Chapter 4. Computation
Chapter 5. Searching and Sorting Algorithms

4.1 An introduction to algorithms
5.1 Searching and algorithms
5.2 Binary search

There are many general classes of problems that arise in discrete mathematics.

Examples:
- find the largest integer in the sequence of integers
- list all subsets of the given set
- given a sequence of integers put them in increasing order, and so on.

4.1 An introduction to algorithms

There are many general classes of problems that arise in discrete mathematics.

Examples:
- find the largest integer in the sequence of integers
- list all subsets of the given set
- given s sequence of integers put them in increasing order, and so on.

The first thing to do:
construct a model that translates the problem into mathematical context

4.1 An introduction to algorithms

There are many general classes of problems that arise in discrete mathematics.

Examples:
- find the largest integer in the sequence of integers
- list all subsets of the given set
- given s sequence of integers put them in increasing order, and so on.

The first thing to do:
construct a model that translates the problem into mathematical context

The next thing to do:
find/construct a method that will solve the [general] problem using the model.
Ideally: *a sequence of steps that leads to the desired answer*

4.1 An introduction to algorithms

There are many general classes of problems that arise in discrete mathematics.

Examples:
- find the largest integer in the sequence of integers
- list all subsets of the given set
- given s sequence of integers put them in increasing order, and so on.

The first thing to do:
construct a model that translates the problem into mathematical context

The next thing to do:
find/construct a method that will solve the [general] problem using the model.
Ideally: *a sequence of steps that leads to the desired answer*

an algorithm is a *finite* set of precise instructions for performing a computation or for solving a problem.

Example 1: describe an algorithm for finding the minimum (smallest) value in a finite sequence of integers.

Example 1: describe an algorithm for finding the minimum (smallest) value in a finite sequence of integers.

Solution (algorithm):
1. set the temporary minimum value (**min**) equal to the first integer in the sequence

Example 1: describe an algorithm for finding the minimum (smallest) value in a finite sequence of integers.

Solution (algorithm):
1. set the temporary minimum value (**min**) equal to the first integer in the sequence

2. compare the next integer in the sequence to the temporary **min** . If it is smaller than the **min**, set **min** equal to this integer. Do nothing otherwise.

Example 1: describe an algorithm for finding the minimum (smallest) value in a finite sequence of integers.

Solution (algorithm):
1. set the temporary minimum value (**min**) equal to the first integer in the sequence

2. compare the next integer in the sequence to the temporary **min** . If it is smaller than the **min**, set **min** equal to this integer. Do nothing otherwise.

3. Repeat step 2 (if there are more integers in the sequence).

Example 1: describe an algorithm for finding the minimum (smallest) value in a finite sequence of integers.

Solution (algorithm):

1. set the temporary minimum value (**min**) equal to the first integer in the sequence

2. compare the next integer in the sequence to the temporary **min** . If it is smaller than the **min**, set **min** equal to this integer. Do nothing otherwise.

3. Repeat step 2 (if there are more integers in the sequence).

4. Stop when there are no integers left in the sequence. The **min** is the smallest integer in the sequence.

Example 1: describe an algorithm for finding the minimum (smallest) value in a finite sequence of integers.

Solution (algorithm):
1. set the temporary minimum value (**min**) equal to the first integer in the sequence

2. compare the next integer in the sequence to the temporary **min** . If it is smaller than the **min**, set **min** equal to this integer. Do nothing otherwise.

3. Repeat step 2 (if there are more integers in the sequence).

4. Stop when there are no integers left in the sequence. The **min** is the smallest integer in the sequence.

Assume that there were 12 integers in the sequence.
How many comparison operations will be performed?

Example 1: describe an algorithm for finding the minimum (smallest) value in a finite sequence of integers.

Solution (algorithm):
1. set the temporary minimum value (**min**) equal to the first integer in the sequence

2. compare the next integer in the sequence to the temporary **min** . If it is smaller than the **min**, set **min** equal to this integer. Do nothing otherwise.

3. Repeat step 2 (if there are more integers in the sequence).

4. Stop when there are no integers left in the sequence. The **min** is the smallest integer in the sequence.

Assume that there were 12 integers in the sequence.
How many comparison operations will be performed? 11

How many assignments/re-assignments will be performed?

Example 1: describe an algorithm for finding the minimum (smallest) value in a finite sequence of integers.

Solution (algorithm):
1. set the temporary minimum value (**min**) equal to the first integer in the sequence

2. compare the next integer in the sequence to the temporary **min** . If it is smaller than the **min**, set **min** equal to this integer. Do nothing otherwise.

3. Repeat step 2 (if there are more integers in the sequence).

4. Stop when there are no integers left in the sequence. The **min** is the smallest integer in the sequence.

Assume that there were 12 integers in the sequence.
How many comparison operations will be performed? 11

How many assignments/re-assignments will be performed?
min: 1 max: 12

Example 1: describe an algorithm for finding the minimum (smallest) value in a finite sequence of integers.

Solution (algorithm):
1. set the temporary minimum value (**min**) equal to the first integer in the sequence

2. compare the next integer in the sequence to the temporary **min** . If it is smaller than the **min**, set **min** equal to this integer. Do nothing otherwise.

3. Repeat step 2 (if there are more integers in the sequence).

4. Stop when there are no integers left in the sequence. The **min** is the smallest integer in the sequence.

Assume that there were 12 integers in the sequence.
How many comparison operations will be performed? 11

How many assignments/re-assignments will be performed?
min: 1 max: 12

This description of the algorithm is nice, but too many words.

We can describe an algorithm using a computer language, but then we are limited to the instructions permitted in the language + many programming languages are in common use (undesirable to chose only one)

A better solution: use pseudocode
(intermediate step between English and a programming language)

We can describe an algorithm using a computer language, but then we are limited to the instructions permitted in the language + many programming languages are in common use (undesirable to chose only one)

A better solution: use pseudocode
(intermediate step between English and a programming language)

Example 1 (using pseudocode):
Input: $a_1, a_2, \ldots, a_n$: integers
Output: the smallest element

```
procedure min(a1,a2,...,an)
min := a1
For i := 2 to n
   If (min > ai) , min := ai
End-for
Return(min)
```

We will come back to it in few slides

4.1 An introduction to algorithms

• assignment operator: `x := 10`

• if-statement tests a *condition*, and executes one or more instructions if the *condition* evaluates to true

```
If (x > 7) , x := x-7
```

If there are more than one instruction to execute, then consider using
`If – End-if`

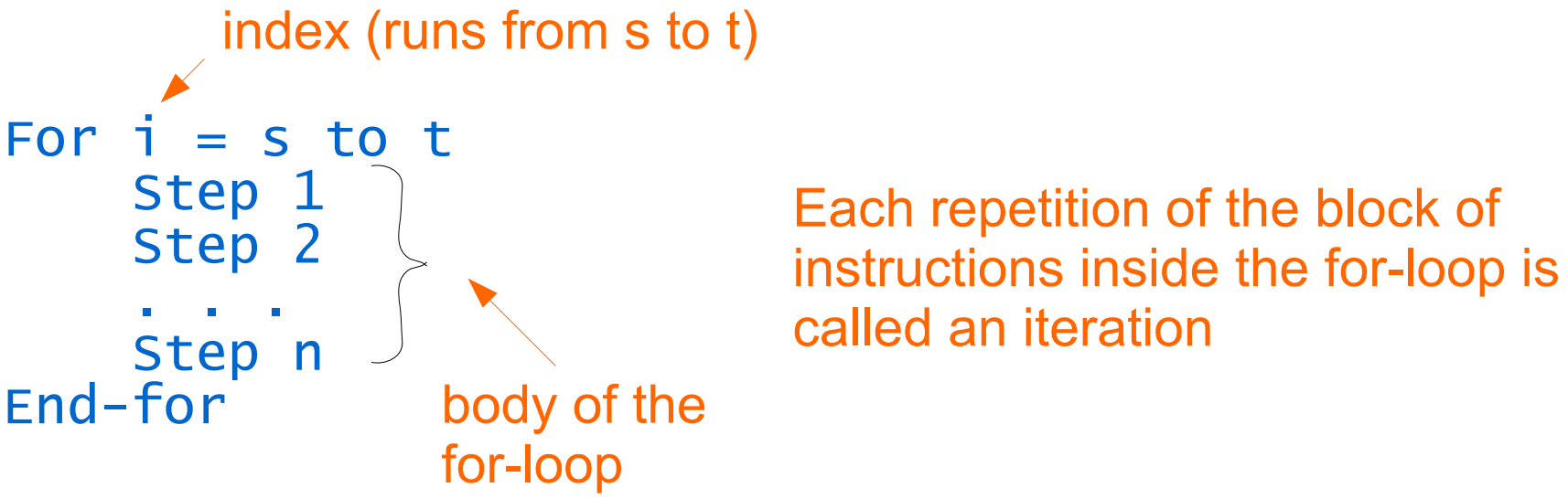
```
If ( condition )
    Step 1
    Step 2
    . . .
    Step n
End-if
```

• The output of an algorithm is specified by a return statement

```
. . .
Return(x)
```

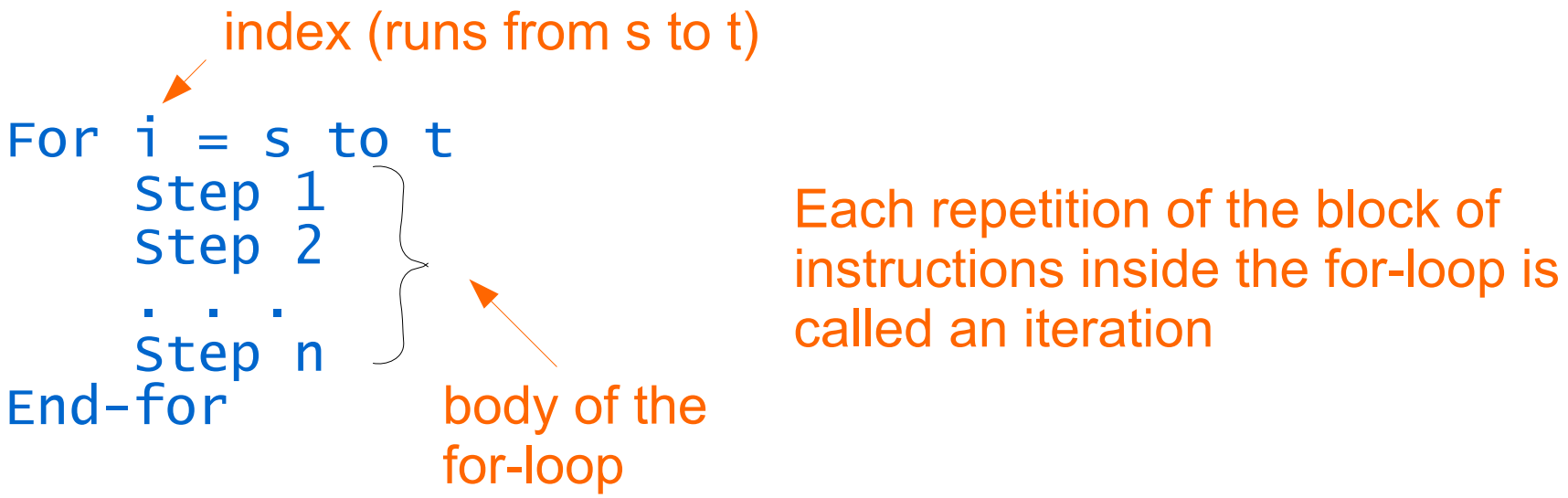
4.1 An introduction to algorithms

• In a for-loop, a block of instructions is executed a fixed number of times as specified in the first line of the for-loop



4.1 An introduction to algorithms

• In a for-loop, a block of instructions is executed a fixed number of times as specified in the first line of the for-loop



Example:

$a_1 := 2$, $a_2 := 7$, $a_3 := 1$, $a_4 := 5$, $s := 0$

For i:= 1 to 4

    $s := s + a_i$

End-for

Return(s)

4.1 An introduction to algorithms

• In a for-loop, a block of instructions is executed a fixed number of times as specified in the first line of the for-loop

index (runs from s to t)

```
For i = s to t
    Step 1
    Step 2
    . . .
    Step n
End-for
```

body of the for-loop

Each repetition of the block of instructions inside the for-loop is called an iteration

Example:

$a_1 := 2$, $a_2 := 7$, $a_3 := 1$, $a_4 := 5$, $s := 0$

```
For i:= 1 to 4
    s := s + a_i
End-for
Return(s)
```

1st iteration: i = 1, s = 0
s := 0 + 2 = 2
2nd iteration: i = 2, s = 2
s := 2 + 7 = 9
3rd iteration: i = 3, s = 9
s := 9 + 1 = 10
4th iteration: i = 4, s = 10
s := 10 + 5 = 15
Return(15)

4.1 An introduction to algorithms

• A while-loop iterates an unknown number of times, ending when a certain condition becomes false

```
While (condition)
     Step 1
     Step 2
     . . .
     Step n
End-while
```

body of the while-loop

Each repetition of the block of instructions inside the while-loop is called an iteration

We can describe an algorithm using a computer language, but then we are limited to the instructions permitted in the language + many programming languages are in common use (undesirable to chose only one)

A better solution: use pseudocode
(intermediate step between English and a programming language)

Example 1 (using pseudocode):
Input: $a_1, a_2, \ldots, a_n$: integers
Output: the smallest element

```
procedure min(a1,a2,...,an)
min := a1
For i := 2 to n
   If (min > ai) , min := ai
End-for
Return(min)
```

We can describe an algorithm using a computer language, but then we are limited to the instructions permitted in the language + many programming languages are in common use (undesirable to chose only one)

A better solution: use pseudocode
(intermediate step between English and a programming language)

Example 1 (using pseudocode):
Input: $a_1, a_2, \ldots, a_n$: integers
Output: the smallest element

```
procedure min(a1,a2,...,an)
min := a1
For i := 2 to n
   If (min > ai) , min := ai
End-for
Return(min)
```

Initially we set the first element from the sequence to the the minimum min.

We can describe an algorithm using a computer language, but then we are limited to the instructions permitted in the language + many programming languages are in common use (undesirable to chose only one)

A better solution: use pseudocode
(intermediate step between English and a programming language)

Example 1 (using pseudocode):
Input: $a_1, a_2, \ldots, a_n$: integers
Output: the smallest element

```
procedure min(a1,a2,...,an)
min := a1
For i := 2 to n
   If (min > ai) , min := ai
End-for
Return(min)
```

At every iteration: we check weather the next element in the sequence is smaller than min. If it is then we store the new minimum value in min, and do nothing otherwise

There are several properties that algorithms usually share:

- *input* an algorithm has input values from a specified set
- *output* for each set of input values an algorithm produces output values (from a specified set)
- *definiteness* the steps of the algorithm must be defined precisely
- *correctness* an algorithm should produce the correct output values for each set of input values
- *finiteness* an algorithm should produce the desired output after a finite number of steps for any input in the set
- *effectiveness* it must be possible to perform each step of an algorithm exactly and in a finite amount of time
- *generality* the procedure should be applicable for all problems of the desired form (not just for a particular set of input values)

Searching Algorithms

The problem of locating an element in an ordered list occurs in many contexts.
For instance, a program that checks the spelling of words, searches them in a dictionary (which is just an ordered list of words)

Problems of this kind are called searching problems.

Searching Algorithms

The problem of locating an element in an ordered list occurs in many contexts.
For instance, a program that checks the spelling of words, searches them in a dictionary (which is just an ordered list of words)

Problems of this kind are called searching problems.

General searching problem:
locate an element x in a list of distinct elements $a_1, \ldots, a_n$, or determine that it is not in the list.

Searching Algorithms

The problem of locating an element in an ordered list occurs in many contexts.
For instance, a program that checks the spelling of words, searches them in a dictionary (which is just an ordered list of words)

Problems of this kind are called searching problems.

General searching problem:
locate an element x in a list of distinct elements $a_1, \ldots, a_n$, or determine that it is not in the list.

the solution/answer to this problem: location i of the term in the list that equals x (i.e. $a_i=x$), and location $i=0$ if x is not in the list.

Searching Algorithms

The problem of locating an element in an ordered list occurs in many contexts.
For instance, a program that checks the spelling of words, searches them in a dictionary (which is just an ordered list of words)

Problems of this kind are called searching problems.

General searching problem:
locate an element x in a list of distinct elements $a_1, \ldots, a_n$, or determine that it is not in the list.

the solution to this problem: location i of the term in the list that equals x, i.e. $a_i = x$ and 0 if x is not in the list.

We will study two algorithms:
The Linear Search (Sequential Search)
The Binary Search

5.1 Linear Search (Sequential Search)

Input: x : integer; $a_1,...,a_n$: distinct integers

Output: *location, i.e.* the index(subscript) of the term that equals x, or *-1* if x is not found

procedure *linear search*($x, a_1,...,a_n$)
i := 1
While ($i \leq n$ and $x \neq a_i$)
    i : = i + 1
End-while
If ($i \leq n$), *location* := *i*
Else *location* := *-1*
Return(*location*)

5.1 Linear Search (Sequential Search)

procedure *linear search*($x, a_1,...,a_n$)
i := 1
While ($i \leq n$ and $x \neq a_i$)
    i : = i + 1
End-while
If ($i \leq n$), *location* := *i*
Else *location* := *-1*
Return(*location*)

Example 2:
Assume that we have the following set **A** = {1, 43, 23, 17, 21, 90}.
And we'd like to search for 23 in this set.

Let's see how the algorithm will work:

5.1 Linear Search (Sequential Search)

procedure *linear search*($x, a_1,...,a_n$)
i := 1
While ($i \leq n$ and $x \neq a_i$)
    i : = i + 1
End-while
If ($i \leq n$), *location* := *i*
Else *location* := *-1*
Return(*location*)

Example 2:
Assume that we have the following set **A** = {1, 43, 23, 17, 21, 90}.
And we'd like to search for 23 in this set.

Let's see how the algorithm will work:
n = 6

5.1 Linear Search (Sequential Search)

procedure *linear search*$(x, a_1,...,a_n)$
i := 1
While $(i \leq n$ and $x \neq a_i)$
    i : = i + 1
End-while
If $(i \leq n)$, *location* := *i*
Else *location* := *-1*
Return(*location*)

Example 2:
Assume that we have the following set **A** = {1, 43, 23, 17, 21, 90}.
And we'd like to search for 23 in this set.

Let's see how the algorithm will work:
n = 6
1. i = 1 $1 \leq 6$ and $23 \neq 1 \leftarrow a_1$ **A** = {1, 43, 23, 17, 21, 90}

5.1 Linear Search (Sequential Search)

procedure *linear search*($x, a_1,...,a_n$)
i := 1
While ($i \leq n$ and $x \neq a_i$)
    i : = i + 1
End-while
If ($i \leq n$), *location* := *i*
Else *location* := *-1*
Return(*location*)

Example 2:
Assume that we have the following set **A** = {1, 43, 23, 17, 21, 90}.
And we'd like to search for 23 in this set.

Let's see how the algorithm will work:
n = 6

1. i = 1 $1 \leq 6$ and $23 \neq 1 \leftarrow a_1$ **A** = {1, 43, 23, 17, 21, 90}

2. i = 2 $2 \leq 6$ and $23 \neq 43 \leftarrow a_2$ **A** = {1, 43, 23, 17, 21, 90}

5.1 Linear Search (Sequential Search)

procedure *linear search*($x, a_1,...,a_n$)
i := 1
While ($i \leq n$ and $x \neq a_i$)
    i : = i + 1
End-while
If ($i \leq n$), *location* := *i*
Else *location* := *-1*
Return(*location*)

Example 2:
Assume that we have the following set **A** = {1, 43, 23, 17, 21, 90}.
And we'd like to search for 23 in this set.

Let's see how the algorithm will work:
n = 6

1. i = 1 $1 \leq 6$ and $23 \neq 1 \leftarrow a_1$ **A** = {1, 43, 23, 17, 21, 90}

2. i = 2 $2 \leq 6$ and $23 \neq 43 \leftarrow a_2$ **A** = {1, 43, 23, 17, 21, 90}

3. i = 3 $3 \leq 6$ and $23 = 23 \leftarrow a_3$ **A** = {1, 43, 23, 17, 21, 90}

5.1 Linear Search (Sequential Search)

procedure *linear search*($x, a_1,...,a_n$)
i := 1
While ($i \leq n$ and $x \neq a_i$)
    i : = i + 1
End-while
If ($i \leq n$), *location* := *i*
Else *location* := *-1*
Return(*location*)

Example 2:
Assume that we have the following set **A** = {1, 43, 23, 17, 21, 90}.
And we'd like to search for 23 in this set.

Let's see how the algorithm will work:
n = 6

1. i = 1 $1 \leq 6$ and $23 \neq 1 \leftarrow a_1$ **A** = {1, 43, 23, 17, 21, 90}

2. i = 2 $2 \leq 6$ and $23 \neq 43 \leftarrow a_2$ **A** = {1, 43, 23, 17, 21, 90}

3. i = 3 $3 \leq 6$ and $23 = 23 \leftarrow a_3$ **A** = {1, 43, 23, 17, 21, 90}

- condition of while is False

5.1 Linear Search (Sequential Search)

procedure *linear search*($x, a_1,...,a_n$)
i := 1
While ($i \leq n$ and $x \neq a_i$)
    i : = i + 1
End-while
If ($i \leq n$), *location* := *i*
Else *location* := *-1*
Return(*location*)

Example 2:
Assume that we have the following set **A** = {1, 43, 23, 17, 21, 90}.
And we'd like to search for 23 in this set.

Let's see how the algorithm will work:
n = 6

1. i = 1 $1 \leq 6$ and $23 \neq 1 \leftarrow a_1$ **A** = {1, 43, 23, 17, 21, 90}

2. i = 2 $2 \leq 6$ and $23 \neq 43 \leftarrow a_2$ **A** = {1, 43, 23, 17, 21, 90}

3. i = 3 $3 \leq 6$ and $23 = 23 \leftarrow a_3$ **A** = {1, 43, 23, 17, 21, 90}

- condition of while is False

$3 \leq 6$ therefore location := 3

Important! for this algorithm the elements of the set have to be *ordered from smallest to largest*.

how it works: it proceeds by comparing the element to be located to the middle term in the list. List is then split into two smaller sublists of the same size (or one less). The appropriate sublist is taken and the comparison of the element to be located and the middle term in the sublist is done. And so on.

Important! for this algorithm the elements of the set have to be *ordered from smallest to largest*.

how it works: it proceeds by comparing the element to be located to the middle term in the list. List is then split into two smaller sublists of the same size (or one less). The appropriate sublist is taken and the comparison of the element to be located and the middle term in the sublist is done. And so on.

5.2 Binary Search

Important! for this algorithm the elements of the set have to be *ordered from smallest to largest*.

how it works: it proceeds by comparing the element to be located to the middle term in the list. List is then split into two smaller sublists of the same size (or one less). The appropriate sublist is taken and the comparison of the element to be located and the middle term in the sublist is done. And so on.

Input: x : integer; $a_1,...,a_n$: increasing integers

Output: *location* of the term that equals x, or *-1* if x is not found

procedure *binary search*(x, $a_1,...,a_n$)

$i := 1$, $j := n$ — i is the left endpoint, j is the right endpoint

While ($i < j$) — *while the* left endpoint is less than the right endpoint

  $m := \lfloor (i+j)/2 \rfloor$ — location/index of the middle element

  If ($x > a_m$), $i := m+1$

  Else $j := m$

If ($x = a_i$), *location* := i

Else *location* := *-1*

Return(location)

Example 3:
Assume that we have the following set **A** = {1, 43, 23, 17, 21, 90}.
And we'd like to search for 23 in this set. Let's use binary search algorithm this time.

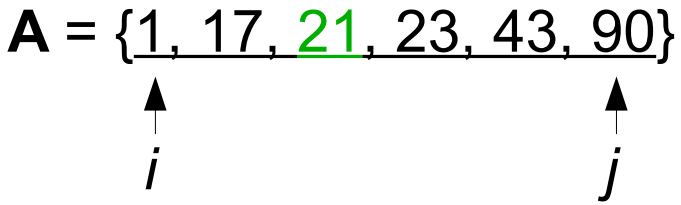
Example 3:
Assume that we have the following set **A** = {1, 43, 23, 17, 21, 90}.
And we'd like to search for 23 in this set. Let's use binary search algorithm this time.

! Set **A** has to be ordered from smallest to largest.
So let **A** = {1, 17, 21, 23, 43, 90}. *n*=6

Example 3:
Assume that we have the following set **A** = {1, 43, 23, 17, 21, 90}.
And we'd like to search for 23 in this set. Let's use binary search algorithm this time.

! Set **A** has to be ordered from smallest to largest.
So let **A** = {1, 17, 21, 23, 43, 90}. *n=6*

1. *i = 1, j=6*
2. 1 < 6 ?



Example 3:
Assume that we have the following set **A** = {1, 43, 23, 17, 21, 90}.
And we'd like to search for 23 in this set. Let's use binary search algorithm this time.

! Set **A** has to be ordered from smallest to largest.
So let **A** = {1, 17, 21, 23, 43, 90}. *n=6*

1. *i = 1, j=6*
2. 1 < 6 ? True
m = $\lfloor (1+6)/2 \rfloor$ = $\lfloor 3.5 \rfloor$ = 3
23 > a_3 = 21 ?

A = {1, 17, 21, 23, 43, 90}
i *j*

Example 3:
Assume that we have the following set **A** = {1, 43, 23, 17, 21, 90}.
And we'd like to search for 23 in this set. Let's use binary search algorithm this time.

! Set **A** has to be ordered from smallest to largest.
So let **A** = {1, 17, 21, 23, 43, 90}. *n=6*

1. *i = 1, j=6*
2. 1 < 6 ? True
m = $\lfloor (1+6)/2 \rfloor$ = $\lfloor 3.5 \rfloor$ = 3
23 > a_3 = 21 ? YES
i = 3+1 = 4

A = {1, 17, 21, 23, 43, 90}
i *j*

Example 3:
Assume that we have the following set **A** = {1, 43, 23, 17, 21, 90}.
And we'd like to search for 23 in this set. Let's use binary search algorithm this time.

! Set **A** has to be ordered from smallest to largest.
So let **A** = {1, 17, 21, 23, 43, 90}. *n=6*

1. *i = 1, j=6*
2. 1 < 6 ? True
m = $\lfloor (1+6)/2 \rfloor$ = $\lfloor 3.5 \rfloor$ = 3
23 > a_3 = 21 ? YES
i = 3+1 = 4
3. 4 < 6 ?

A = {1, 17, 21, 23, 43, 90}
i *j*

Example 3:
Assume that we have the following set **A** = {1, 43, 23, 17, 21, 90}.
And we'd like to search for 23 in this set. Let's use binary search algorithm this time.

! Set **A** has to be ordered from smallest to largest.
So let **A** = {1, 17, 21, 23, 43, 90}. *n=6*

1. *i = 1, j=6*
2. 1 < 6 ? True
m = $\lfloor (1+6)/2 \rfloor$ = $\lfloor 3.5 \rfloor$ = 3
23 > a_3 = 21 ? YES
i = 3+1 = 4

A = {1, 17, 21, 23, 43, 90}
i *j*

3. 4 < 6 ? True
m = $\lfloor (4+6)/2 \rfloor$ = $\lfloor 5 \rfloor$ = 5
23 > a_5 = 43 ?

A = {1, 17, 21, 23, 43, 90}
i *j*

Example 3:
Assume that we have the following set **A** = {1, 43, 23, 17, 21, 90}.
And we'd like to search for 23 in this set. Let's use binary search algorithm this time.

! Set **A** has to be ordered from smallest to largest.
So let **A** = {1, 17, 21, 23, 43, 90}. *n=6*

1. *i = 1, j=6*
2. 1 < 6 ? True
m = $\lfloor (1+6)/2 \rfloor$ = $\lfloor 3.5 \rfloor$ = 3
23 > a_3 = 21 ? YES
i = 3+1 = 4

A = {1, 17, 21, 23, 43, 90}
i *j*

3. 4 < 6 ? True
m = $\lfloor (4+6)/2 \rfloor$ = $\lfloor 5 \rfloor$ = 5
23 > a_5 = 43 ? NO
j = 5

A = {1, 17, 21, 23, 43, 90}
i *j*

Example 3:
Assume that we have the following set **A** = {1, 43, 23, 17, 21, 90}.
And we'd like to search for 23 in this set. Let's use binary search algorithm this time.

! Set **A** has to be ordered from smallest to largest.
So let **A** = {1, 17, 21, 23, 43, 90}. *n=6*

1. *i = 1, j=6*
2. 1 < 6 ? True
m = $\lfloor (1+6)/2 \rfloor$ = $\lfloor 3.5 \rfloor$ = 3
23 > a_3 = 21 ? YES
i = 3+1 = 4

A = {1, 17, 21, 23, 43, 90}
i *j*

3. 4 < 6 ? True
m = $\lfloor (4+6)/2 \rfloor$ = $\lfloor 5 \rfloor$ = 5
23 > a_5 = 43 ? NO
j = 5

A = {1, 17, 21, 23, 43, 90}
i *j*

4. 4 < 5 ?

Example 3:
Assume that we have the following set **A** = {1, 43, 23, 17, 21, 90}.
And we'd like to search for 23 in this set. Let's use binary search algorithm this time.

! Set **A** has to be ordered from smallest to largest.
So let **A** = {1, 17, 21, 23, 43, 90}. *n=6*

1. *i = 1, j=6*
2. 1 < 6 ? True
m = $\lfloor (1+6)/2 \rfloor$ = $\lfloor 3.5 \rfloor$ = 3
23 > a_3 = 21 ? YES
i = 3+1 = 4
A = {1, 17, 21, 23, 43, 90}
i *j*
3. 4 < 6 ? True
m = $\lfloor (4+6)/2 \rfloor$ = $\lfloor 5 \rfloor$ = 5
23 > a_5 = 43 ? NO
j = 5
A = {1, 17, 21, 23, 43, 90}
i *j*
4. 4 < 5 ? True
m = $\lfloor (4+5)/2 \rfloor$ = $\lfloor 4.5 \rfloor$ = 4
23 > a_4 = 23 ?
A = {1, 17, 21, 23, 43, 90}
i *j*

Example 3:
Assume that we have the following set **A** = {1, 43, 23, 17, 21, 90}.
And we'd like to search for 23 in this set. Let's use binary search algorithm this time.

! Set **A** has to be ordered from smallest to largest.
So let **A** = {1, 17, 21, 23, 43, 90}. *n=6*

1. *i = 1, j=6*
2. 1 < 6 ? True
m = $\lfloor (1+6)/2 \rfloor$ = $\lfloor 3.5 \rfloor$ = 3
23 > a_3 = 21 ? YES
i = 3+1 = 4
A = {1, 17, 21, 23, 43, 90}
i *j*
3. 4 < 6 ? True
m = $\lfloor (4+6)/2 \rfloor$ = $\lfloor 5 \rfloor$ = 5
23 > a_5 = 43 ? NO
j = 5
A = {1, 17, 21, 23, 43, 90}
i *j*
4. 4 < 5 ? True
m = $\lfloor (4+5)/2 \rfloor$ = $\lfloor 4.5 \rfloor$ = 4
23 > a_4 = 23 ? NO
j = 4
A = {1, 17, 21, 23, 43, 90}
i *j*

Example 3:
Assume that we have the following set **A** = {1, 43, 23, 17, 21, 90}.
And we'd like to search for 23 in this set. Let's use binary search algorithm this time.

! Set **A** has to be ordered from smallest to largest.
So let **A** = {1, 17, 21, 23, 43, 90}. *n*=6

5. 4 < 4 ?

A = {1, 17, 21, 23, 43, 90}
i *j*

Example 3:
Assume that we have the following set **A** = {1, 43, 23, 17, 21, 90}.
And we'd like to search for 23 in this set. Let's use binary search algorithm this time.

! Set **A** has to be ordered from smallest to largest.
So let **A** = {1, 17, 21, 23, 43, 90}. *n=6*

5. 4 < 4 ? False
23 = a_4(=23) ?

A = {1, 17, 21, 23, 43, 90}
i *j*

Example 3:
Assume that we have the following set **A** = {1, 43, 23, 17, 21, 90}.
And we'd like to search for 23 in this set. Let's use binary search algorithm this time.

! Set **A** has to be ordered from smallest to largest.
So let **A** = {1, 17, 21, 23, 43, 90}. *n=6*

5. 4 < 4 ? False
23 = a_4(=23) ? YES location = 4 **A** = {1, 17, 21, 23, 43, 90}
i *j*

Output: 4 (*23* is the fourth element in the ordered set).