Algorithms and Data Structures I

Pontus Ekberg

Uppsala University

(Based on previous material by Mohamed Faouzi Atig and Parosh Aziz Abdulla)

People to help you!

- Lecturer:
  - Pontus Ekberg, office 105194, pontus.ekberg@it.uu.se
- Teaching Assistants:
  - Sarbojit Das (tutorials, assignments), sarbojit.das@it.uu.se
  - Stephan Spengler (tutorials, assignments), stephan.spengler@it.uu.se
  - Theodora Moldovan (grading)
  - Andreas Pihl (grading)

The course in a nutshell

- 15 lectures (+1 reserve slot)
- 6 tutorials
- 3 assignments
- 1 written exam

The course in a nutshell

- 15 lectures (+1 reserve slot)
  - Not mandatory
- 6 tutorials
- 3 assignments
- 1 written exam

The course in a nutshell

- 15 lectures (+1 reserve slot)
  - Not mandatory
- 6 tutorials
  - Each tutorial is given in three separate (identical) sessions
  - Require signing up to attend!
  - Not mandatory
- 3 assignments
- 1 written exam

The course in a nutshell

- 15 lectures (+1 reserve slot)
  - Not mandatory
- 6 tutorials
  - Each tutorial is given in three separate (identical) sessions
  - Require signing up to attend!
  - Not mandatory
- 3 assignments
  - In groups of 4 students
  - Assess the 1 hp assignments module
- 1 written exam

The course in a nutshell

- 15 lectures (+1 reserve slot)
  - Not mandatory
- 6 tutorials
  - Each tutorial is given in three separate (identical) sessions
  - Require signing up to attend!
  - Not mandatory
- 3 assignments
  - In groups of 4 students
  - Assess the 1 hp assignments module
- 1 written exam
  - Individual, closed-book, written exam
  - Assesses the 4 hp exam module

Assessment

- The assignment part is graded U (fail) or G (pass).

Assessment

- The assignment part is graded U (fail) or G (pass).
  - Each assignment can give up to 10 points.
  - To pass the assignment part, you need at least 21 points in total.

Assessment

- The assignment part is graded U (fail) or G (pass).
    - Each assignment can give up to 10 points.
    - To pass the assignment part, you need at least 21 points in total.
- The exam is graded U (fail), 3, 4 or 5.

Assessment

- The assignment part is graded U (fail) or G (pass).
  - Each assignment can give up to 10 points.
  - To pass the assignment part, you need at least 21 points in total.
- The exam is graded U (fail), 3, 4 or 5.
- When *both* parts are passed, the final course grade will be the same as the exam grade.

Literature

- **Introduction to Algorithms** by Cormen, Leiserson, Rivest and Stein, 3rd edition. MIT Press.
- Lecture slides uploaded to Studium.

Prerequisites

- Background in Programming and Programming Languages.
- Basic understanding in Mathematics, including basic algebra.

Registration

If you haven't yet registered to the course, please do so asap.

Introduction

Algorithms and Data Structures

Algorithms

What is an algorithm?

- An algorithm is a well-defined procedure that takes some input and produces a solution for a particular problem.
- An algorithm is a finite and unambiguous sequence of operations.
- A program is an implementation of an algorithm using a programming language.

Algorithms: A Very Old Science

- Antiquity: Euclide: Compute the greatest common divisor of two numbers, Archimedes: Approximate the value of π
- The word “Algorithm” comes from the name of a 9th century Persian mathematician, Muḥammad ibn Mūsā al-Khwārizmī, later latinized as Algorizmi
- Algorithms are the basis of computers (Software and Hardware)

Greatest Common Divisor

```
GCD(m, n)
x ← m
y ← n
while y > 0
    do tmp ← x
       x ← y
       y ← tmp%y

return x
```

This algorithm computes the greatest common divisor of the two natural numbers m and n

The Traveling Salesperson Problem

- A salesperson has to visit a certain number of cities where each city is visited just once, and the whole trip is a short as possible. Which route should be chosen?

Problem Description

Specification

An abstract description of the behavior of the algorithms (e.g., the computations performed by the algorithm).

Problem Description

Specification

An abstract description of the behavior of the algorithms (e.g., the computations performed by the algorithm).

Sorting

- Input: list $x = (x_1, x_2, \ldots, x_n)$ of natural numbers.
- Output: list $y = (y_1, y_2, \ldots, y_n)$ such that:
  - y is a permutation of x.
  - $y_1 \leq y_2 \leq \cdots \leq y_n$.

Problems and Algorithms

- A problem can have several different algorithmic solutions.

Problems and Algorithms

- A problem can have several different algorithmic solutions.
- Some problems have NO algorithmic solutions (undecidability).

Properties of Algorithms

- Correctness
  - Is the algorithm correct with respect to the specification?
  - Does the algorithm terminate?
- Efficiency
  - How much time and memory the algorithm takes to solve a problem?
  - Is our algorithm optimal (memory and time)?

Algorithms and Data Structures

Data Structures

Data Structures

- A Data structure is a particular way to store and organize data in order to facilitate access, processing and modification.
- No single data structure works well for all purposes.
- Essential ingredient for the algorithm efficiency.
- A Data structure is defined by a set of operations that can be performed on the stored data.

Goal of this course

In this course we study how to:

- Analyze the runtime performance of (simple) algorithms in terms of the size of its inputs, and this in the average, best, and worst cases.
- Choose appropriate algorithms and data structures for storing data, searching and sorting, as well as implement those algorithms.
- Use and manipulate basic graph algorithms.

Preliminaries

Pseudocode Conventions

Pseudocode

We use pseudocode to describe algorithms in order to:

- Describe algorithms in a simple, understandable and concise manner.
- Be independent of a particular programming language.
- Remove the issue of software engineering.
- Be able to use english to describe algorithms.

Example (1)

- Variable types are omitted when it is clear from the context.

```
GCD(m, n)
x ← m
y ← n
while y > 0
      do tmp ← x
         x ← y
         y ← tmp%y

▷ gcd of m and n is stored in x
return x
```

Example (1)

- Variable types are omitted when it is clear from the context.
- Indentation reflects block structure.

```
GCD(m, n)
x ← m
y ← n
while y > 0
      do tmp ← x
         x ← y
         y ← tmp%y

▷ gcd of m and n is stored in x
return x
```

Example (1)

- Variable types are omitted when it is clear from the context.
- Indentation reflects block structure.
- Conditional and iterative constructs such as **if** , **for** , **while** , and **repeat** have their standard interpretations.

GCD(m, n)

$x \leftarrow m$
$y \leftarrow n$
while $y > 0$
    do $tmp \leftarrow x$
        $x \leftarrow y$
        $y \leftarrow tmp\%y$

$\triangleright$ gcd of m and n is stored in x
return x

Example (1)

- Variable types are omitted when it is clear from the context.
- Indentation reflects block structure.
- Conditional and iterative constructs such as **if** , **for** , **while** , and **repeat** have their standard interpretations.
- Symbol ▷ indicates a comment

GCD(m, n)

```
x ← m
y ← n
while y > 0
        do tmp ← x
            x ← y
            y ← tmp%y

▷ gcd of m and n is stored in x
return x
```

Example (1)

- Variable types are omitted when it is clear from the context.
- Indentation reflects block structure.
- Conditional and iterative constructs such as **if** , **for** , **while** , and **repeat** have their standard interpretations.
- Symbol ▷ indicates a comment
- Assignment of the form $x \leftarrow e$ assigns to the variable x the value of the expression e.

```
GCD(m, n)
x ← m
y ← n
while y > 0
      do tmp ← x
         x ← y
         y ← tmp%y

▷ gcd of m and n is stored in x
return x
```

Example (2)

INSERTION-SORT(A)

for $j \leftarrow 2$ **to** $A.\mathit{length}$
  do $\mathit{key} \leftarrow A[j]$
    $\triangleright$ Insert $A[j]$ into $A[1\,..\,j-1]$.
    $i \leftarrow j - 1$
    while $i > 0$ and $A[i] > \mathit{key}$
      do $A[i+1] \leftarrow A[i]$
        $i \leftarrow i - 1$
    $A[i+1] \leftarrow \mathit{key}$

Example (2)

- For an array A:
  - $A[i]$ denotes the ith element of A.
  - $A[i\,..\,j]$ denotes the subarray of elements $A[i], A[i+1], \ldots, A[j]$.
  - The index of the first element of A is 1.

INSERTION-SORT(A)

for $j \leftarrow 2$ **to** $A.\mathit{length}$
    do $\mathit{key} \leftarrow A[j]$
        $\triangleright$ Insert $A[j]$ into $A[1\,..\,j-1]$.
        $i \leftarrow j - 1$
        while $i > 0$ and $A[i] > \mathit{key}$
            do $A[i+1] \leftarrow A[i]$
                $i \leftarrow i - 1$
        $A[i+1] \leftarrow \mathit{key}$

Example (2)

- Compound data are organized into objects, which are composed of attributes. We access to a particular attribute *attr* for an object x by $x.\,attr$

```
INSERTION-SORT(A)
for j ← 2 to A.length
    do key ← A[j]
       ▷ Insert A[j] into A[1 . . j − 1].
       i ← j − 1
       while i > 0 and A[i] > key
           do A[i + 1] ← A[i]
              i ← i − 1
       A[i + 1] ← key
```

Pseudocode: Alternative Command

An alternative command can have one of the following forms:

```
if Condition
    then statement_1
         statement_2
         ...
         statement_n
```

```
if Condition
    then statement_1
         statement_2
         ...
         statement_n
    else statement'_1
         statement'_2
         ...
         statement'_m
```

Pseudocode: Iterative Command

An iterative command can have one of the following forms:

```
while Condition
    do statement_1
       statement_2
       ...
       statement_n
```

```
for i ← 0 to k
    do statement_1
       statement_2
       ...
       statement_n
```

```
for i ← k downto 0
    do statement_1
       statement_2
       ...
       statement_n
```

```
for each element ∈ set
    do statement_1
       statement_2
       ...
       statement_n
```

```
repeat
        statement_1
        statement_2
        ...
        statement_n
    until Condition
```

Mathematical Induction

Proof By Induction

- To prove that a property $P(n)$ holds for all natural numbers n, show the following:
  - Base Case: Show that $P(0)$ holds.
  - Induction Step: Show that $P(n)$ implies $P(n+1)$

    (i.e., assume $P(n)$ holds and show that $P(n+1)$ holds).

Proof By Induction

- To prove that a property $P(n)$ holds for all natural numbers n, show the following:
  - Base Case: Show that $P(0)$ holds. (Or some other base case than 0)
  - Induction Step: Show that $P(n)$ implies $P(n+1)$

    (i.e., assume $P(n)$ holds and show that $P(n+1)$ holds).

Proof By Induction

- To prove that a property $P(n)$ holds for all natural numbers n, show the following:
  - Base Case: Show that $P(0)$ holds. (Or some other base case than 0)
  - Induction Step: Show that $P(n)$ implies $P(n+1)$

    (i.e., assume $P(n)$ holds and show that $P(n+1)$ holds).

    (The assumption that $P(n)$ holds is called the *induction hypothesis*.)

Example

Show that $\sum_{i=0}^{n} i = \frac{n(n+1)}{2}$ for all $n \geq 0$

Variant of Mathematical Induction

- To prove that a property $P(n)$ holds for all natural numbers $n \geq n_0$, show the following:
  - Base Case: Show that $P(n_0)$ holds.
  - Induction Step: Show that $P(n)$ implies $P(n+1)$

    (i.e., assume $P(i)$ holds for all $i \leq n$ and show that $P(n+1)$ holds).

Variant of Mathematical Induction

- To prove that a property $P(n)$ holds for all natural numbers $n \geq n_0$, show the following:
  - Base Case: Show that $P(n_0)$ holds.
  - Induction Step: Show that $P(n)$ implies $P(n+1)$

    (i.e., assume $P(i)$ holds for all $i \leq n$ and show that $P(n+1)$ holds).

    (This variant is called *strong* induction.)

Recurrence Equations

Consider the following equation defined inductively as follows:

$$f(n) = \begin{cases} 1 & \mathit{if}\, n = 0 \\ 2 + f(n-1) & \mathit{if}\, n > 0 \end{cases}$$

Recurrence Equations

Consider the following equation defined inductively as follows:

$$f(n) = \begin{cases} 1 & \textit{if } n = 0 \\ 2 + f(n-1) & \textit{if } n > 0 \end{cases}$$

- Difficult form to work with.

Recurrence Equations

Consider the following equation defined inductively as follows:

$$f(n) = \begin{cases} 1 & \text{if } n = 0 \\ 2 + f(n-1) & \text{if } n > 0 \end{cases}$$

- Difficult form to work with.
- We want a closed form (non-recursive equation), if possible.

Recurrence Equations

Consider the following equation defined inductively as follows:

$$f(n) = \begin{cases} 1 & \textit{if } n = 0 \\ 2 + f(n-1) & \textit{if } n > 0 \end{cases}$$

- Difficult form to work with.
- We want a closed form (non-recursive equation), if possible.
- $f(n)$ has the following equivalent definition:

$$f(n) = 2n + 1$$

Recurrence Equations

Consider the following equation defined inductively as follows:

$$f(n) = \begin{cases} 1 & \mathit{if}\, n = 0 \\ 2 + f(n-1) & \mathit{if}\, n > 0 \end{cases}$$

- Difficult form to work with.
- We want a closed form (non-recursive equation), if possible.
- $f(n)$ has the following equivalent definition:

$$f(n) = 2n + 1 \qquad \qquad \mathit{Much\ \ Simpler}!$$

Recurrence Equations

Consider the following equation defined inductively as follows:

$$f(n) = \begin{cases} 1 & \textit{if}\, n = 0 \\ 2 + f(n-1) & \textit{if}\, n > 0 \end{cases}$$

- Difficult form to work with.
- We want a closed form (non-recursive equation), if possible.
- $f(n)$ has the following equivalent definition:

$$f(n) = 2n + 1 \qquad\qquad \textit{Much Simpler}!$$

How can we construct a closed form and prove the equivalence?

Constructing Closed Forms

There is no general method to solve recurrence equations. The recommended method is:

- Guess a solution
- Prove by induction the correctness of the solution.

Example

Let

$$f(n) = \begin{cases} 1 & \mathit{if}\, n = 0 \\ 2 + f(n-1) & \mathit{if}\, n > 0 \end{cases}$$

Theorem: $f(n) = 2n + 1$, for all $n \geq 0$.

Algorithm Analysis

Time Complexity

- Time it takes for the algorithm to solve a given instance of the problem.
- Depends on:
  - Size of the input: Sorting 100 numbers takes longer than sorting 10 numbers.
  - Value of the input: A given sorting algorithm may even take different amount of time on two inputs of the same size.

Time Complexity

- Time it takes for the algorithm to solve a given instance of the problem.
- Depends on:
  - Size of the input: Sorting 100 numbers takes longer than sorting 10 numbers.
  - Value of the input: A given sorting algorithm may even take different amount of time on two inputs of the same size.
- The same principles can be applied to memory consumption.

Input Size

The input size depends on the problem being studied:

- The number of items in the input such as the size of the array being sorted.
- But could be something else. If multiplying two integers, could be the total number of bits in the two integers.
- Could be described by more than one number. For example, graph algorithm running times are usually expressed in terms of the number of vertices and the number of edges in the input graph.

Measuring the runtime

- Experimentation:
  - Implement the algorithm and compute the runtime for some input.
- Disadvantages:
  - The runtime will depend on the implementation: CPU, OS, language, compiler, . . .
  - On which input data should we run the program?

Algorithm's Computational Complexity

We want a mathematical characterization of the performance :

- Robust with respect to the implementation.
- Gives an estimation of the algorithm runtime in terms of the size of the input.
- Gives some insights on the difficulty of the problem.

Does Complexity Matter?

Recall the Traveling Salesperson Problem (TSP): Assume that we design an algorithm that enumerates all routes and chooses the shortest one.

Does Complexity Matter?

Recall the Traveling Salesperson Problem (TSP): Assume that we design an algorithm that enumerates all routes and chooses the shortest one.

Idea: We could just loop through all the possible routes, check each route's length, and pick the shortest one!

Recall the Traveling Salesperson Problem (TSP): Assume that we design an algorithm that enumerates all routes and chooses the shortest one.

Idea: We could just loop through all the possible routes, check each route's length, and pick the shortest one!

For n cities, there are $n!$ possible routes, where $n! = 1 \times 2 \times \cdots \times (n-1) \times n$.

Does Complexity Matter?

Recall the Traveling Salesperson Problem (TSP): Assume that we design an algorithm that enumerates all routes and chooses the shortest one.

Idea: We could just loop through all the possible routes, check each route's length, and pick the shortest one!

For n cities, there are $n!$ possible routes, where $n! = 1 \times 2 \times \cdots \times (n-1) \times n$.

How many possible routes between all the cities, towns and villages in Sweden?

Does Complexity Matter?

Recall the Traveling Salesperson Problem (TSP): Assume that we design an algorithm that enumerates all routes and chooses the shortest one.

Idea: We could just loop through all the possible routes, check each route's length, and pick the shortest one!

For n cities, there are $n!$ possible routes, where $n! = 1 \times 2 \times \cdots \times (n-1) \times n$.

How many possible routes between all the cities, towns and villages in Sweden?

You should not use this algorithm except for very small number of cities!

Performing Time Complexity Analysis

Based on an abstract model of computation (Random-Access Machine):

- Instructions are executed one after another (i.e., No concurrency).
- Elementary instructions can be performed in constant time (i.e., the time cost does not depend on the size of arguments):
    - Arithmetical operations: addition, subtraction, multiplication, integer division, modulo, . . .
    - Assignment operations, accessing to an array element, . . .
    - Control operations: branching (if-statements), loop control, procedure calls and returns.

Performing Time Complexity Analysis

- The running time of an algorithm is

$$\sum_{\text{all elementary operations}} (\text{cost of operation}) \cdot (\text{number of times operation is executed})$$

Example

Problem statement:

- Input: An array A and a data value x.
- Output: The last position of x in A, or 0 if not found.

```
SEARCH(A, x)
i ← A.length
while A[i] ≠ x and i > 0
      do i ← i − 1
return i
```

6	3	5	1	2	4

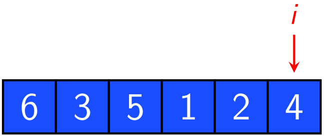
$\boxed{x = 5}$

Example

Problem statement:

- Input: An array A and a data value x.
- Output: The last position of x in A, or 0 if not found.

```
SEARCH(A, x)
i ← A.length
while A[i] ≠ x and i > 0
      do i ← i − 1
return i
```



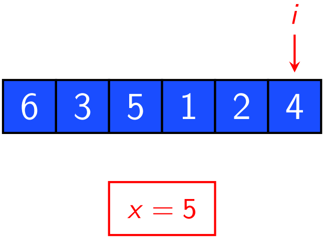
$\boxed{x = 5}$

Example

Problem statement:

- Input: An array A and a data value x.
- Output: The last position of x in A, or 0 if not found.

```
SEARCH(A, x)
  i ← A.length
  while A[i] ≠ x and i > 0
        do i ← i − 1
  return i
```

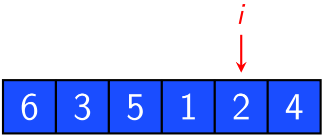


Example

Problem statement:

- Input: An array A and a data value x.
- Output: The last position of x in A, or 0 if not found.

```
SEARCH(A, x)
i ← A.length
while A[i] ≠ x and i > 0
      do i ← i − 1
return i
```



$\boxed{x = 5}$

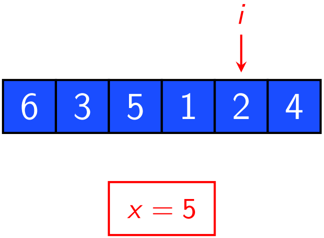
Example

Problem statement:

- Input: An array A and a data value x.
- Output: The last position of x in A, or 0 if not found.

SEARCH(A, x)

$i \leftarrow A.\mathit{length}$
while $A[i] \neq x$ **and** $i > 0$
  do $i \leftarrow i - 1$
return i

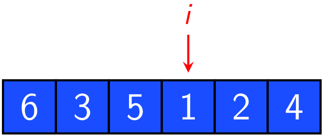


Example

Problem statement:

- Input: An array A and a data value x.
- Output: The last position of x in A, or 0 if not found.

```
SEARCH(A, x)
i ← A.length
while A[i] ≠ x and i > 0
      do i ← i − 1
return i
```



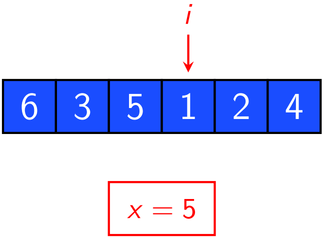
$\boxed{x = 5}$

Example

Problem statement:

- Input: An array A and a data value x.
- Output: The last position of x in A, or 0 if not found.

```
SEARCH(A, x)
i ← A.length
while A[i] ≠ x and i > 0
      do i ← i − 1
return i
```

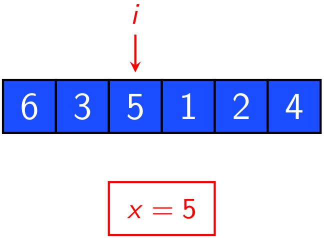


Example

Problem statement:

- Input: An array A and a data value x.
- Output: The last position of x in A, or 0 if not found.

```
SEARCH(A, x)
i ← A.length
while A[i] ≠ x and i > 0
      do i ← i − 1
return i
```

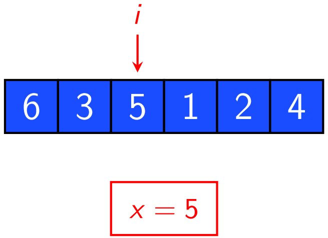


Example

Problem statement:

- Input: An array A and a data value x.
- Output: The last position of x in A, or 0 if not found.

```
SEARCH(A, x)
i ← A.length
while A[i] ≠ x and i > 0
    do i ← i − 1
return i
```



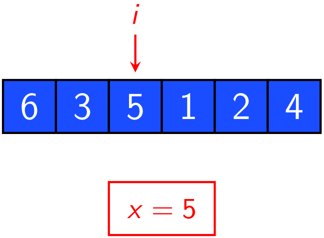
Example

Problem statement:

- Input: An array A and a data value x.
- Output: The last position of x in A, or 0 if not found.

SEARCH(A, x)

```
i ← A.length
while A[i] ≠ x and i > 0
      do i ← i − 1
return i
```



Example-Cost Analysis

Search(A, x)	cost	times
$i \leftarrow A.\mathit{length}$		
while $A[i] \neq x$ and $i > 0$		
do $i \leftarrow i - 1$		
return i		

Example-Cost Analysis

- Assume that each line i has a constant cost c_i

SEARCH(A, x)	cost	times
$i \leftarrow A.\,length$	c_1	
while $A[i] \neq x$ and $i > 0$	c_2	
do $i \leftarrow i - 1$	c_3	
return i	c_4	

Example-Cost Analysis

- Assume that each line i has a constant cost c_i
- Calculate the total number of times that each line code is executed during a run of the algorithm

SEARCH(A, x)	cost	times
1 $i \leftarrow A.\,length$	c_1	
2 **while** $A[i] \neq x$ and $i > 0$	c_2	
3 **do** $i \leftarrow i - 1$	c_3	
4 **return** i	c_4	

Example-Cost Analysis

- Assume that each line i has a constant cost c_i
- Calculate the total number of times that each line code is executed during a run of the algorithm

SEARCH(A, x)	cost	times
$i \leftarrow A.\,length$	c_1	1
while $A[i] \neq x$ and $i > 0$	c_2	
do $i \leftarrow i - 1$	c_3	
return i	c_4	

Example-Cost Analysis

- Assume that each line i has a constant cost c_i
- Calculate the total number of times that each line code is executed during a run of the algorithm

SEARCH(A, x)	cost	times
1 $i \leftarrow A.\,length$	c_1	1
2 **while** $A[i] \neq x$ and $i > 0$	c_2	t
3 **do** $i \leftarrow i - 1$	c_3	
4 **return** i	c_4	

- t is number of times that the condition of **while** is tested

Example-Cost Analysis

- Assume that each line i has a constant cost c_i
- Calculate the total number of times that each line code is executed during a run of the algorithm

SEARCH(A, x)	cost	times
$i \leftarrow A.\mathit{length}$	c_1	1
while $A[i] \neq x$ and $i > 0$	c_2	t
do $i \leftarrow i - 1$	c_3	$t - 1$
return i	c_4	

- t is number of times that the condition of **while** is tested

Example-Cost Analysis

- Assume that each line i has a constant cost c_i
- Calculate the total number of times that each line code is executed during a run of the algorithm

SEARCH(A, x)	cost	times
$i \leftarrow A.\,length$	c_1	1
while $A[i] \neq x$ and $i > 0$	c_2	t
do $i \leftarrow i - 1$	c_3	$t - 1$
return i	c_4	1

- t is number of times that the condition of **while** is tested

Example-Cost Analysis

- Assume that each line i has a constant cost c_i
- Calculate the total number of times that each line code is executed during a run of the algorithm

SEARCH(A, x)	cost	times
$i \leftarrow A.\mathit{length}$	c_1	1
while $A[i] \neq x$ and $i > 0$	c_2	t
do $i \leftarrow i - 1$	c_3	$t - 1$
return i	c_4	1

- t is number of times that the condition of **while** is tested
- The runtime of is $c_1 + t \cdot c_2 + (t-1) \cdot c_3 + c_4$

Example - Cost Analysis

SEARCH(A, x)	cost	times
$i \leftarrow A.length$	c_1	1
while $A[i] \neq x$ and $i > 0$	c_2	4
do $i \leftarrow i - 1$	c_3	3
return i	c_4	1

6	3	5	1	2	4

$\boxed{x = 5}$

The runtime is: $c_1 + 4c_2 + 3c_3 + c_4$

Best, Worst and Average-Case Complexity

Let D_n be the set of all instances of size n and $T(i_n)$ be the runtime for an instance $i_n \in D_n$. We can define three interesting measures:

Best, Worst and Average-Case Complexity

Let D_n be the set of all instances of size n and $T(i_n)$ be the runtime for an instance $i_n \in D_n$. We can define three interesting measures:

- Worst-Case Complexity gives the maximum number of steps taken by the algorithm in any instance of size n: $T(n) = max\{T(i_n) \,|\, i_n \in D_n\}$

Best, Worst and Average-Case Complexity

Let D_n be the set of all instances of size n and $T(i_n)$ be the runtime for an instance $i_n \in D_n$. We can define three interesting measures:

- Worst-Case Complexity gives the maximum number of steps taken by the algorithm in any instance of size n: $T(n) = max\{T(i_n) \,|\, i_n \in D_n\}$
- Best-Case Complexity gives the minimum number of steps taken by the algorithm in any instance of size n: $T(n) = min\{T(i_n) \,|\, i_n \in D_n\}$

Best, Worst and Average-Case Complexity

Let D_n be the set of all instances of size n and $T(i_n)$ be the runtime for an instance $i_n \in D_n$. We can define three interesting measures:

- Worst-Case Complexity gives the maximum number of steps taken by the algorithm in any instance of size n: $T(n) = max\{T(i_n) \mid i_n \in D_n\}$
- Best-Case Complexity gives the minimum number of steps taken by the algorithm in any instance of size n: $T(n) = min\{T(i_n) \mid i_n \in D_n\}$
- Average-Case Complexity gives the average number of steps taken by the algorithm in any instance of size n: $T(n) = \sum_{i_n \in D_n} P(i_n)T(i_n)$ where $P(i_n)$ is the probability of i_n given D_n

Example: Worst Complexity Analysis

SEARCH(A, x)	cost	times
$i \leftarrow A.\,length$	c_1	1
while $A[i] \neq x$ and $i > 0$	c_2	$t + 1$
do $i \leftarrow i - 1$	c_3	t
return i	c_4	1

Example: Worst Complexity Analysis

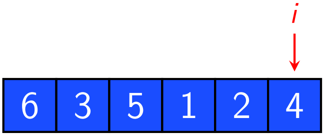
Worst Case: The value of x does not appear in A (i.e., $t = n$).

$\textsc{Search}(A, x)$	cost	times
$i \leftarrow A.\mathit{length}$	c_1	1
while $A[i] \neq x$ and $i > 0$	c_2	$t + 1$
do $i \leftarrow i - 1$	c_3	t
return i	c_4	1

Example: Worst Complexity Analysis

Worst Case: The value of x does not appear in A (i.e., $t = n$).

SEARCH(A, x)	cost	times
$i \leftarrow A.length$	c_1	1
while $A[i] \neq x$ and $i > 0$	c_2	$t + 1$
do $i \leftarrow i - 1$	c_3	t
return i	c_4	1



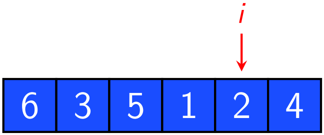
$\boxed{x = 0}$

Example: Worst Complexity Analysis

Worst Case: The value of x does not appear in A (i.e., $t = n$).

SEARCH(A, x)

	cost	times
$i \leftarrow A.\,length$	c_1	1
while $A[i] \neq x$ and $i > 0$	c_2	$t + 1$
do $i \leftarrow i - 1$	c_3	t
return i	c_4	1

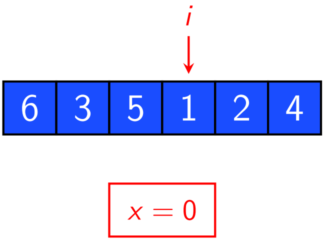


$\boxed{x = 0}$

Example: Worst Complexity Analysis

Worst Case: The value of x does not appear in A (i.e., $t = n$).

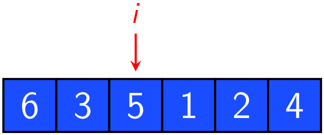
SEARCH(A, x)	cost	times
$i \leftarrow A.\,length$	c_1	1
while $A[i] \neq x$ and $i > 0$	c_2	$t + 1$
do $i \leftarrow i - 1$	c_3	t
return i	c_4	1



Example: Worst Complexity Analysis

Worst Case: The value of x does not appear in A (i.e., $t = n$).

SEARCH(A, x)	cost	times
$i \leftarrow A.\mathit{length}$	c_1	1
while $A[i] \neq x$ and $i > 0$	c_2	$t+1$
do $i \leftarrow i - 1$	c_3	t
return i	c_4	1

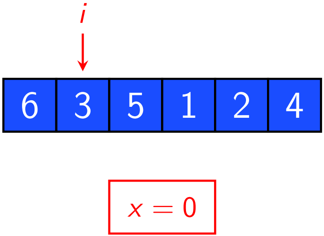


$x = 0$

Example: Worst Complexity Analysis

Worst Case: The value of x does not appear in A (i.e., $t = n$).

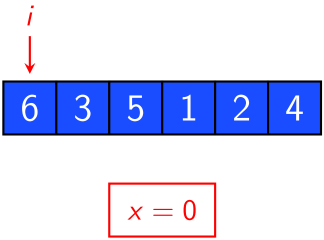
$\textsc{Search}(A, x)$	cost	times
$i \leftarrow A.\mathit{length}$	c_1	1
while $A[i] \neq x$ and $i > 0$	c_2	$t + 1$
do $i \leftarrow i - 1$	c_3	t
return i	c_4	1



Example: Worst Complexity Analysis

Worst Case: The value of x does not appear in A (i.e., $t = n$).

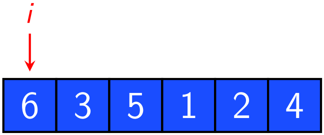
SEARCH(A, x)	cost	times
$i \leftarrow A.\,length$	c_1	1
while $A[i] \neq x$ and $i > 0$	c_2	$t + 1$
do $i \leftarrow i - 1$	c_3	t
return i	c_4	1



Example: Worst Complexity Analysis

Worst Case: The value of x does not appear in A (i.e., $t = n$).

$\textsc{Search}(A, x)$	cost	times
$i \leftarrow A.\mathit{length}$	c_1	1
while $A[i] \neq x$ and $i > 0$	c_2	$n + 1$
do $i \leftarrow i - 1$	c_3	n
return i	c_4	1

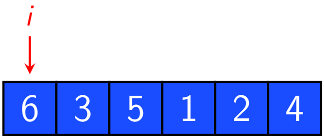


$\boxed{x = 0}$

Example: Worst Complexity Analysis

Worst Case: The value of x does not appear in A (i.e., $t = n$).

SEARCH(A, x)	cost	times
$i \leftarrow A.length$	c_1	1
while $A[i] \neq x$ and $i > 0$	c_2	$n + 1$
do $i \leftarrow i - 1$	c_3	n
return i	c_4	1



$\boxed{x = 0}$

The runtime is: $T(n) = c_1 + (n+1)c_2 + nc_3 + c_4$

Example: Best Complexity Analysis

SEARCH(A, x)	cost	times
$i \leftarrow A.\mathit{length}$	c_1	1
while $A[i] \neq x$ and $i > 0$	c_2	$t + 1$
do $i \leftarrow i - 1$	c_3	t
return i	c_4	1

Example: Best Complexity Analysis

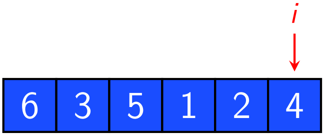
Best Case: The value of x appears at the last position of A ($t = 0$).

SEARCH(A, x)	cost	times
$i \leftarrow A.\mathit{length}$	c_1	1
while $A[i] \neq x$ and $i > 0$	c_2	$t + 1$
do $i \leftarrow i - 1$	c_3	t
return i	c_4	1

Example: Best Complexity Analysis

Best Case: The value of x appears at the last position of A ($t = 0$).

$\textsc{Search}(A, x)$	cost	times
$i \leftarrow A.\mathit{length}$	c_1	1
while $A[i] \neq x$ and $i > 0$	c_2	$t + 1$
do $i \leftarrow i - 1$	c_3	t
return i	c_4	1

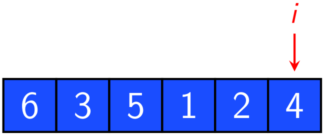


$\boxed{x = 4}$

Example: Best Complexity Analysis

Best Case: The value of x appears at the last position of A ($t = 0$).

SEARCH(A, x)	cost	times
$i \leftarrow A.length$	c_1	1
while $A[i] \neq x$ and $i > 0$	c_2	1
do $i \leftarrow i - 1$	c_3	0
return i	c_4	1

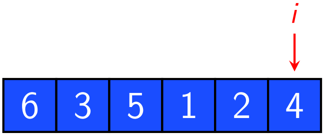


$\boxed{x = 4}$

Example: Best Complexity Analysis

Best Case: The value of x appears at the last position of A ($t = 0$).

SEARCH(A, x)	cost	times
$i \leftarrow A.length$	c_1	1
while $A[i] \neq x$ and $i > 0$	c_2	1
do $i \leftarrow i - 1$	c_3	0
return i	c_4	1

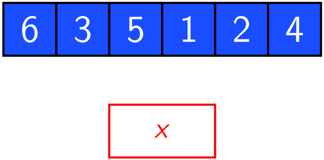


$\boxed{x = 4}$

The runtime is: $T(n) = c_1 + c_2 + c_4$

Example: Average Complexity Analysis

$\textsc{Search}(A, x)$	cost	times
$i \leftarrow A.\mathit{length}$	c_1	1
while $A[i] \neq x$ and $i > 0$	c_2	$t + 1$
do $i \leftarrow i - 1$	c_3	t
return i	c_4	1

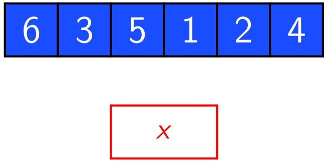


The runtime is: $c_1 + (t+1)c_2 + tc_3 + c_4$

Example: Average Complexity Analysis

Let D_n be the set of all instances of size n, where $T(j_n)$ is the runtime and $P(j_n)$ is the probability for an instance $j_n \in D_n$.

SEARCH(A, x)	cost	times
$i \leftarrow A.\mathit{length}$	c_1	1
while $A[i] \neq x$ and $i > 0$	c_2	$t + 1$
do $i \leftarrow i - 1$	c_3	t
return i	c_4	1



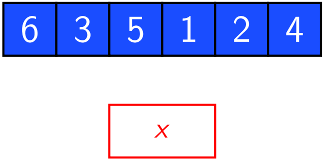
The runtime is: $c_1 + (t + 1)c_2 + tc_3 + c_4$

Example: Average Complexity Analysis

Let D_n be the set of all instances of size n, where $T(j_n)$ is the runtime and $P(j_n)$ is the probability for an instance $j_n \in D_n$.

Average Case: $T(n) = \sum_{j_n \in D_n} P(j_n) T(j_n)$

SEARCH(A, x)	cost	times
$i \leftarrow A.length$	c_1	1
while $A[i] \neq x$ and $i > 0$	c_2	$t + 1$
do $i \leftarrow i - 1$	c_3	t
return i	c_4	1



The runtime is: $c_1 + (t+1)c_2 + tc_3 + c_4$

Example: Average Complexity Analysis

Let D_n be the set of all instances of size n, where $T(j_n)$ is the runtime and $P(j_n)$ is the probability for an instance $j_n \in D_n$.

Average Case: $T(n) = \sum_{j_n \in D_n} P(j_n) T(j_n)$

In this example we assume that all returned values k of the algorithm are equally likely.

Search(A, x)	cost	times
$i \leftarrow A.\mathit{length}$	c_1	1
while $A[i] \neq x$ and $i > 0$	c_2	$t+1$
do $i \leftarrow i - 1$	c_3	t
return i	c_4	1

6	3	5	1	2	4

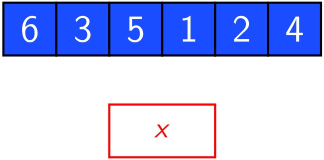
x

The runtime is: $c_1 + (t+1)c_2 + tc_3 + c_4$

Example: Average Complexity Analysis

Let k be a natural number such that $k : 0 \leq k \leq n$. Let j_n be an instance such that the last position of x in these instances is k.

SEARCH(A, x)	cost	times
$i \leftarrow A.\,length$	c_1	1
while $A[i] \neq x$ and $i > 0$	c_2	$t + 1$
do $i \leftarrow i - 1$	c_3	t
return i	c_4	1

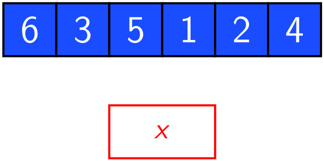


The runtime is: $T(j_n) = c_1 + (t+1)c_2 + tc_3 + c_4$

Example: Average Complexity Analysis

Let k be a natural number such that $k : 0 \leq k \leq n$. Let j_n be an instance such that the last position of x in these instances is k.

SEARCH(A, x)	cost	times
$i \leftarrow A.\mathit{length}$	c_1	1
while $A[i] \neq x$ and $i > 0$	c_2	$n - k + 1$
do $i \leftarrow i - 1$	c_3	$n - k$
return i	c_4	1

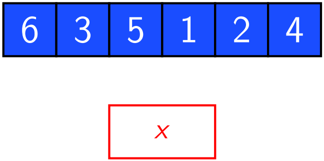


The runtime is: $T(j_n) = c_1 + (t+1)c_2 + tc_3 + c_4$

Example: Average Complexity Analysis

Let k be a natural number such that $k : 0 \leq k \leq n$. Let j_n be an instance such that the last position of x in these instances is k.

$\textsc{Search}(A, x)$	cost	times
$i \leftarrow A.\mathit{length}$	c_1	1
while $A[i] \neq x$ and $i > 0$	c_2	$n - k + 1$
do $i \leftarrow i - 1$	c_3	$n - k$
return i	c_4	1



The runtime is: $T(j_n) = c_1 + (n - k + 1)c_2 + (n - k)c_3 + c_4$

Example: Average Complexity Analysis

Let k be a natural number such that $k : 0 \leq k \leq n$. Let D_n^k be the set of all instances of size n such that the last position of the value of x in this instance is k.

Average Case:

$$T(n) = \sum_{j_n \in D_n} P(j_n) T(j_n)$$

SEARCH(A, x)	cost	times
$i \leftarrow A.length$	c_1	1
while $A[i] \neq x$ and $i > 0$	c_2	$n - k + 1$
do $i \leftarrow i - 1$	c_3	$n - k$
return i	c_4	1

6	3	5	1	2	4

$\boxed{x}$

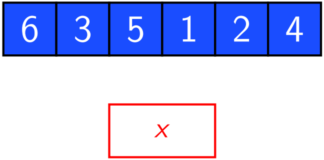
Example: Average Complexity Analysis

Let k be a natural number such that $k : 0 \leq k \leq n$. Let D_n^k be the set of all instances of size n such that the last position of the value of x in this instance is k.

Average Case:

$$T(n) = \sum_{j_n \in D_n} P(j_n) T(j_n)$$

SEARCH(A, x)	cost	times
$i \leftarrow A.\,length$	c_1	1
while $A[i] \neq x$ and $i > 0$	c_2	$n - k + 1$
do $i \leftarrow i - 1$	c_3	$n - k$
return i	c_4	1



Replace $\sum_{j_n \in D_n}$ by $\sum_{k=0}^{n} \sum_{j_n \in D_n^k}$

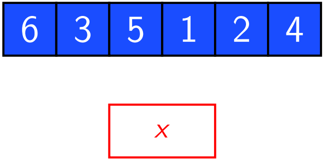
Example: Average Complexity Analysis

Let k be a natural number such that $k : 0 \leq k \leq n$. Let D_n^k be the set of all instances of size n such that the last position of the value of x in this instance is k.

Average Case:

$$T(n) = \sum_{k=0}^{n} \sum_{j_n \in D_n^k} P(j_n) T(j_n)$$

SEARCH(A, x)	cost	times
$i \leftarrow A.\,length$	c_1	1
while $A[i] \neq x$ and $i > 0$	c_2	$n - k + 1$
do $i \leftarrow i - 1$	c_3	$n - k$
return i	c_4	1



Replace $\sum_{j_n \in D_n}$ by $\sum_{k=0}^{n} \sum_{j_n \in D_n^k}$

Example: Average Complexity Analysis

Let k be a natural number such that $k : 0 \leq k \leq n$. Let D_n^k be the set of all instances of size n such that the last position of the value of x in this instance is k.

Average Case:

$$T(n) = \sum_{k=0}^{n} \sum_{j_n \in D_n^k} P(j_n) T(j_n)$$

SEARCH(A, x)	cost	times
$i \leftarrow A.\mathit{length}$	c_1	1
while $A[i] \neq x$ and $i > 0$	c_2	$n - k + 1$
do $i \leftarrow i - 1$	c_3	$n - k$
return i	c_4	1

6	3	5	1	2	4

$\boxed{x}$

Recall that $T(j_n) = c_1 + (n - k + 1)c_2 + (n - k)c_3 + c_4$

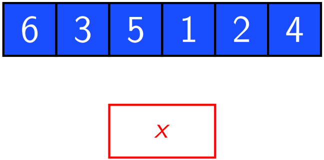
Example: Average Complexity Analysis

Let k be a natural number such that $k : 0 \leq k \leq n$. Let D_n^k be the set of all instances of size n such that the last position of the value of x in this instance is k.

Average Case:

$$T(n) = \sum_{k=0}^{n} \sum_{j_n \in D_n^k} P(j_n)(c_1 + (n-k+1)c_2 + (n-k)c_3 + c_4)$$

$\text{SEARCH}(A, x)$	cost	times
$i \leftarrow A.\,length$	c_1	1
while $A[i] \neq x$ and $i > 0$	c_2	$n-k+1$
do $i \leftarrow i - 1$	c_3	$n-k$
return i	c_4	1



Recall that $T(j_n) = c_1 + (n-k+1)c_2 + (n-k)c_3 + c_4$

Example: Average Complexity Analysis

Let k be a natural number such that $k : 0 \leq k \leq n$. Let D_n^k be the set of all instances of size n such that the last position of the value of x in this instance is k.

Average Case:

$$T(n) = \sum_{k=0}^{n} \sum_{j_n \in D_n^k} P(j_n)(c_1 + (n-k+1)c_2 + (n-k)c_3 + c_4)$$

Search(A, x)	cost	times
$i \leftarrow A.\mathit{length}$	c_1	1
while $A[i] \neq x$ and $i > 0$	c_2	$n-k+1$
do $i \leftarrow i - 1$	c_3	$n-k$
return i	c_4	1

6	3	5	1	2	4

$\boxed{x}$

Rewrite

Example: Average Complexity Analysis

Let k be a natural number such that $k : 0 \leq k \leq n$. Let D_n^k be the set of all instances of size n such that the last position of the value of x in this instance is k.

Average Case:

$$T(n) = \sum_{k=0}^{n} (c_1 + (n-k+1)c_2 + (n-k)c_3 + c_4)(\sum_{j_n \in D_n^k} P(j_n))$$

SEARCH(A, x)	cost	times
$i \leftarrow A.\mathit{length}$	c_1	1
while $A[i] \neq x$ and $i > 0$	c_2	$n-k+1$
do $i \leftarrow i - 1$	c_3	$n-k$
return i	c_4	1

6	3	5	1	2	4

$\boxed{x}$

Rewrite

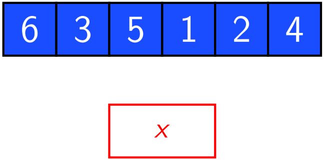
Example: Average Complexity Analysis

Let k be a natural number such that $k : 0 \leq k \leq n$. Let D_n^k be the set of all instances of size n such that the last position of the value of x in this instance is k.

Average Case:

$$T(n) = \quad \sum_{k=0}^{n} (c_1 + (n-k+1)c_2 + (n-k)c_3 + c_4)(\sum_{j_n \in D_n^k} P(j_n))$$

SEARCH(A, x)	cost	times
$i \leftarrow A.\mathit{length}$	c_1	1
while $A[i] \neq x$ and $i > 0$	c_2	$n-k+1$
do $i \leftarrow i - 1$	c_3	$n-k$
return i	c_4	1



By assumption $\sum_{j_n \in D_n^k} P(j_n) = \frac{1}{n+1}$

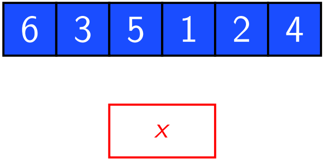
Example: Average Complexity Analysis

Let k be a natural number such that $k : 0 \leq k \leq n$. Let D_n^k be the set of all instances of size n such that the last position of the value of x in this instance is k.

Average Case:

$$T(n) = \sum_{k=0}^{n} \frac{(c_1+(n-k+1)c_2+(n-k)c_3+c_4)}{n+1}$$

SEARCH(A, x)	cost	times
$i \leftarrow A.length$	c_1	1
while $A[i] \neq x$ and $i > 0$	c_2	$n-k+1$
do $i \leftarrow i-1$	c_3	$n-k$
return i	c_4	1



By assumption $\sum_{j_n \in D_n^k} P(j_n) = \frac{1}{n+1}$

Example: Average Complexity Analysis

Let k be a natural number such that $k : 0 \leq k \leq n$. Let D_n^k be the set of all instances of size n such that the last position of the value of x in this instance is k.

Average Case:

$$T(n) = \sum_{k=0}^{n} \frac{(c_1 + (n-k+1)c_2 + (n-k)c_3 + c_4)}{n+1}$$

SEARCH(A, x)	cost	times
$i \leftarrow A.length$	c_1	1
while $A[i] \neq x$ and $i > 0$	c_2	$n-k+1$
do $i \leftarrow i - 1$	c_3	$n-k$
return i	c_4	1

6	3	5	1	2	4

$\boxed{x}$

Rewrite

Example: Average Complexity Analysis

Let k be a natural number such that $k : 0 \leq k \leq n$. Let D_n^k be the set of all instances of size n such that the last position of the value of x in this instance is k.

Average Case:

$$T(n) = \quad c_1 + (n+1)c_2 + nc_3 + c_4 - \left(\tfrac{c_2+c_3}{n+1}\right) \cdot \sum_{k=0}^{n} k$$

SEARCH(A, x)	cost	times
$i \leftarrow A.\mathit{length}$	c_1	1
while $A[i] \neq x$ and $i > 0$	c_2	$n - k + 1$
do $i \leftarrow i - 1$	c_3	$n - k$
return i	c_4	1

6	3	5	1	2	4

$\boxed{x}$

Rewrite

Example: Average Complexity Analysis

Let k be a natural number such that $k : 0 \le k \le n$. Let D_n^k be the set of all instances of size n such that the last position of the value of x in this instance is k.

Average Case:

$$T(n) = \quad c_1 + (n+1)c_2 + nc_3 + c_4 - \left(\frac{c_2+c_3}{n+1}\right) \cdot \sum_{k=0}^{n} k$$

SEARCH(A, x)	cost	times
$i \leftarrow A.\mathit{length}$	c_1	1
while $A[i] \neq x$ and $i > 0$	c_2	$n-k+1$
do $i \leftarrow i - 1$	c_3	$n-k$
return i	c_4	1

6	3	5	1	2	4

$\boxed{x}$

Recall that $\sum_{k=0}^{n} k = \frac{n(n+1)}{2}$

Example: Average Complexity Analysis

Let k be a natural number such that $k : 0 \leq k \leq n$. Let D_n^k be the set of all instances of size n such that the last position of the value of x in this instance is k.

Average Case:

$$T(n) = \qquad c_1 + (n+1)c_2 + nc_3 + c_4 - \left(\tfrac{c_2+c_3}{n+1}\right) \cdot \left(\tfrac{n(n+1)}{2}\right)$$

SEARCH(A, x)	cost	times
$i \leftarrow A.\mathit{length}$	c_1	1
while $A[i] \neq x$ and $i > 0$	c_2	$n - k + 1$
do $i \leftarrow i - 1$	c_3	$n - k$
return i	c_4	1

6	3	5	1	2	4

$\boxed{x}$

Recall that $\sum_{k=0}^{n} k = \frac{n(n+1)}{2}$

Example: Average Complexity Analysis

Let k be a natural number such that $k : 0 \leq k \leq n$. Let D_n^k be the set of all instances of size n such that the last position of the value of x in this instance is k.

Average Case:

$$T(n) = \qquad c_1 + (n+1)c_2 + nc_3 + c_4 - \left(\tfrac{c_2+c_3}{n+1}\right) \cdot \left(\tfrac{n(n+1)}{2}\right)$$

SEARCH(A, x)	cost	times
$i \leftarrow A.length$	c_1	1
while $A[i] \neq x$ and $i > 0$	c_2	$n - k + 1$
do $i \leftarrow i - 1$	c_3	$n - k$
return i	c_4	1

6	3	5	1	2	4

$\boxed{x}$

Rewrite

Example: Average Complexity Analysis

Let k be a natural number such that $k : 0 \leq k \leq n$. Let D_n^k be the set of all instances of size n such that the last position of the value of x in this instance is k.

Average Case:

$$T(n) = \qquad c_1 + (n+1)c_2 + nc_3 + c_4 - (c_2 + c_3) \cdot \left(\frac{n}{2}\right)$$

$\textsc{Search}(A, x)$	cost	times
$i \leftarrow A.\,length$	c_1	1
while $A[i] \neq x$ and $i > 0$	c_2	$n - k + 1$
do $i \leftarrow i - 1$	c_3	$n - k$
return i	c_4	1

6	3	5	1	2	4

$\boxed{x}$

Rewrite

Example: Average Complexity Analysis

Let k be a natural number such that $k : 0 \leq k \leq n$. Let D_n^k be the set of all instances of size n such that the last position of the value of x in this instance is k.

Average Case:

$$T(n) = \qquad c_1 + (\frac{n}{2} + 1)c_2 + \frac{n}{2}c_3 + c_4$$

SEARCH(A, x)	cost	times
$i \leftarrow A.length$	c_1	1
while $A[i] \neq x$ and $i > 0$	c_2	$n - k + 1$
do $i \leftarrow i - 1$	c_3	$n - k$
return i	c_4	1

6	3	5	1	2	4

$\boxed{x}$

Rewrite

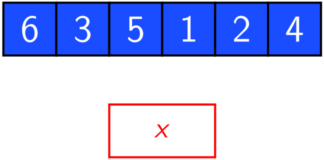
Example: Average Complexity Analysis

Let k be a natural number such that $k : 0 \leq k \leq n$. Let D_n^k be the set of all instances of size n such that the last position of the value of x in this instance is k.

Average Case:

$$T(n) = \qquad c_1 + (\tfrac{n}{2} + 1)c_2 + \tfrac{n}{2}c_3 + c_4$$

SEARCH(A, x)	cost	times
$i \leftarrow A.\mathit{length}$	c_1	1
while $A[i] \neq x$ and $i > 0$	c_2	$n - k + 1$
do $i \leftarrow i - 1$	c_3	$n - k$
return i	c_4	1



Correctness – Loop Invariants

Loop Invariant:

- Property which remains true throughout the execution of the loop.
- It implies “correctness” of the program.

How to Show a Loop Invariant?

- Initialization: The invariant holds prior to the first iteration of the loop.
- Maintenance: The invariant is preserved by each iteration of the loop. In other words, if it holds before the next iteration, it will also hold after performing that iteration. This continues until (and including) the point of termination of the loop.
- Termination: At the point of termination, the invariant implies a "useful property", which in turn can be used for proving correctness of the program.