i+1) \cdot O(1)) + O(1) = O(1) + O(1)\sum_{i=0}^{n-1}(i+1)$$

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

Example 2

Require: Integer array A of length n	
$s \leftarrow 0$	$\mathbf{O(1)}$
for $i \leftarrow 0 \ldots n-1$ **do**	**n times**
for $j \leftarrow i \ldots 2i$ **do**	$\mathbf{i+1}$ **times**
$s \leftarrow s + A[i]$	$\mathbf{O(1)}$
return s	$\mathbf{O(1)}$

Runtime:

$$O(1) + \sum_{i=0}^{n-1}((i+1) \cdot O(1)) + O(1) = O(1) + O(1)\sum_{i=0}^{n-1}(i+1)$$

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

$$= O(1) + O(\frac{n^2}{2} + \frac{n}{2})$$

Example 2

Require: Integer array A of length n	
$s \leftarrow 0$	$\mathbf{O(1)}$
for $i \leftarrow 0 \dots n-1$ **do**	**n times**
for $j \leftarrow i \dots 2i$ **do**	$\mathbf{i+1}$ **times**
$s \leftarrow s + A[i]$	$\mathbf{O(1)}$
return s	$\mathbf{O(1)}$

Runtime:

$$O(1) + \sum_{i=0}^{n-1} ((i+1) \cdot O(1)) + O(1) = O(1) + O(1) \sum_{i=0}^{n-1} (i+1)$$

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

$$= O(1) + O(\frac{n^2}{2} + \frac{n}{2}) = O(1) + O(n^2)$$

Example 2

Require: Integer array A of length n

$s \leftarrow 0$	$\mathbf{O(1)}$
for $i \leftarrow 0 \ldots n-1$ **do**	**n times**
for $j \leftarrow i \ldots 2i$ **do**	$\mathbf{i+1}$ **times**
$s \leftarrow s + A[i]$	$\mathbf{O(1)}$
return s	$\mathbf{O(1)}$

Runtime:

$$O(1) + \sum_{i=0}^{n-1} ((i+1) \cdot O(1)) + O(1) = O(1) + O(1) \sum_{i=0}^{n-1} (i+1)$$

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

$$= O(1) + O(\frac{n^2}{2} + \frac{n}{2}) = O(1) + O(n^2) = O(n^2) \ .$$

Example 3

Algorithm: Given is an integer array of length n. Run through the array from left to right and maintain the minimum seen so far.

Example 3

Algorithm: Given is an integer array of length n. Run through the array from left to right and maintain the minimum seen so far.

Runtime:

Example 3

Algorithm: Given is an integer array of length n. Run through the array from left to right and maintain the minimum seen so far.

Runtime: $O(n)$