Chapter 4.1: Algorithms

Computer Fundamentals and Introduction to Programming

Problem · Algorithm · Tracing · Searching · Sorting

Sercan KÜLCÜ · Giresun University

© Sercan Külcü. All rights reserved. · Interactive HTML version of these slides: https://sercankulcu.github.io/files/computer/slides/Chapter_04_01_Algorithms.html

Learning Outcomes

Specify

Pin down the inputs, outputs and constraints.

Design

Break the solution into clear, finite steps.

Verify

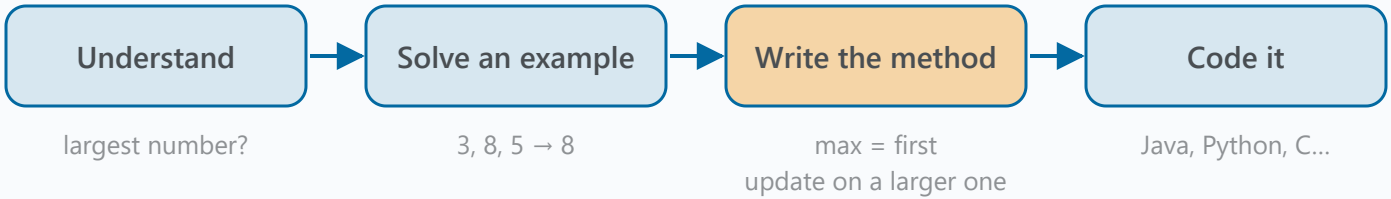
Trace and test the algorithm on examples.

Compare

Evaluate different searching and sorting approaches.

Algorithm First, Code Second

- Code is one implementation of a solution idea, in one particular language.
- Code written without an algorithm leads to trial and error and hidden bugs.
- The right order: understand the problem, solve an example, write the method, then code it.



The orange step (the algorithm) is language-independent; the same method can be coded in any language.

Problem Definition

Goal
Which problem are we solving?

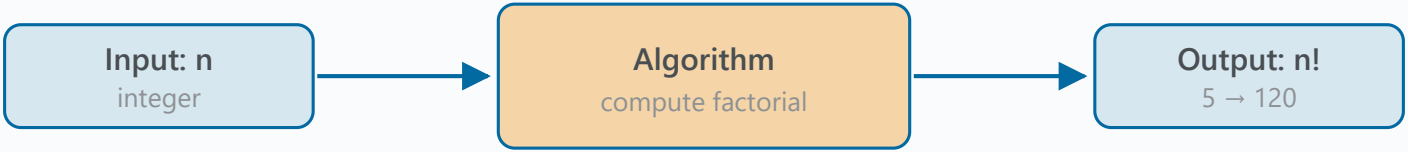
Input
What data does the program receive?

Output
What result must it produce?

Constraint
What are the valid ranges and conditions?

Example
Which small example confirms the expected result?

Error case
What happens on invalid or missing input?

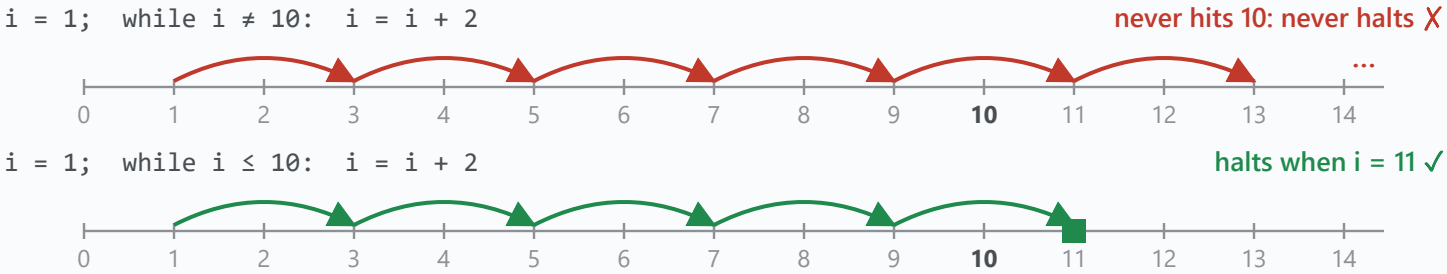


Constraint: 0 ≤ n ≤ 20 (fits in a long) · error if n < 0

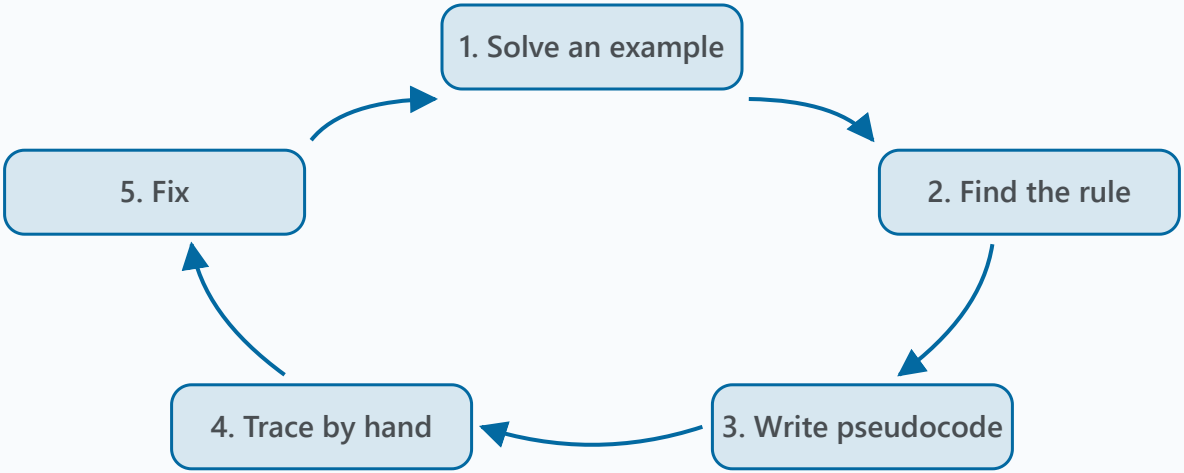
Properties of a Good Algorithm

An algorithm is a finite sequence of well-defined steps that solves a given problem.

- It takes zero or more inputs and produces at least one output.
- Its steps are unambiguous, ordered and executable.
- It halts after a finite number of steps.
- It produces the correct result for every valid input.
- Its resource use stays acceptable as the input grows.



The Algorithm Design Cycle

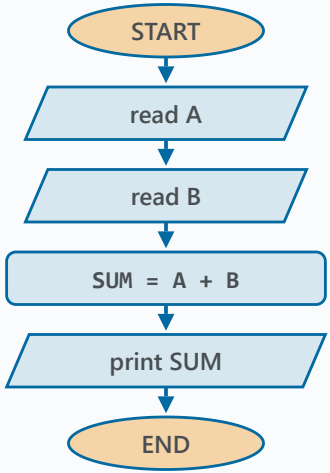


- If a hand trace reveals a bug, fix it and go around again; once no bugs remain, start coding.
- When several solutions exist, compare their correctness, readability and efficiency.

Pseudocode

Pseudocode describes an algorithm's logic without tying it to a programming language.

```
START
  read A
  read B
  SUM = A + B
  print SUM
END
```



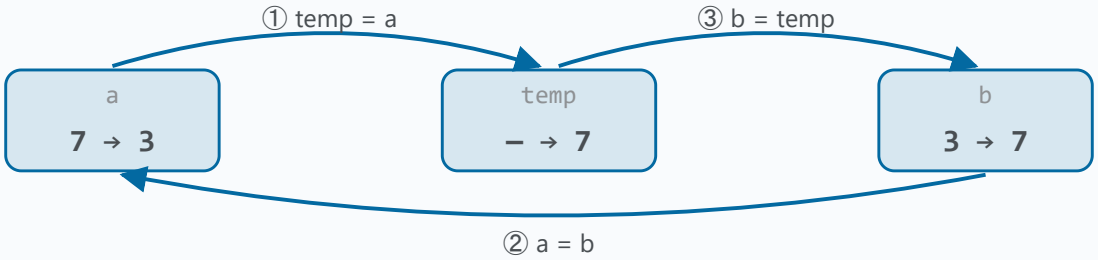
- Use short, consistent phrases close to natural language.
- Indentation shows the scope of conditions and loops.
- The point is clarity, not syntax.

EXAMPLES

Example: Sort Two Numbers

```
read a, b
if a > b then
    temp = a
    a = b
    b = temp
print a, b
```

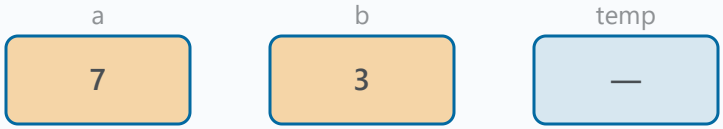
- Equal values need no swap.
- A swap takes **three assignments**.
- The output must satisfy a ≤ b.



Why `temp` ? Writing **a = b; b = a** directly overwrites a's old value in the first assignment: for a = 7, b = 3 the result is **3 3**.

EXAMPLES · STEP-BY-STEP TRACE

Sort Two Numbers Step by Step (a = 7, b = 3)



Read the inputs: **a = 7**, **b = 3**. Goal: print the smaller one first.

Step 1 / 6

```
▶ read a, b
2 if a > b then
3     temp = a
4     a = b
5     b = temp
6 print a, b
```

VARIABLES

a = 7 b = 3

OUTPUT

Sort Two Numbers Step by Step (a = 7, b = 3)



Condition: **7 > 3** → **true**, so we enter the swap block.

Step 2 / 6

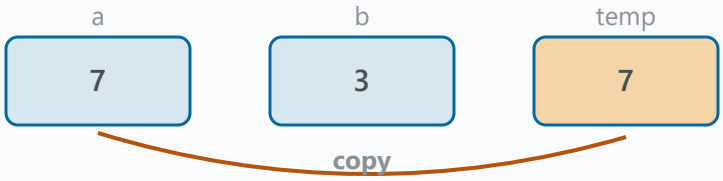
```
1 read a, b
▶ if a > b then
3     temp = a
4     a = b
5     b = temp
6 print a, b
```

VARIABLES

`a = 7` `b = 3` `a > b = true`

OUTPUT

Sort Two Numbers Step by Step (a = 7, b = 3)



temp = a = 7: back up a's value first so it isn't lost.

Step 3 / 6

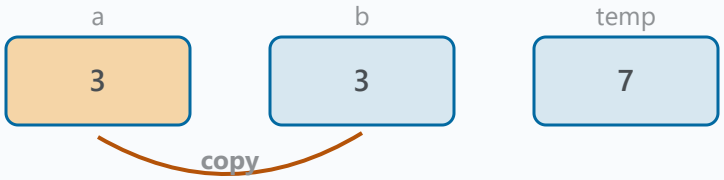
```
1 read a, b
2 if a > b then
▶     temp = a
4     a = b
5     b = temp
6 print a, b
```

VARIABLES

a = 7 | b = 3 | temp = 7

OUTPUT

Sort Two Numbers Step by Step (a = 7, b = 3)



a = b = 3. The old 7 now lives only in temp.

Step 4 / 6

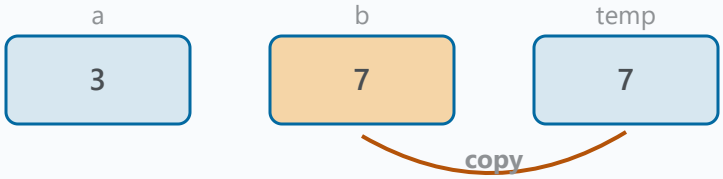
```
1 read a, b
2 if a > b then
3     temp = a
▶     a = b
5     b = temp
6 print a, b
```

VARIABLES

a = 3 | b = 3 | temp = 7

OUTPUT

Sort Two Numbers Step by Step (a = 7, b = 3)



b = temp = 7. Three assignments, and the values have swapped.

Step 5 / 6

```
1 read a, b
2 if a > b then
3     temp = a
4     a = b
▶     b = temp
6 print a, b
```

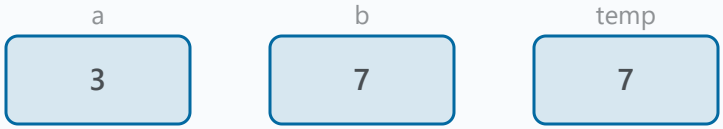
VARIABLES

a = 3 | b = 7 | temp = 7

OUTPUT

EXAMPLES · STEP-BY-STEP TRACE

Sort Two Numbers Step by Step (a = 7, b = 3)



Output **3 7**: a ≤ b holds. (With a = 2, b = 9 the condition is false and lines 3–5 are skipped.)

Step 6 / 6

```
1 read a, b
2 if a > b then
3     temp = a
4     a = b
5     b = temp
▶ print a, b
```

VARIABLES

`a = 3` `b = 7` `temp = 7`

OUTPUT

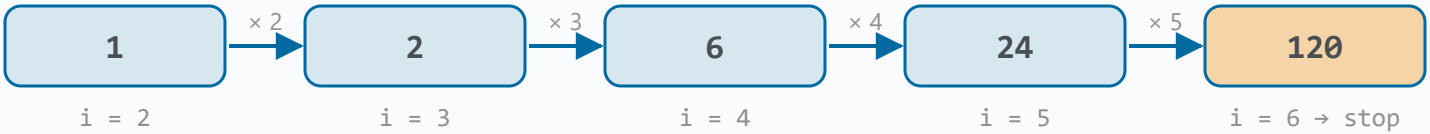
```
3 7
```

Example: Factorial

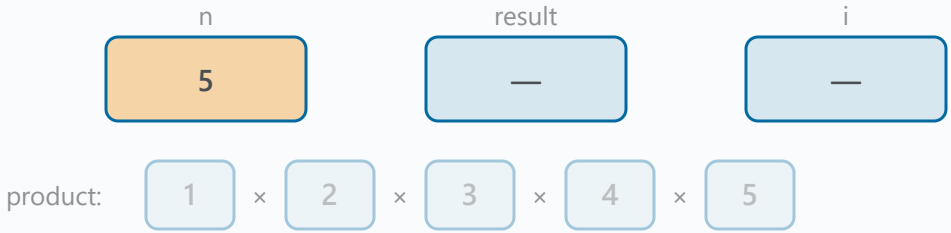
```
read n
if n < 0 then print "error" and stop
result = 1
i = 2
while i ≤ n
    result = result × i
    i = i + 1
print result
```

- 0! and 1! are 1: the loop never runs and result stays 1.
- Loop invariant: at the start of each round, **result = (i − 1)!**
- On negative input, report an error and stop.

n = 5: at the start of each round, result = (i − 1)!



Factorial Step by Step (n = 5)



Read **n = 5**. Expected: 5! = 1×2×3×4×5.

Step 1 / 18

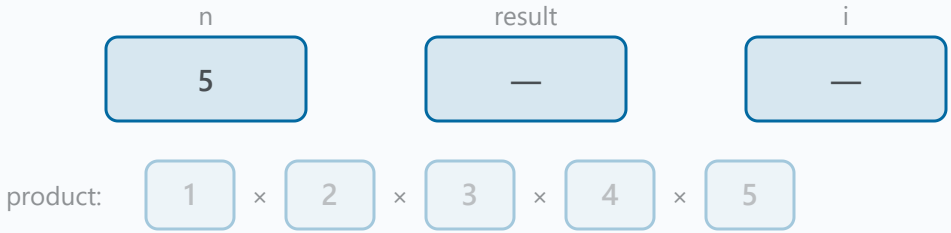
```
▶ read n
2 if n < 0 then print "error" and stop
3 result = 1
4 i = 2
5 while i ≤ n
6     result = result × i
7     i = i + 1
8 print result
```

VARIABLES

n = **5**

OUTPUT

Factorial Step by Step (n = 5)



Condition: **5 < 0** → **false**; no error, continue.

Step 2 / 18

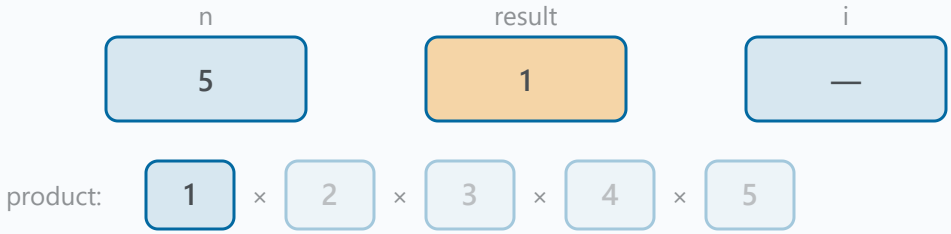
```
1 read n
▶ if n < 0 then print "error" and stop
3 result = 1
4 i = 2
5 while i ≤ n
6     result = result × i
7     i = i + 1
8 print result
```

VARIABLES

`n = 5`

OUTPUT

Factorial Step by Step (n = 5)



A product starts at **1** (starting at 0 would make everything 0).

Step 3 / 18

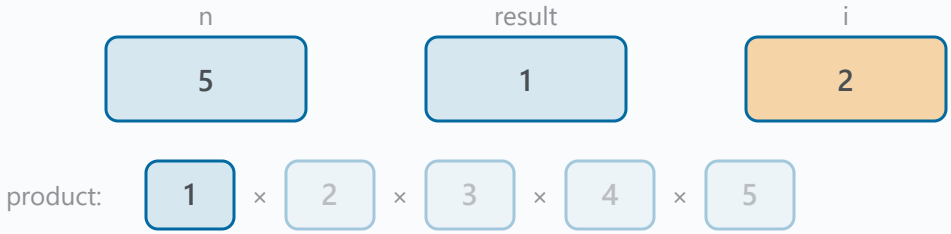
```
1 read n
2 if n < 0 then print "error" and stop
▶ result = 1
4 i = 2
5 while i ≤ n
6     result = result × i
7     i = i + 1
8 print result
```

VARIABLES

`n = 5` `result = 1`

OUTPUT

Factorial Step by Step (n = 5)



i = 2: multiplying by 1 changes nothing, so start at 2.

Step 4 / 18

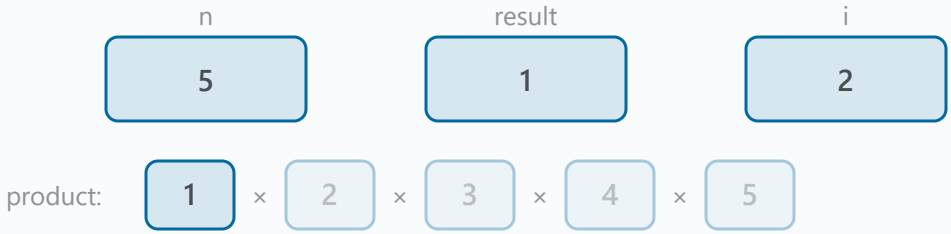
```
1 read n
2 if n < 0 then print "error" and stop
3 result = 1
▶ i = 2
5 while i ≤ n
6     result = result × i
7     i = i + 1
8 print result
```

VARIABLES

`n = 5` `result = 1` `i = 2`

OUTPUT

Factorial Step by Step (n = 5)



Condition: **2 ≤ 5** → **true**.

Step 5 / 18

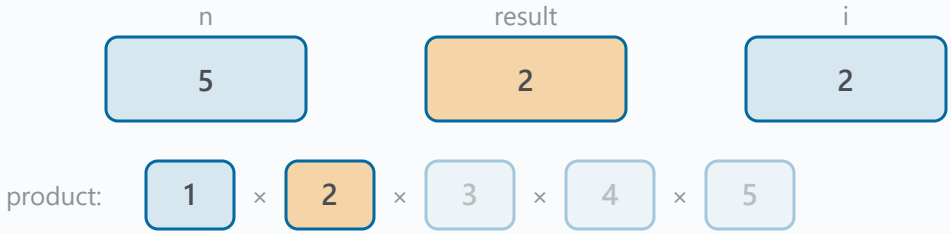
```
1 read n
2 if n < 0 then print "error" and stop
3 result = 1
4 i = 2
▶ while i ≤ n
6     result = result × i
7     i = i + 1
8 print result
```

VARIABLES

`n = 5` `result = 1` `i = 2`

OUTPUT

Factorial Step by Step (n = 5)



result = 1 × 2 = 2 (= 2!).

Step 6 / 18

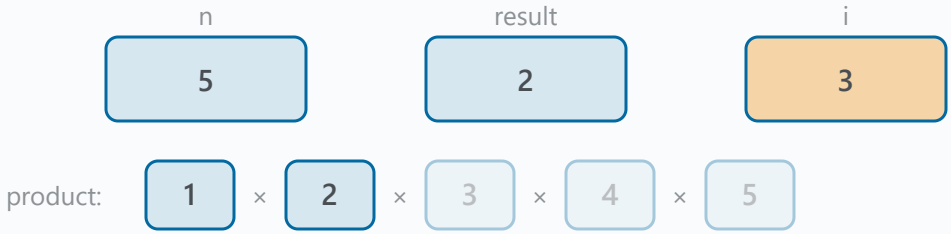
```
1 read n
2 if n < 0 then print "error" and stop
3 result = 1
4 i = 2
5 while i ≤ n
      result = result × i
7     i = i + 1
8 print result
```

VARIABLES

n = 5 | result = 2 | i = 2

OUTPUT

Factorial Step by Step (n = 5)



i = 3.

Step 7 / 18

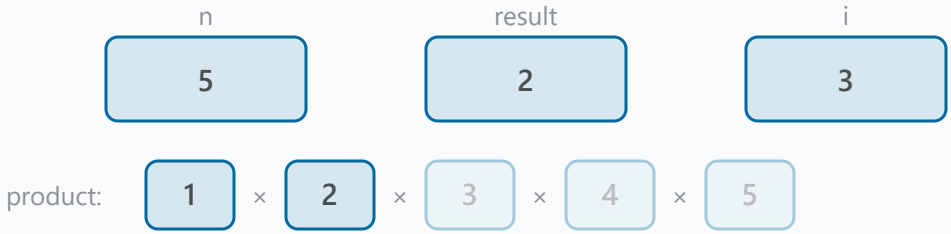
```
1 read n
2 if n < 0 then print "error" and stop
3 result = 1
4 i = 2
5 while i ≤ n
6     result = result × i
▶     i = i + 1
8 print result
```

VARIABLES

`n = 5` `result = 2` `i = 3`

OUTPUT

Factorial Step by Step (n = 5)



Condition: **3 ≤ 5 → true**.

Step 8 / 18

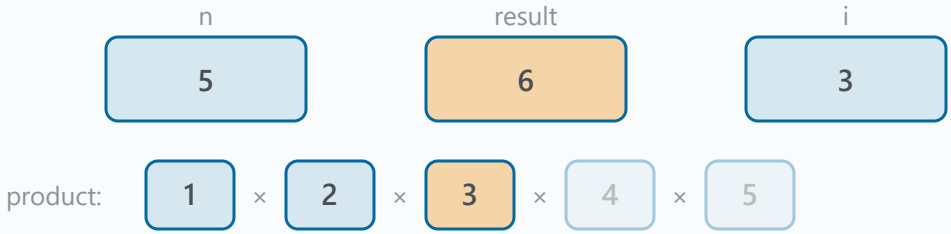
```
1 read n
2 if n < 0 then print "error" and stop
3 result = 1
4 i = 2
▶ while i ≤ n
6     result = result × i
7     i = i + 1
8 print result
```

VARIABLES

`n = 5` `result = 2` `i = 3`

OUTPUT

Factorial Step by Step (n = 5)



result = 2 × 3 = 6 (= 3!).

Step 9 / 18

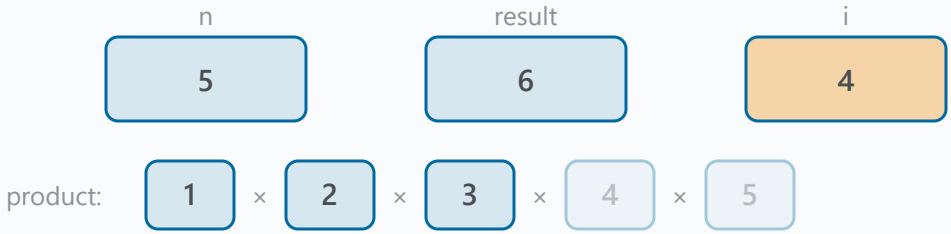
```
1 read n
2 if n < 0 then print "error" and stop
3 result = 1
4 i = 2
5 while i ≤ n
      result = result × i
7     i = i + 1
8 print result
```

VARIABLES

`n = 5` `result = 6` `i = 3`

OUTPUT

Factorial Step by Step (n = 5)



i = 4.

Step 10 / 18

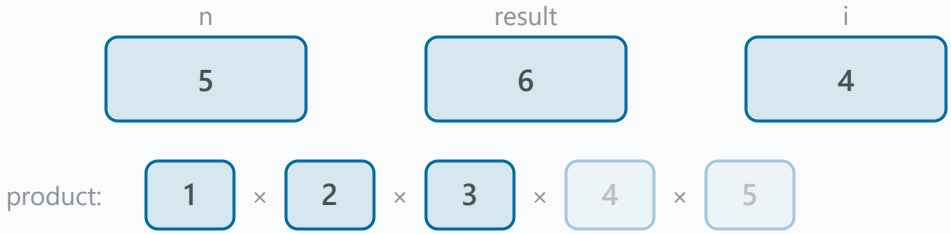
```
1 read n
2 if n < 0 then print "error" and stop
3 result = 1
4 i = 2
5 while i ≤ n
6     result = result × i
▶     i = i + 1
8 print result
```

VARIABLES

`n = 5` `result = 6` `i = 4`

OUTPUT

Factorial Step by Step (n = 5)



Condition: **4 ≤ 5** → **true**.

Step 11 / 18

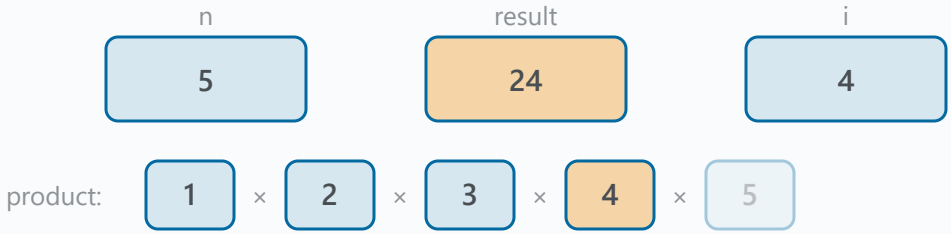
```
1 read n
2 if n < 0 then print "error" and stop
3 result = 1
4 i = 2
▶ while i ≤ n
6     result = result × i
7     i = i + 1
8 print result
```

VARIABLES

`n = 5` `result = 6` `i = 4`

OUTPUT

Factorial Step by Step (n = 5)



result = 6 × 4 = 24 (= 4!).

Step 12 / 18

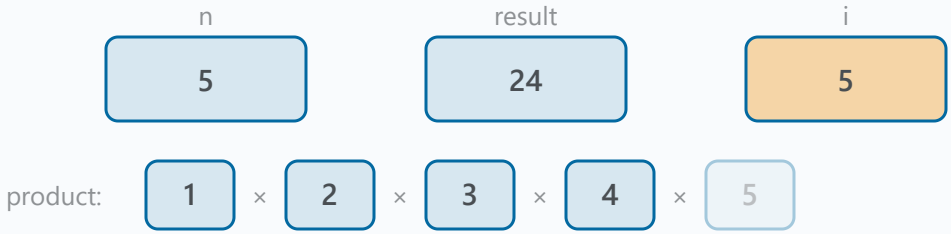
```
1 read n
2 if n < 0 then print "error" and stop
3 result = 1
4 i = 2
5 while i ≤ n
      result = result × i
7     i = i + 1
8 print result
```

VARIABLES

`n = 5` `result = 24` `i = 4`

OUTPUT

Factorial Step by Step (n = 5)



i = 5.

Step 13 / 18

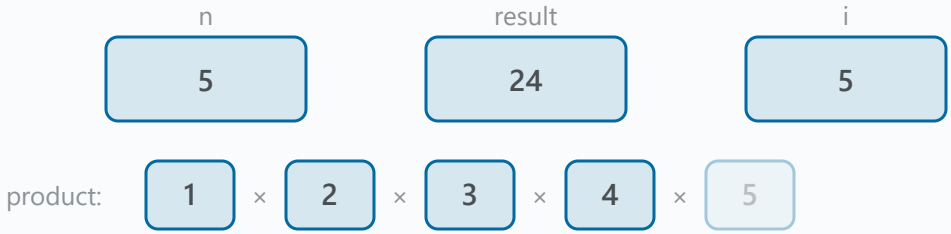
```
1 read n
2 if n < 0 then print "error" and stop
3 result = 1
4 i = 2
5 while i ≤ n
6     result = result × i
▶     i = i + 1
8 print result
```

VARIABLES

`n = 5` `result = 24` `i = 5`

OUTPUT

Factorial Step by Step (n = 5)



Condition: **5 ≤ 5** → **true**.

Step 14 / 18

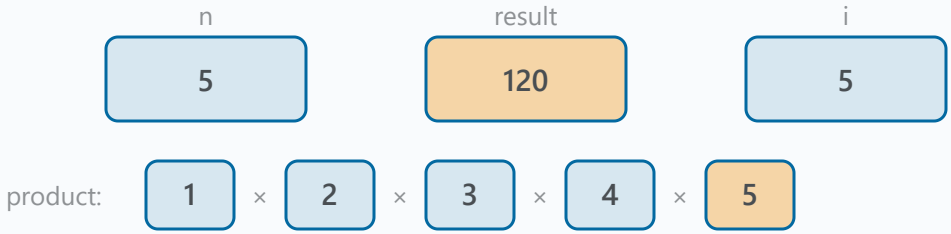
```
1 read n
2 if n < 0 then print "error" and stop
3 result = 1
4 i = 2
▶ while i ≤ n
6     result = result × i
7     i = i + 1
8 print result
```

VARIABLES

`n = 5` `result = 24` `i = 5`

OUTPUT

Factorial Step by Step (n = 5)



result = 24 × 5 = 120 (= 5!).

Step 15 / 18

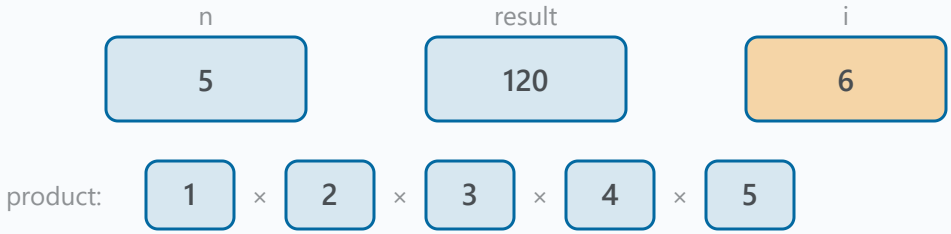
```
1 read n
2 if n < 0 then print "error" and stop
3 result = 1
4 i = 2
5 while i ≤ n
      result = result × i
7     i = i + 1
8 print result
```

VARIABLES

`n = 5` `result = 120` `i = 5`

OUTPUT

Factorial Step by Step (n = 5)



i = 6.

Step 16 / 18

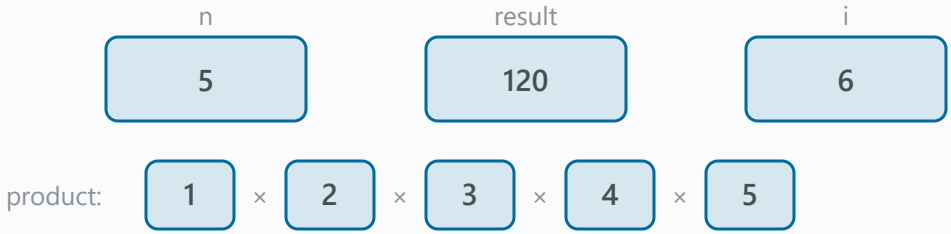
```
1 read n
2 if n < 0 then print "error" and stop
3 result = 1
4 i = 2
5 while i ≤ n
6     result = result × i
▶     i = i + 1
8 print result
```

VARIABLES

`n = 5` `result = 120` `i = 6`

OUTPUT

Factorial Step by Step (n = 5)



Condition: **6 ≤ 5 → false**, so the loop ends.

Step 17 / 18

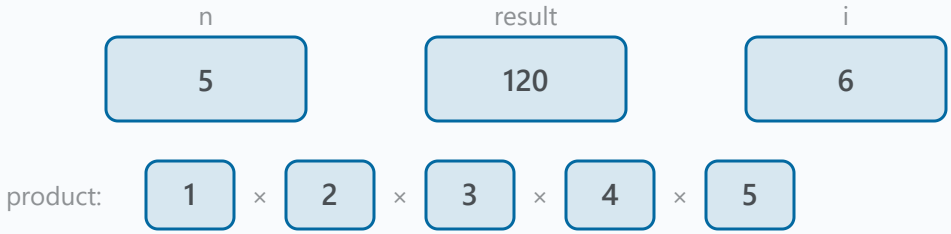
```
1 read n
2 if n < 0 then print "error" and stop
3 result = 1
4 i = 2
▶ while i ≤ n
6     result = result × i
7     i = i + 1
8 print result
```

VARIABLES

`n = 5` `result = 120` `i = 6`

OUTPUT

Factorial Step by Step (n = 5)



Print **120**. The loop ran n − 1 = 4 times.

Step 18 / 18

```
1 read n
2 if n < 0 then print "error" and stop
3 result = 1
4 i = 2
5 while i ≤ n
6     result = result × i
7     i = i + 1
▶ print result
```

VARIABLES

`n = 5` `result = 120` `i = 6`

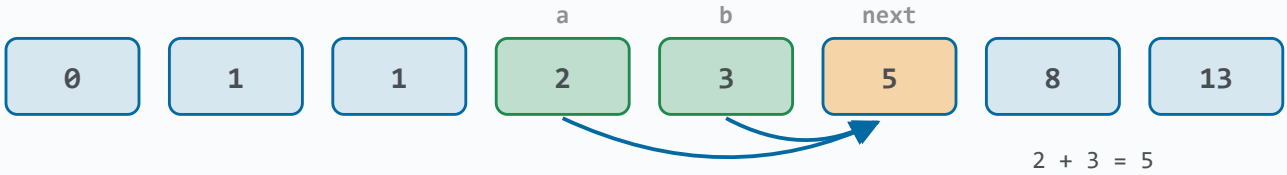
OUTPUT

```
120
```

Example: Fibonacci

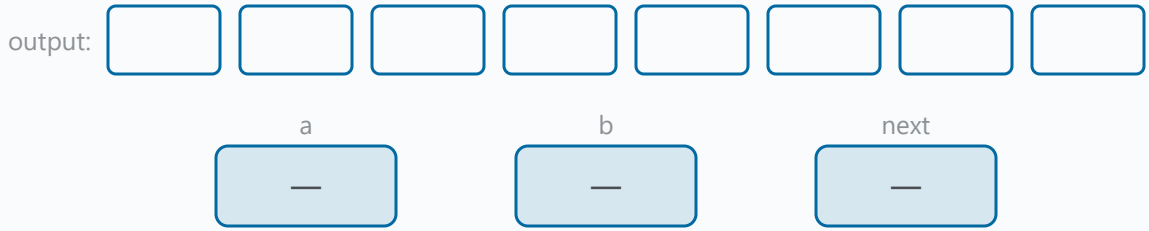
```
read n
a = 0, b = 1
for round = 0 to n-1
    print a
    next = a + b
    a = b
    b = next
```

- Keep the two previous values: a and b form a window that slides one step right each round.
- The temporary variable prevents losing information during the update.



Round 3: print a (2), next = a + b; then a = 3, b = 5 and the window slides one step right.

Fibonacci Step by Step (n = 8, rounds 0–2)



n = 8: print the first 8 Fibonacci numbers.

Step 1 / 14

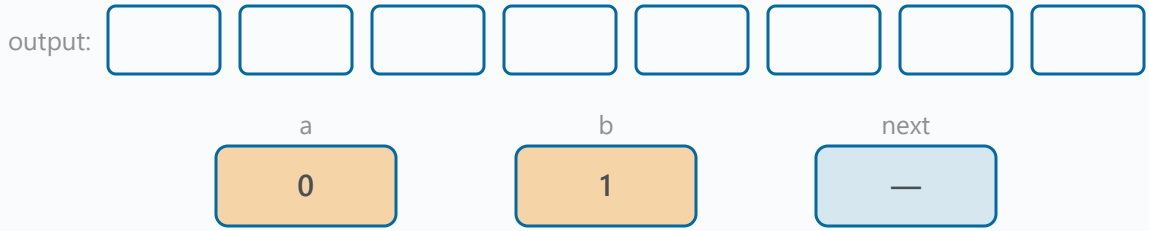
```
▶ read n
2 a = 0, b = 1
3 for round = 0 to n-1
4     print a
5     next = a + b
6     a = b
7     b = next
```

VARIABLES

`n = 8`

OUTPUT

Fibonacci Step by Step (n = 8, rounds 0–2)



The first two terms: **a = 0**, **b = 1**.

Step 2 / 14

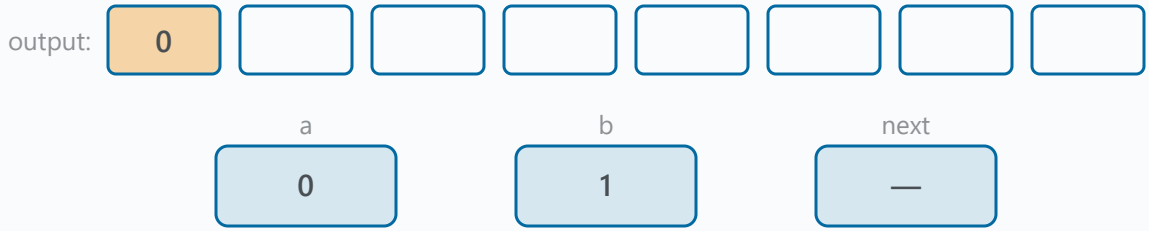
```
1 read n
▶ a = 0, b = 1
3 for round = 0 to n-1
4     print a
5     next = a + b
6     a = b
7     b = next
```

VARIABLES

`n = 8` `a = 0` `b = 1`

OUTPUT

Fibonacci Step by Step (n = 8, rounds 0–2)



Round 0 (0…7): print **a = 0**.

Step 3 / 14

```
1 read n
2 a = 0, b = 1
▶ for round = 0 to n-1
▶     print a
5     next = a + b
6     a = b
7     b = next
```

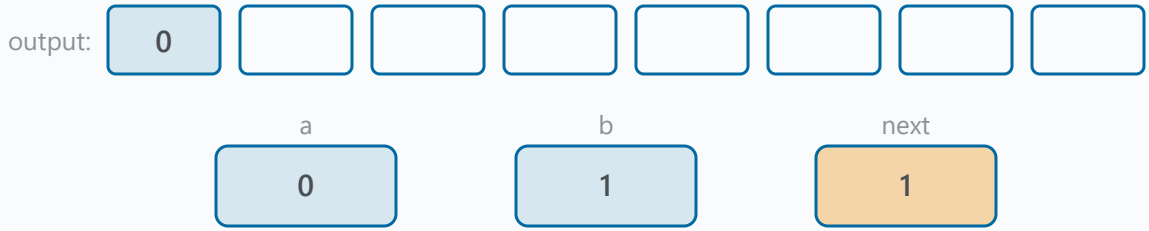
VARIABLES

`n = 8` `round = 0` `a = 0` `b = 1` `next = –`

OUTPUT

```
0
```

Fibonacci Step by Step (n = 8, rounds 0–2)



next = 0 + 1 = 1.

Step 4 / 14

```
1 read n
2 a = 0, b = 1
3 for round = 0 to n-1
4     print a
      next = a + b
6     a = b
7     b = next
```

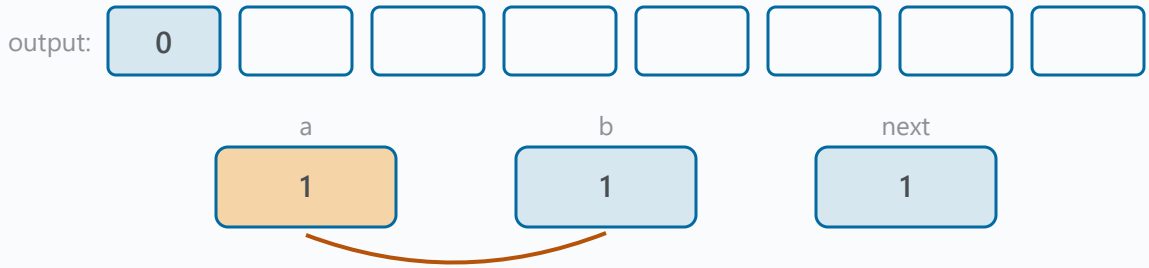
VARIABLES

`n = 8` `round = 0` `a = 0` `b = 1` `next = 1`

OUTPUT

```
0
```

Fibonacci Step by Step (n = 8, rounds 0–2)



a = b = 1: the window slides.

Step 5 / 14

```
1 read n
2 a = 0, b = 1
3 for round = 0 to n-1
4     print a
5     next = a + b
▶     a = b
7     b = next
```

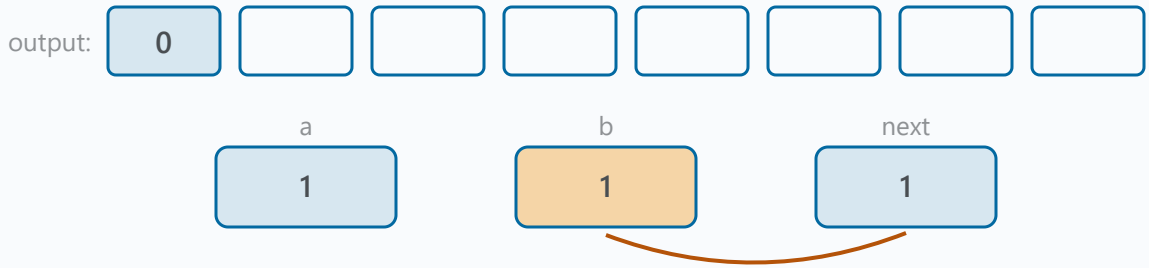
VARIABLES

`n = 8` `round = 0` `a = 1` `b = 1` `next = 1`

OUTPUT

```
0
```

Fibonacci Step by Step (n = 8, rounds 0–2)



b = next = 1. Order matters: without a = b first, the old b would be lost.

Step 6 / 14

```
1 read n
2 a = 0, b = 1
3 for round = 0 to n-1
4     print a
5     next = a + b
6     a = b
▶     b = next
```

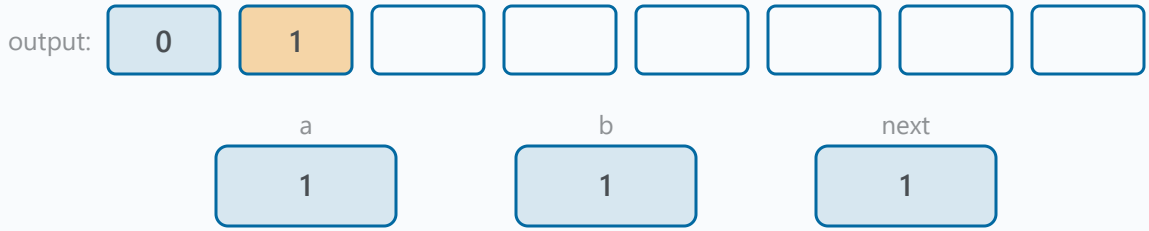
VARIABLES

`n = 8` `round = 0` `a = 1` `b = 1` `next = 1`

OUTPUT

```
0
```

Fibonacci Step by Step (n = 8, rounds 0–2)



Round 1 (0...7): print **a = 1**.

Step 7 / 14

```
1 read n
2 a = 0, b = 1
▶ for round = 0 to n-1
▶     print a
5     next = a + b
6     a = b
7     b = next
```

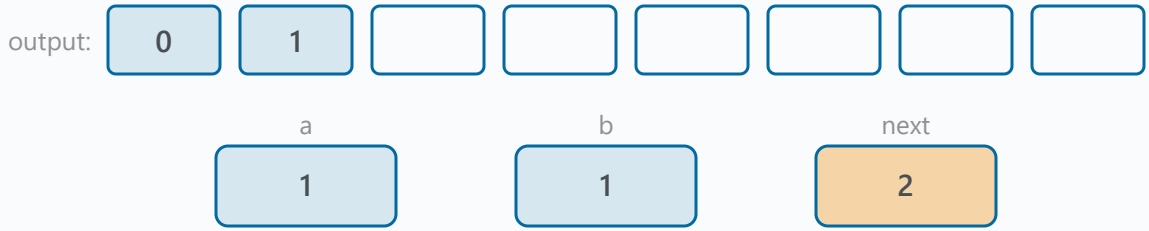
VARIABLES

n = 8 | round = 1 | a = 1 | b = 1 | next = 1

OUTPUT

```
0 1
```

Fibonacci Step by Step (n = 8, rounds 0–2)



next = 1 + 1 = 2.

Step 8 / 14

```
1 read n
2 a = 0, b = 1
3 for round = 0 to n-1
4     print a
      next = a + b
6     a = b
7     b = next
```

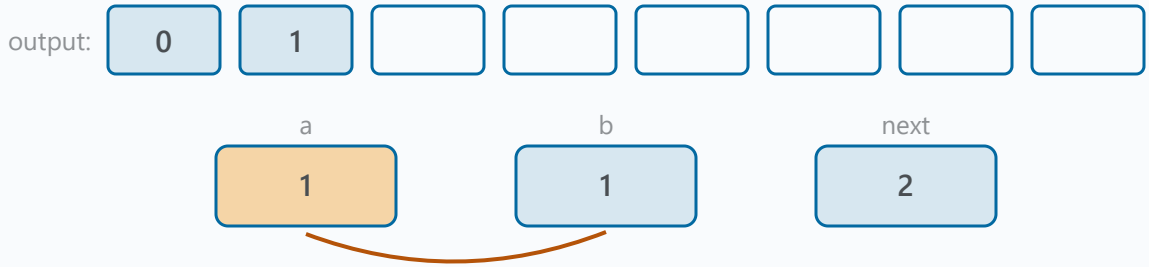
VARIABLES

`n = 8` `round = 1` `a = 1` `b = 1` `next = 2`

OUTPUT

```
0 1
```

Fibonacci Step by Step (n = 8, rounds 0–2)



a = b = 1: the window slides.

Step 9 / 14

```
1 read n
2 a = 0, b = 1
3 for round = 0 to n-1
4     print a
5     next = a + b
▶     a = b
7     b = next
```

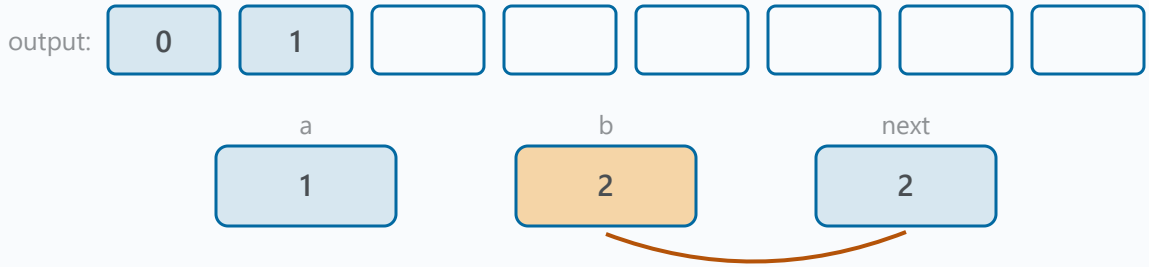
VARIABLES

`n = 8` `round = 1` `a = 1` `b = 1` `next = 2`

OUTPUT

```
0 1
```

Fibonacci Step by Step (n = 8, rounds 0–2)



b = next = 2.

Step 10 / 14

```
1 read n
2 a = 0, b = 1
3 for round = 0 to n-1
4     print a
5     next = a + b
6     a = b
▶     b = next
```

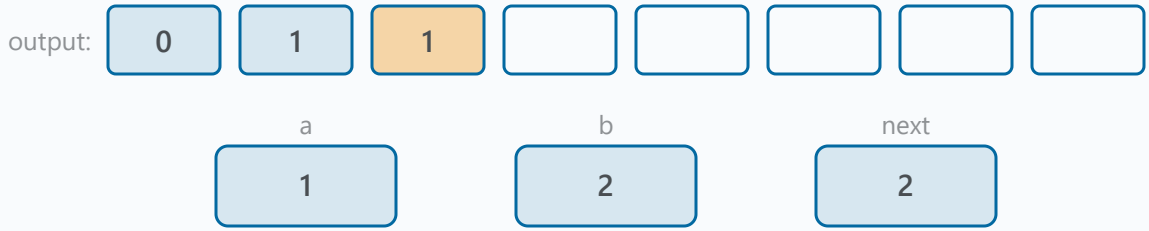
VARIABLES

n = 8 | round = 1 | a = 1 | b = 2 | next = 2

OUTPUT

```
0 1
```

Fibonacci Step by Step (n = 8, rounds 0–2)



Round 2 (0...7): print **a = 1**.

Step 11 / 14

```
1 read n
2 a = 0, b = 1
▶ for round = 0 to n-1
▶     print a
5     next = a + b
6     a = b
7     b = next
```

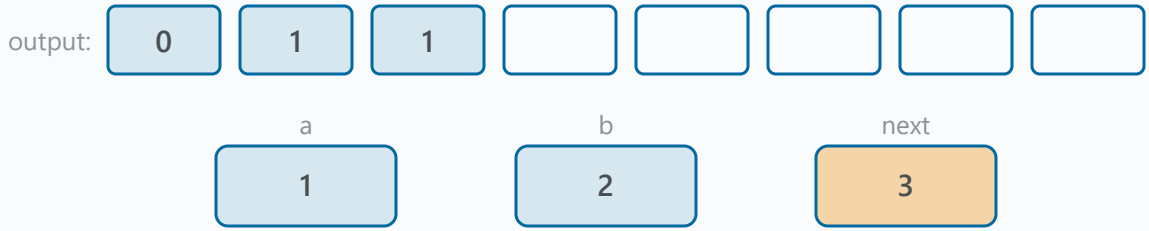
VARIABLES

n = 8 | round = 2 | a = 1 | b = 2 | next = 2

OUTPUT

```
0 1 1
```

Fibonacci Step by Step (n = 8, rounds 0–2)



next = 1 + 2 = 3.

Step 12 / 14

```
1 read n
2 a = 0, b = 1
3 for round = 0 to n-1
4     print a
▶     next = a + b
6     a = b
7     b = next
```

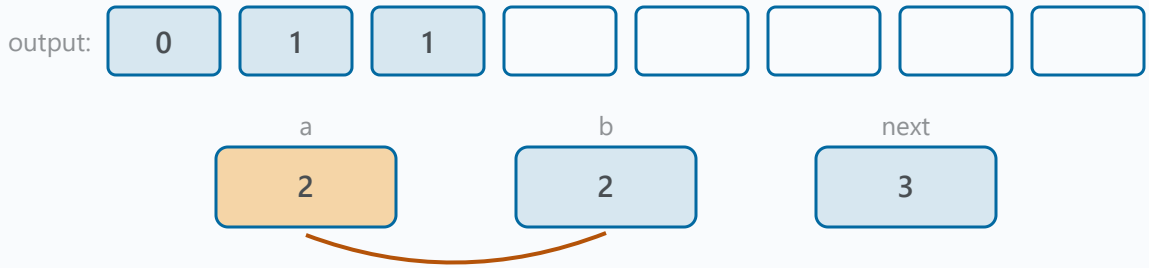
VARIABLES

`n = 8` `round = 2` `a = 1` `b = 2` `next = 3`

OUTPUT

```
0 1 1
```

Fibonacci Step by Step (n = 8, rounds 0–2)



a = b = 2: the window slides.

Step 13 / 14

```
1 read n
2 a = 0, b = 1
3 for round = 0 to n-1
4     print a
5     next = a + b
▶     a = b
7     b = next
```

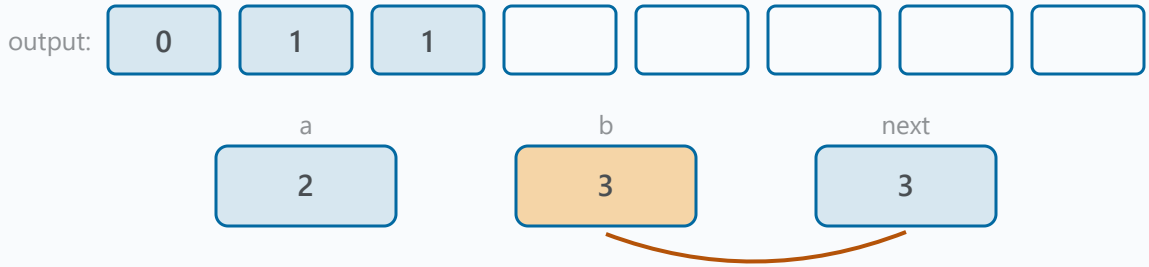
VARIABLES

n = 8 | round = 2 | a = 2 | b = 2 | next = 3

OUTPUT

```
0 1 1
```

Fibonacci Step by Step (n = 8, rounds 0–2)



b = next = 3.

Step 14 / 14

```
1 read n
2 a = 0, b = 1
3 for round = 0 to n-1
4     print a
5     next = a + b
6     a = b
▶     b = next
```

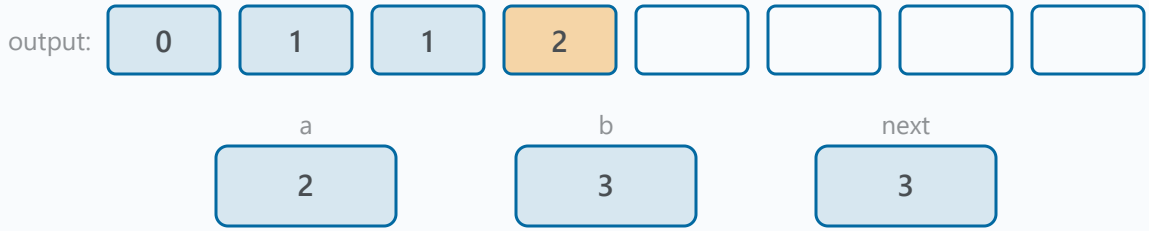
VARIABLES

n = 8 | round = 2 | a = 2 | b = 3 | next = 3

OUTPUT

```
0 1 1
```

Fibonacci Step by Step (cont.: rounds 3–5)



Round 3 (0…7): print **a = 2**.

Step 1 / 12

```
1 read n
2 a = 0, b = 1
▶ for round = 0 to n-1
▶     print a
5     next = a + b
6     a = b
7     b = next
```

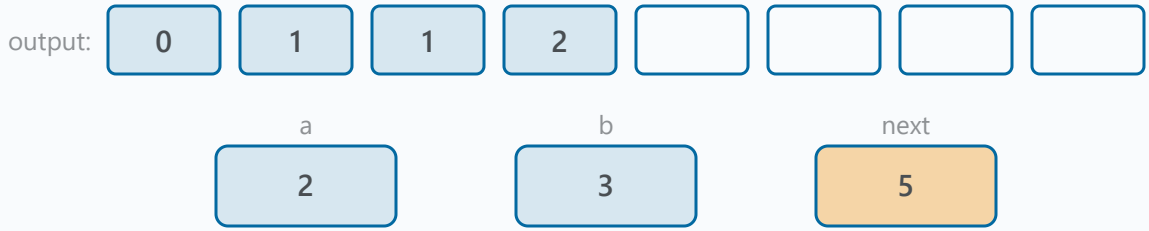
VARIABLES

`n = 8` `round = 3` `a = 2` `b = 3` `next = 3`

OUTPUT

```
0 1 1 2
```

Fibonacci Step by Step (cont.: rounds 3–5)



next = 2 + 3 = 5.

Step 2 / 12

```
1 read n
2 a = 0, b = 1
3 for round = 0 to n-1
4     print a
▶     next = a + b
6     a = b
7     b = next
```

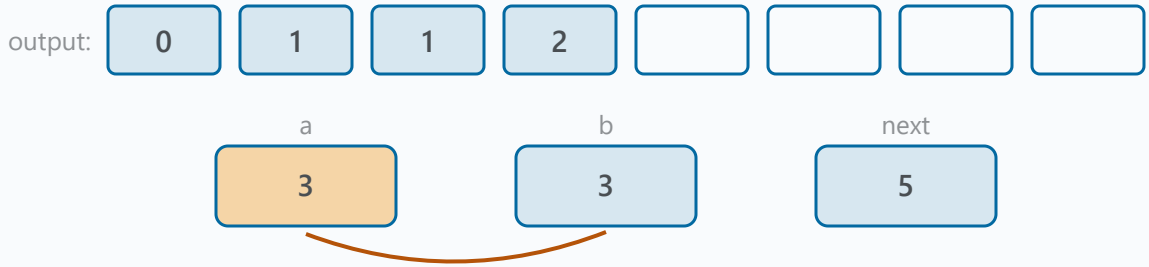
VARIABLES

`n = 8` `round = 3` `a = 2` `b = 3` `next = 5`

OUTPUT

```
0 1 1 2
```

Fibonacci Step by Step (cont.: rounds 3–5)



a = b = 3: the window slides.

Step 3 / 12

```
1 read n
2 a = 0, b = 1
3 for round = 0 to n-1
4     print a
5     next = a + b
▶     a = b
7     b = next
```

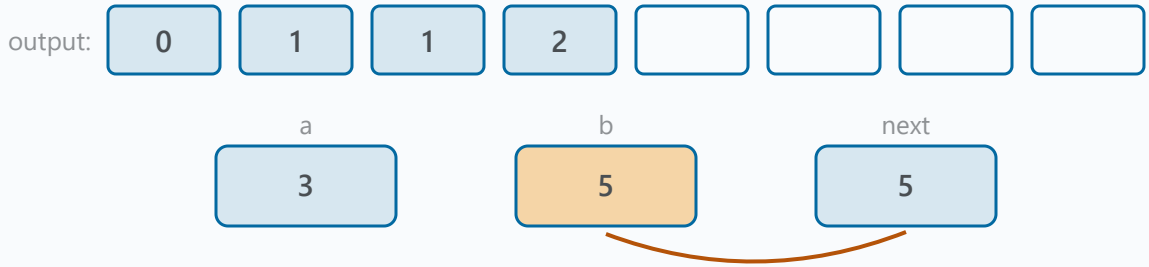
VARIABLES

`n = 8` `round = 3` `a = 3` `b = 3` `next = 5`

OUTPUT

```
0 1 1 2
```

Fibonacci Step by Step (cont.: rounds 3–5)



b = next = 5.

Step 4 / 12

```
1 read n
2 a = 0, b = 1
3 for round = 0 to n-1
4     print a
5     next = a + b
6     a = b
▶     b = next
```

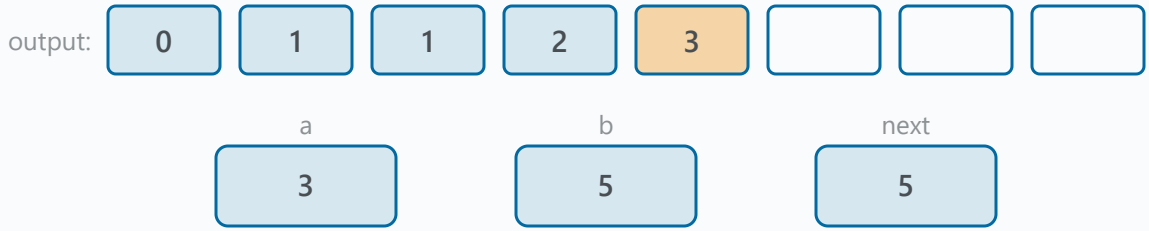
VARIABLES

`n = 8` `round = 3` `a = 3` `b = 5` `next = 5`

OUTPUT

```
0 1 1 2
```

Fibonacci Step by Step (cont.: rounds 3–5)



Round 4 (0…7): print **a = 3**.

Step 5 / 12

```
1 read n
2 a = 0, b = 1
▶ for round = 0 to n-1
▶     print a
5     next = a + b
6     a = b
7     b = next
```

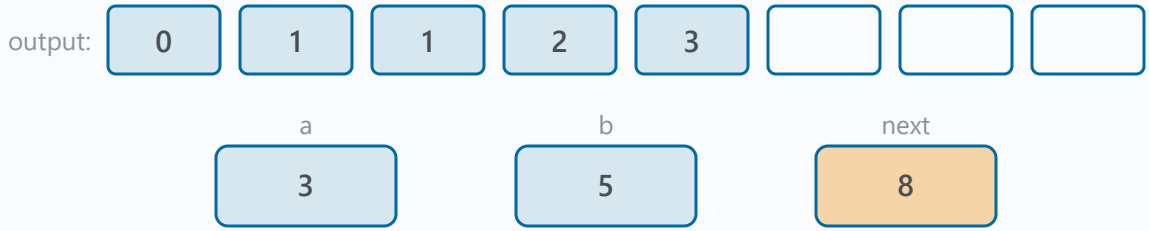
VARIABLES

`n = 8` `round = 4` `a = 3` `b = 5` `next = 5`

OUTPUT

```
0 1 1 2 3
```

Fibonacci Step by Step (cont.: rounds 3–5)



next = 3 + 5 = 8.

Step 6 / 12

```
1 read n
2 a = 0, b = 1
3 for round = 0 to n-1
4     print a
      next = a + b
6     a = b
7     b = next
```

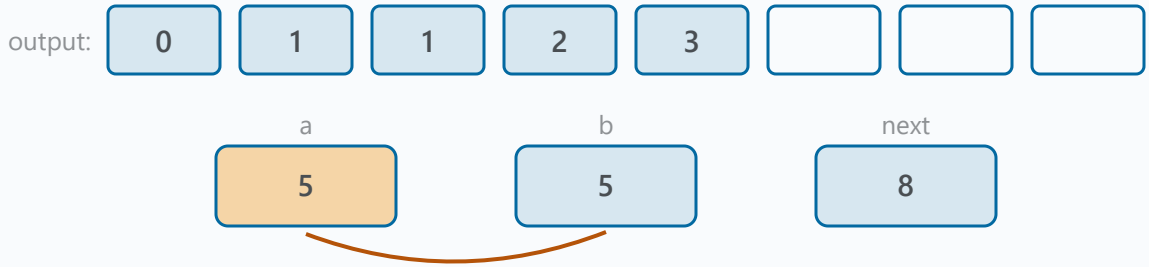
VARIABLES

`n = 8` `round = 4` `a = 3` `b = 5` `next = 8`

OUTPUT

```
0 1 1 2 3
```

Fibonacci Step by Step (cont.: rounds 3–5)



a = b = 5: the window slides.

Step 7 / 12

```
1 read n
2 a = 0, b = 1
3 for round = 0 to n-1
4     print a
5     next = a + b
▶     a = b
7     b = next
```

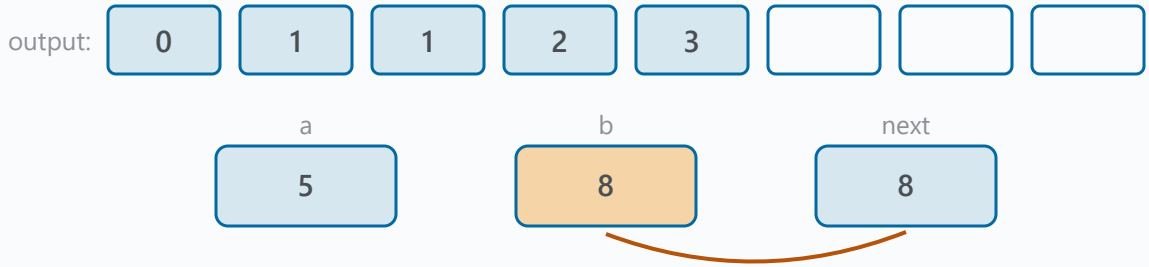
VARIABLES

`n = 8` `round = 4` `a = 5` `b = 5` `next = 8`

OUTPUT

```
0 1 1 2 3
```

Fibonacci Step by Step (cont.: rounds 3–5)



b = next = 8.

Step 8 / 12

```
1 read n
2 a = 0, b = 1
3 for round = 0 to n-1
4     print a
5     next = a + b
6     a = b
▶     b = next
```

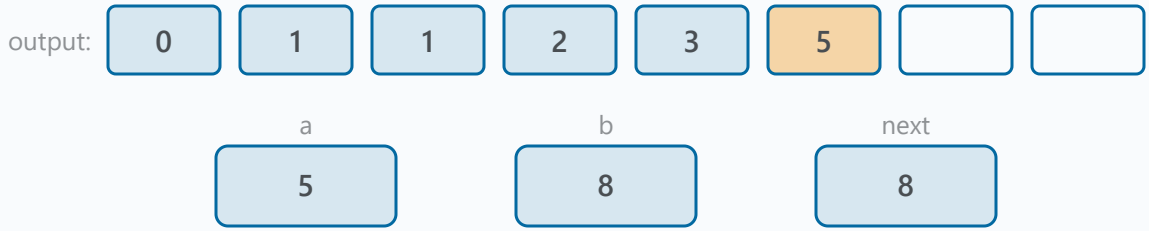
VARIABLES

`n = 8` `round = 4` `a = 5` `b = 8` `next = 8`

OUTPUT

```
0 1 1 2 3
```

Fibonacci Step by Step (cont.: rounds 3–5)



Round 5 (0…7): print **a = 5**.

Step 9 / 12

```
1 read n
2 a = 0, b = 1
▶ for round = 0 to n-1
▶     print a
5     next = a + b
6     a = b
7     b = next
```

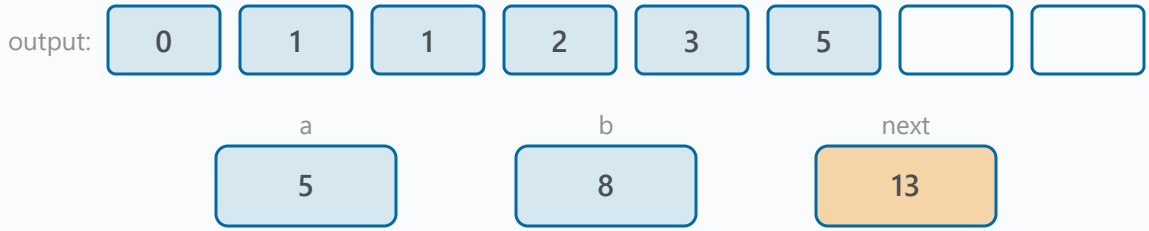
VARIABLES

`n = 8` `round = 5` `a = 5` `b = 8` `next = 8`

OUTPUT

```
0 1 1 2 3 5
```

Fibonacci Step by Step (cont.: rounds 3–5)



next = 5 + 8 = 13.

Step 10 / 12

```
1 read n
2 a = 0, b = 1
3 for round = 0 to n-1
4     print a
      next = a + b
6     a = b
7     b = next
```

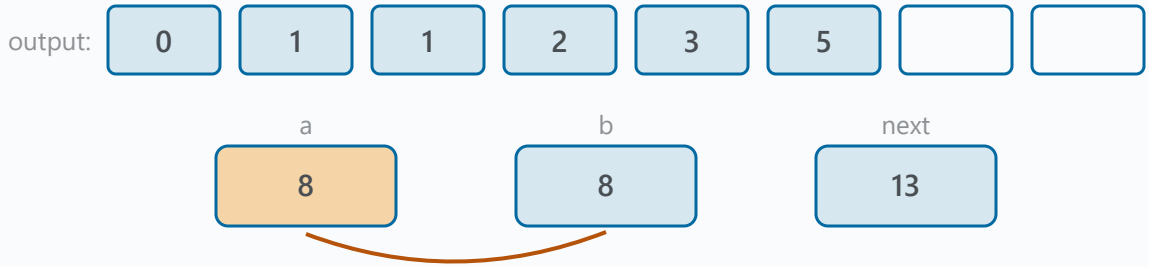
VARIABLES

`n = 8` `round = 5` `a = 5` `b = 8` `next = 13`

OUTPUT

```
0 1 1 2 3 5
```

Fibonacci Step by Step (cont.: rounds 3–5)



a = b = 8: the window slides.

Step 11 / 12

```
1 read n
2 a = 0, b = 1
3 for round = 0 to n-1
4     print a
5     next = a + b
▶     a = b
7     b = next
```

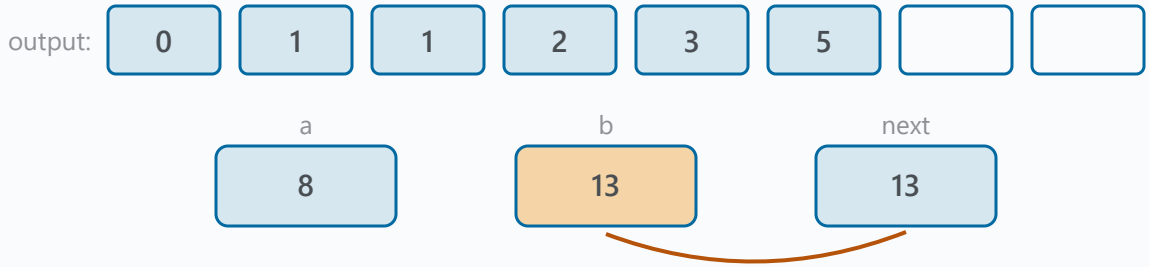
VARIABLES

`n = 8` `round = 5` `a = 8` `b = 8` `next = 13`

OUTPUT

```
0 1 1 2 3 5
```

Fibonacci Step by Step (cont.: rounds 3–5)



b = next = 13.

Step 12 / 12

```
1 read n
2 a = 0, b = 1
3 for round = 0 to n-1
4     print a
5     next = a + b
6     a = b
▶     b = next
```

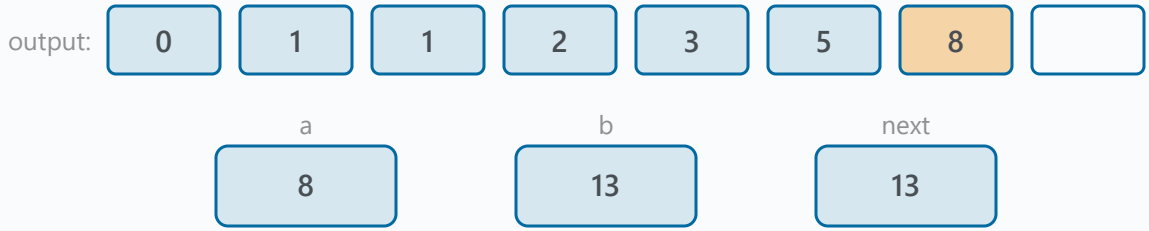
VARIABLES

n = 8 | round = 5 | a = 8 | b = 13 | next = 13

OUTPUT

```
0 1 1 2 3 5
```

Fibonacci Step by Step (cont.: rounds 6–7)



Round 6 (0…7): print **a = 8**.

Step 1 / 9

```
1 read n
2 a = 0, b = 1
▶ for round = 0 to n-1
▶     print a
5     next = a + b
6     a = b
7     b = next
```

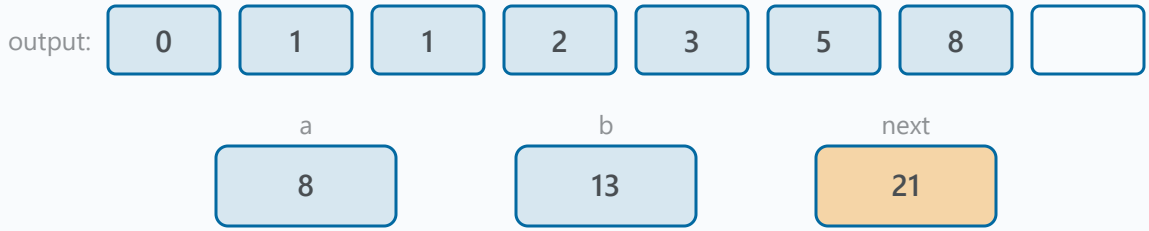
VARIABLES

`n = 8` `round = 6` `a = 8` `b = 13` `next = 13`

OUTPUT

```
0 1 1 2 3 5 8
```

Fibonacci Step by Step (cont.: rounds 6–7)



next = 8 + 13 = 21.

Step 2 / 9

```
1 read n
2 a = 0, b = 1
3 for round = 0 to n-1
4     print a
      next = a + b
6     a = b
7     b = next
```

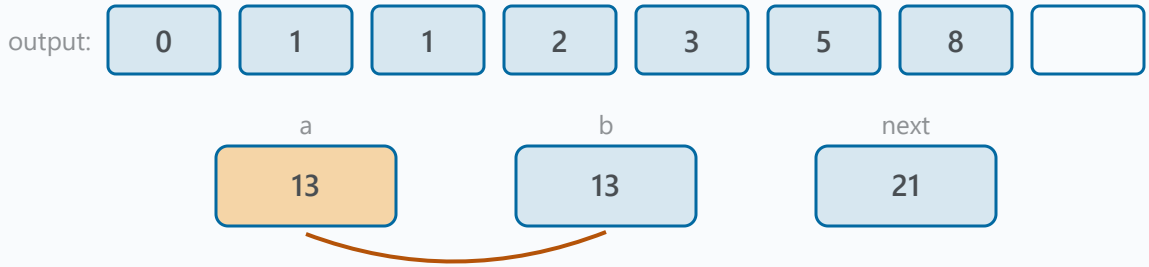
VARIABLES

`n = 8` `round = 6` `a = 8` `b = 13` `next = 21`

OUTPUT

```
0 1 1 2 3 5 8
```

Fibonacci Step by Step (cont.: rounds 6–7)



a = b = 13: the window slides.

Step 3 / 9

```
1 read n
2 a = 0, b = 1
3 for round = 0 to n-1
4     print a
5     next = a + b
▶     a = b
7     b = next
```

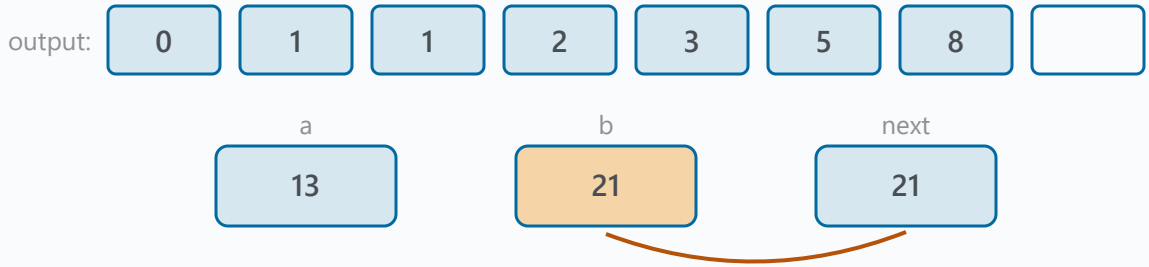
VARIABLES

`n = 8` `round = 6` `a = 13` `b = 13` `next = 21`

OUTPUT

```
0 1 1 2 3 5 8
```

Fibonacci Step by Step (cont.: rounds 6–7)



b = next = 21.

Step 4 / 9

```
1 read n
2 a = 0, b = 1
3 for round = 0 to n-1
4     print a
5     next = a + b
6     a = b
▶     b = next
```

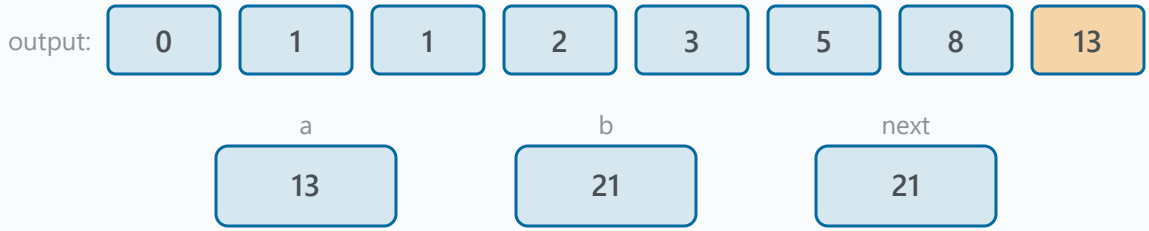
VARIABLES

`n = 8` `round = 6` `a = 13` `b = 21` `next = 21`

OUTPUT

```
0 1 1 2 3 5 8
```

Fibonacci Step by Step (cont.: rounds 6–7)



Round 7 (0…7): print **a = 13**.

Step 5 / 9

```
1 read n
2 a = 0, b = 1
▶ for round = 0 to n-1
▶     print a
5     next = a + b
6     a = b
7     b = next
```

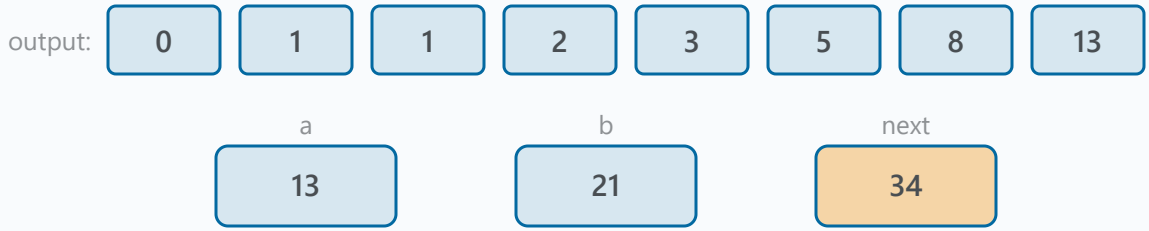
VARIABLES

n = 8 | round = 7 | a = 13 | b = 21 | next = 21

OUTPUT

```
0 1 1 2 3 5 8 13
```

Fibonacci Step by Step (cont.: rounds 6–7)



next = 13 + 21 = 34.

Step 6 / 9

```
1 read n
2 a = 0, b = 1
3 for round = 0 to n-1
4     print a
      next = a + b
6     a = b
7     b = next
```

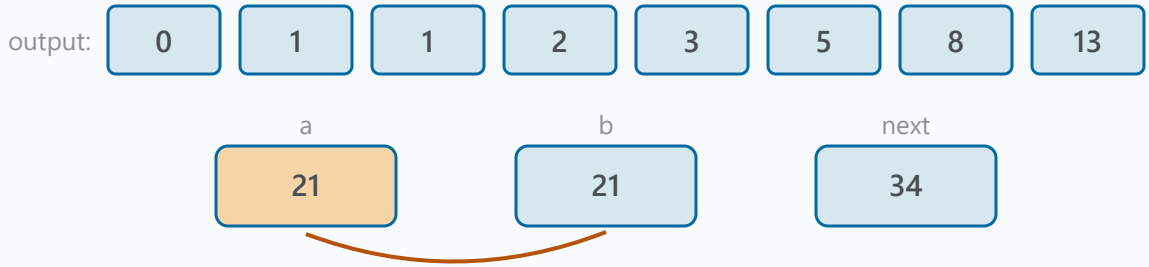
VARIABLES

`n = 8` `round = 7` `a = 13` `b = 21` `next = 34`

OUTPUT

```
0 1 1 2 3 5 8 13
```

Fibonacci Step by Step (cont.: rounds 6–7)



a = b = 21: the window slides.

Step 7 / 9

```
1 read n
2 a = 0, b = 1
3 for round = 0 to n-1
4     print a
5     next = a + b
▶     a = b
7     b = next
```

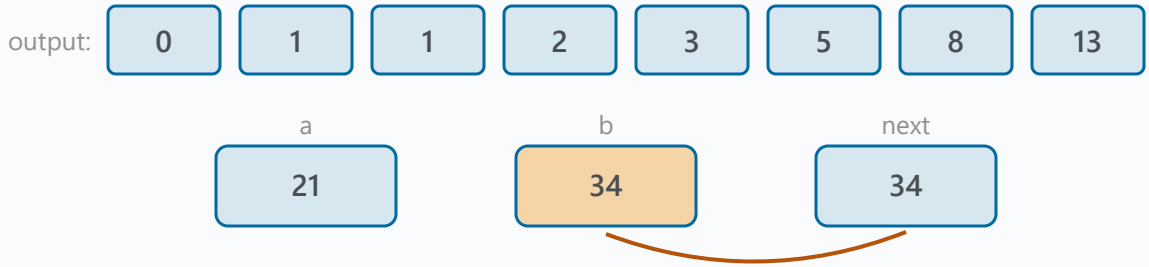
VARIABLES

n = 8 | round = 7 | a = 21 | b = 21 | next = 34

OUTPUT

```
0 1 1 2 3 5 8 13
```

Fibonacci Step by Step (cont.: rounds 6–7)



b = next = 34.

Step 8 / 9

```
1 read n
2 a = 0, b = 1
3 for round = 0 to n-1
4     print a
5     next = a + b
6     a = b
▶     b = next
```

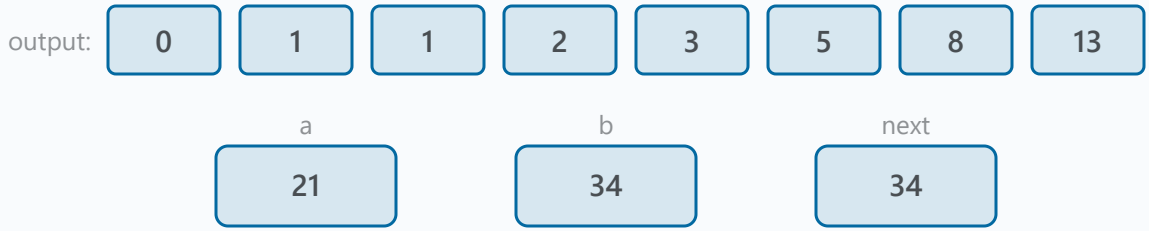
VARIABLES

`n = 8` `round = 7` `a = 21` `b = 34` `next = 34`

OUTPUT

```
0 1 1 2 3 5 8 13
```

Fibonacci Step by Step (cont.: rounds 6–7)



All 8 rounds done, so the loop ends. Output: **0 1 1 2 3 5 8 13**. The last term printed is **13**.

Step 9 / 9

```
1 read n
2 a = 0, b = 1
▶ for round = 0 to n-1
4     print a
5     next = a + b
6     a = b
7     b = next
```

VARIABLES

`n = 8` `a = 21` `b = 34` `next = 34`

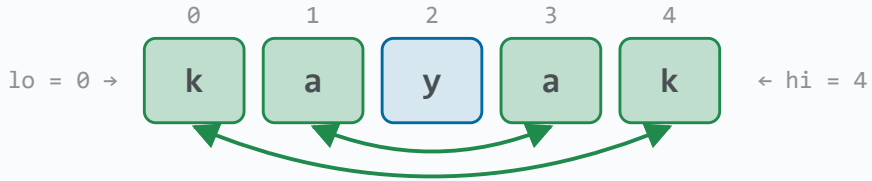
OUTPUT

```
0 1 1 2 3 5 8 13
```

Example: Palindrome

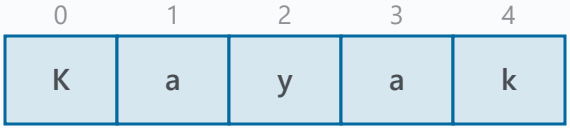
```
read text
clean text by the rule
lo = 0, hi = length - 1
while lo < hi
    if the characters differ, return false
    lo = lo + 1, hi = hi - 1
return true
```

- Comparison moves from both ends toward the middle.
- Decide the case and space rules up front (e.g. "Kayak" → "kayak").
- At most length / 2 comparisons.



Outside in: k = k, a = a; the middle y needs no comparison → ⌊5 / 2⌋ = 2 comparisons

Palindrome Step by Step ("Kayak")



Read the text: **"Kayak"**. The first letter is uppercase.

Step 1 / 11

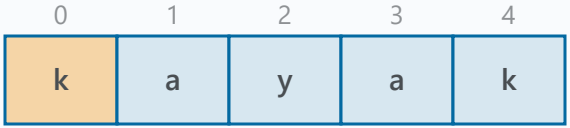
```
▶ read text
2 clean text by the rule
3 lo = 0, hi = length - 1
4 while lo < hi
5     if the characters differ, return false
6     lo = lo + 1, hi = hi - 1
7 return true
```

VARIABLES

`text = "Kayak"`

OUTPUT

Palindrome Step by Step ("Kayak")



Rule: lowercase it and drop spaces → **"kayak"**. Otherwise K ≠ k would be a mismatch.

Step 2 / 11

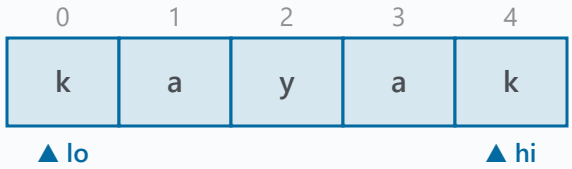
```
1 read text
▶ clean text by the rule
3 lo = 0, hi = length - 1
4 while lo < hi
5     if the characters differ, return false
6     lo = lo + 1, hi = hi - 1
7 return true
```

VARIABLES

`text = "kayak"`

OUTPUT

Palindrome Step by Step ("Kayak")



The two ends: **lo = 0**, **hi = 4**.

Step 3 / 11

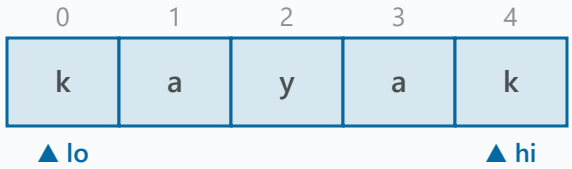
```
1 read text
2 clean text by the rule
▶ lo = 0, hi = length - 1
4 while lo < hi
5     if the characters differ, return false
6     lo = lo + 1, hi = hi - 1
7 return true
```

VARIABLES

`text = "kayak"` `lo = 0` `hi = 4`

OUTPUT

Palindrome Step by Step ("Kayak")



Condition: **0 < 4 → true**.

Step 4 / 11

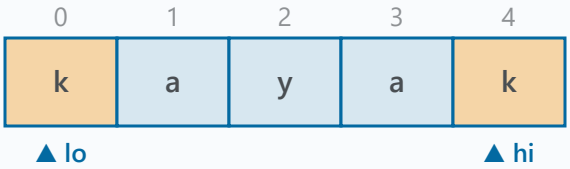
```
1 read text
2 clean text by the rule
3 lo = 0, hi = length - 1
▶ while lo < hi
5     if the characters differ, return false
6     lo = lo + 1, hi = hi - 1
7 return true
```

VARIABLES

`text = "kayak"` `lo = 0` `hi = 4`

OUTPUT

Palindrome Step by Step ("Kayak")



text[0] = 'k', text[4] = 'k' → **equal**, continue.

Step 5 / 11

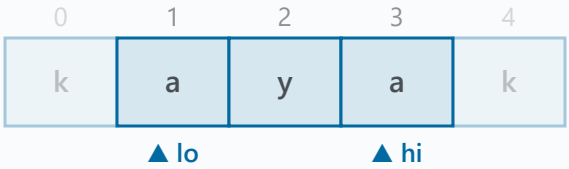
```
1 read text
2 clean text by the rule
3 lo = 0, hi = length - 1
4 while lo < hi
▶     if the characters differ, return false
6     lo = lo + 1, hi = hi - 1
7 return true
```

VARIABLES

`text = "kayak"` `lo = 0` `hi = 4`

OUTPUT

Palindrome Step by Step ("Kayak")



One step inward: **lo = 1**, **hi = 3**.

Step 6 / 11

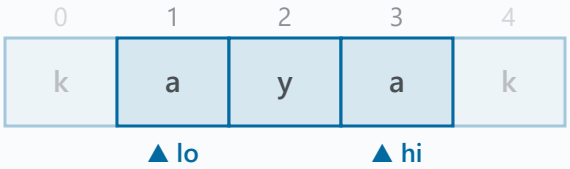
```
1 read text
2 clean text by the rule
3 lo = 0, hi = length - 1
4 while lo < hi
5     if the characters differ, return false
      lo = lo + 1, hi = hi - 1
7 return true
```

VARIABLES

`text = "kayak"` `lo = 1` `hi = 3`

OUTPUT

Palindrome Step by Step ("Kayak")



Condition: **1 < 3 → true**.

Step 7 / 11

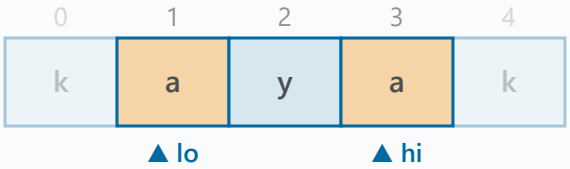
```
1 read text
2 clean text by the rule
3 lo = 0, hi = length - 1
▶ while lo < hi
5     if the characters differ, return false
6     lo = lo + 1, hi = hi - 1
7 return true
```

VARIABLES

`text = "kayak"` `lo = 1` `hi = 3`

OUTPUT

Palindrome Step by Step ("Kayak")



text[1] = 'a', text[3] = 'a' → **equal**, continue.

Step 8 / 11

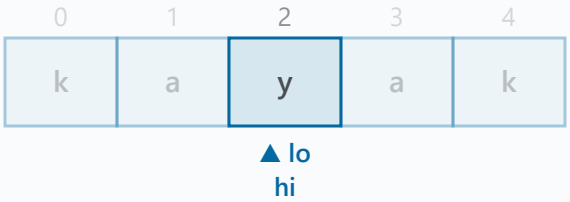
```
1 read text
2 clean text by the rule
3 lo = 0, hi = length - 1
4 while lo < hi
▶     if the characters differ, return false
6     lo = lo + 1, hi = hi - 1
7 return true
```

VARIABLES

`text = "kayak"` `lo = 1` `hi = 3`

OUTPUT

Palindrome Step by Step ("Kayak")



One step inward: **lo = 2**, **hi = 2**.

Step 9 / 11

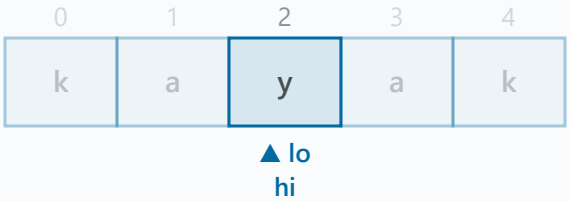
```
1 read text
2 clean text by the rule
3 lo = 0, hi = length - 1
4 while lo < hi
5     if the characters differ, return false
▶     lo = lo + 1, hi = hi - 1
7 return true
```

VARIABLES

`text = "kayak"` `lo = 2` `hi = 2`

OUTPUT

Palindrome Step by Step ("Kayak")



Condition: **2 < 2** → **false**: we reached the middle, so the loop ends.

Step 10 / 11

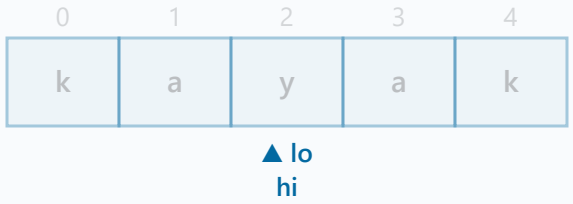
```
1 read text
2 clean text by the rule
3 lo = 0, hi = length - 1
▶ while lo < hi
5     if the characters differ, return false
6     lo = lo + 1, hi = hi - 1
7 return true
```

VARIABLES

`text = "kayak"` `lo = 2` `hi = 2`

OUTPUT

Palindrome Step by Step ("Kayak")



No mismatched pair → **true**: "Kayak" is a palindrome. Only 2 comparisons for 5 letters.

Step 11 / 11

```
1 read text
2 clean text by the rule
3 lo = 0, hi = length - 1
4 while lo < hi
5     if the characters differ, return false
6     lo = lo + 1, hi = hi - 1
▶ return true
```

VARIABLES

`text = "kayak"` `lo = 2` `hi = 2`

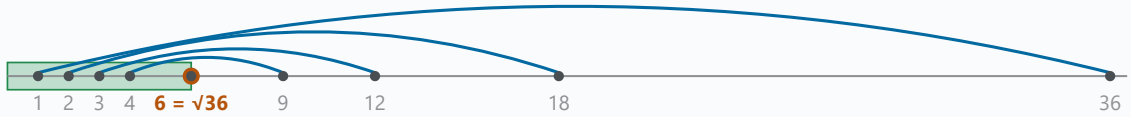
OUTPUT

```
true
```

Example: Prime Number

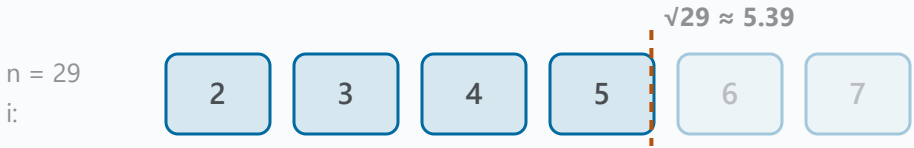
```
read n
if n < 2 then print "not prime" and stop
i = 2
while i × i ≤ n
    if n % i == 0 then print "not prime" and stop
    i = i + 1
print "prime"
```

- The search stops at the first divisor.
- Square-root bound: if n = a × b, then a and b cannot **both** exceed √n; any small divisor lies in the range i × i ≤ n.
- % gives the remainder of a division.



36 = 1×36 = 2×18 = 3×12 = 4×9 = 6×6: the smaller of each pair ≤ √36 → trying i = 2…6 is enough

Prime Number Step by Step (n = 29)



Read **n = 29**.

Step 1 / 17

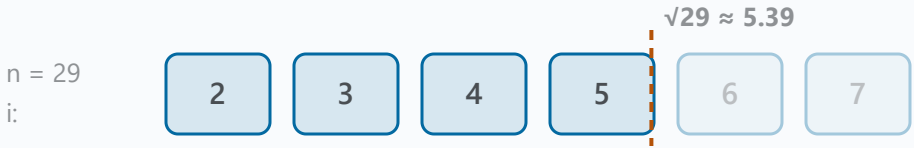
```
▶ read n
2 if n < 2 then print "not prime" and stop
3 i = 2
4 while i × i ≤ n
5     if n % i == 0 then print "not prime" and
  stop
6     i = i + 1
7 print "prime"
```

VARIABLES

n = **29**

OUTPUT

Prime Number Step by Step (n = 29)



29 < 2 → **false**; not 0 or 1, continue.

Step 2 / 17

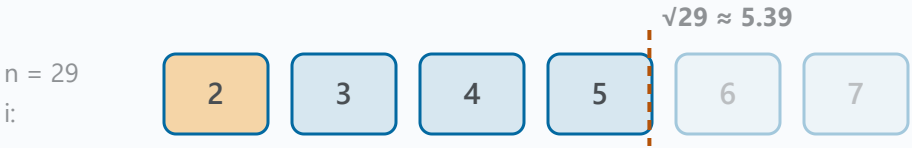
```
1 read n
▶ if n < 2 then print "not prime" and stop
3 i = 2
4 while i × i ≤ n
5     if n % i == 0 then print "not prime" and
  stop
6     i = i + 1
7 print "prime"
```

VARIABLES

n = 29

OUTPUT

Prime Number Step by Step (n = 29)



i = 2: the first candidate divisor.

Step 3 / 17

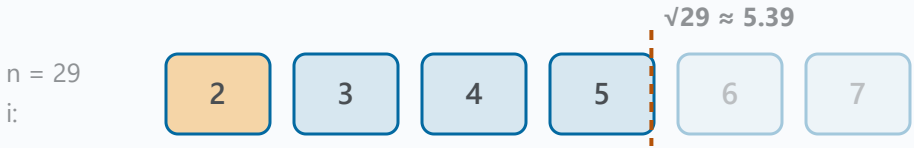
```
1 read n
2 if n < 2 then print "not prime" and stop
▶ i = 2
4 while i × i ≤ n
5     if n % i == 0 then print "not prime" and
  stop
6     i = i + 1
7 print "prime"
```

VARIABLES

n = 29 i = 2

OUTPUT

Prime Number Step by Step (n = 29)



Condition: **2×2 = 4 ≤ 29 → true**.

Step 4 / 17

```
1 read n
2 if n < 2 then print "not prime" and stop
3 i = 2
▶ while i × i ≤ n
5     if n % i == 0 then print "not prime" and stop
6     i = i + 1
7 print "prime"
```

VARIABLES

`n = 29` `i = 2` `i×i = 4`

OUTPUT

Prime Number Step by Step (n = 29)



29 % 2 = 1 → not zero, so i is not a divisor.

Step 5 / 17

```
1 read n
2 if n < 2 then print "not prime" and stop
3 i = 2
4 while i × i ≤ n
▶     if n % i == 0 then print "not prime" and
  stop
6     i = i + 1
7 print "prime"
```

VARIABLES

`n = 29` `i = 2` `n%i = 1`

OUTPUT

Prime Number Step by Step (n = 29)



i = 3.

Step 6 / 17

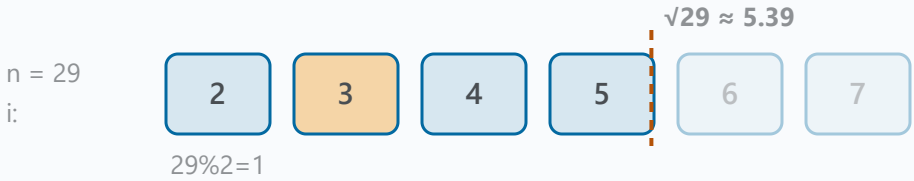
```
1 read n
2 if n < 2 then print "not prime" and stop
3 i = 2
4 while i × i ≤ n
5     if n % i == 0 then print "not prime" and stop
      i = i + 1
7 print "prime"
```

VARIABLES

n = 29 i = 3

OUTPUT

Prime Number Step by Step (n = 29)



Condition: **3×3 = 9 ≤ 29 → true**.

Step 7 / 17

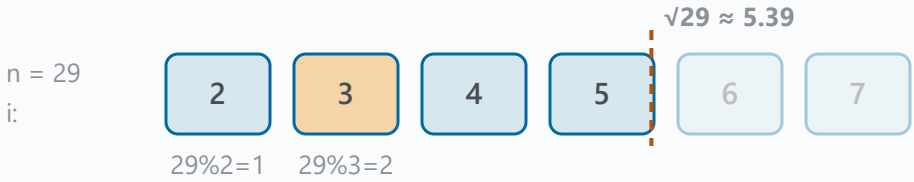
```
1 read n
2 if n < 2 then print "not prime" and stop
3 i = 2
▶ while i × i ≤ n
5     if n % i == 0 then print "not prime" and stop
6     i = i + 1
7 print "prime"
```

VARIABLES

`n = 29` `i = 3` `i×i = 9`

OUTPUT

Prime Number Step by Step (n = 29)



29 % 3 = 2 → not zero, so i is not a divisor.

Step 8 / 17

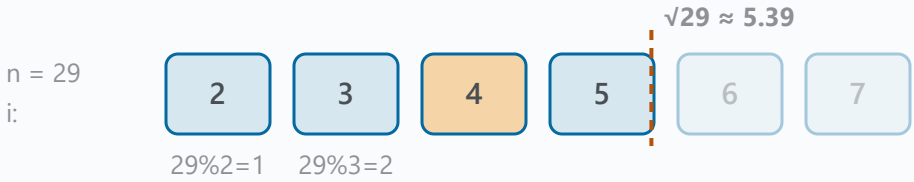
```
1 read n
2 if n < 2 then print "not prime" and stop
3 i = 2
4 while i × i ≤ n
▶     if n % i == 0 then print "not prime" and
  stop
6     i = i + 1
7 print "prime"
```

VARIABLES

`n = 29` `i = 3` `n%i = 2`

OUTPUT

Prime Number Step by Step (n = 29)



i = 4.

Step 9 / 17

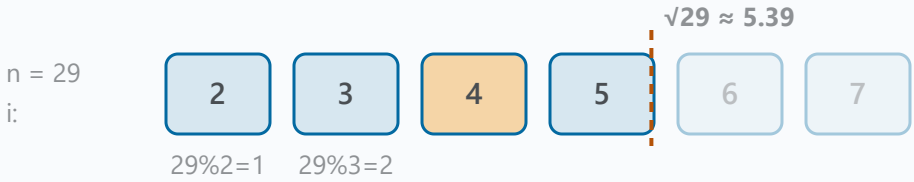
```
1 read n
2 if n < 2 then print "not prime" and stop
3 i = 2
4 while i × i ≤ n
5     if n % i == 0 then print "not prime" and stop
▶     i = i + 1
7 print "prime"
```

VARIABLES

n = 29 i = 4

OUTPUT

Prime Number Step by Step (n = 29)



Condition: **4×4 = 16 ≤ 29 → true**.

Step 10 / 17

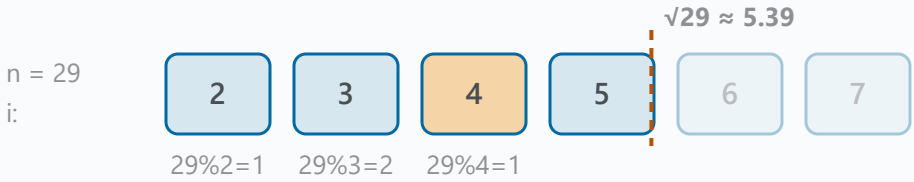
```
1 read n
2 if n < 2 then print "not prime" and stop
3 i = 2
▶ while i × i ≤ n
5     if n % i == 0 then print "not prime" and stop
6     i = i + 1
7 print "prime"
```

VARIABLES

`n = 29` `i = 4` `i×i = 16`

OUTPUT

Prime Number Step by Step (n = 29)



29 % 4 = 1 → not zero, so i is not a divisor.

Step 11 / 17

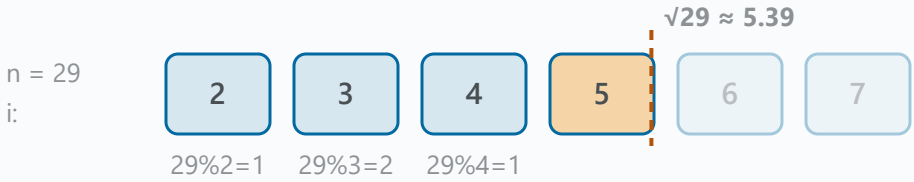
```
1 read n
2 if n < 2 then print "not prime" and stop
3 i = 2
4 while i × i ≤ n
▶     if n % i == 0 then print "not prime" and
  stop
6     i = i + 1
7 print "prime"
```

VARIABLES

`n = 29` `i = 4` `n%i = 1`

OUTPUT

Prime Number Step by Step (n = 29)



i = 5.

Step 12 / 17

```
1 read n
2 if n < 2 then print "not prime" and stop
3 i = 2
4 while i × i ≤ n
5     if n % i == 0 then print "not prime" and stop
      i = i + 1
7 print "prime"
```

VARIABLES

n = 29 i = 5

OUTPUT

Prime Number Step by Step (n = 29)



Condition: **5×5 = 25 ≤ 29 → true**.

Step 13 / 17

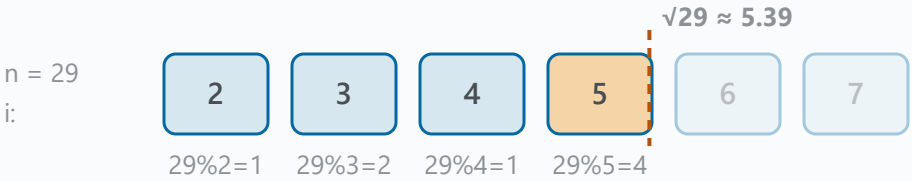
```
1 read n
2 if n < 2 then print "not prime" and stop
3 i = 2
▶ while i × i ≤ n
5     if n % i == 0 then print "not prime" and
  stop
6     i = i + 1
7 print "prime"
```

VARIABLES

`n = 29` `i = 5` `i×i = 25`

OUTPUT

Prime Number Step by Step (n = 29)



29 % 5 = 4 → not zero, so i is not a divisor.

Step 14 / 17

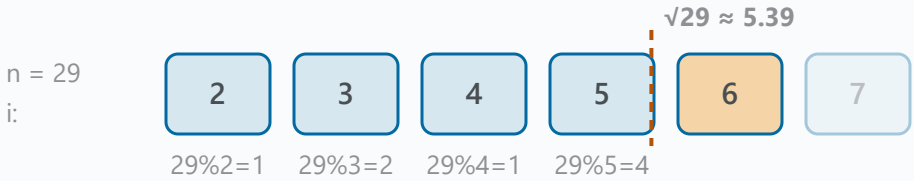
```
1 read n
2 if n < 2 then print "not prime" and stop
3 i = 2
4 while i × i ≤ n
▶     if n % i == 0 then print "not prime" and
  stop
6     i = i + 1
7 print "prime"
```

VARIABLES

n = 29 | i = 5 | n%i = 4

OUTPUT

Prime Number Step by Step (n = 29)



i = 6.

Step 15 / 17

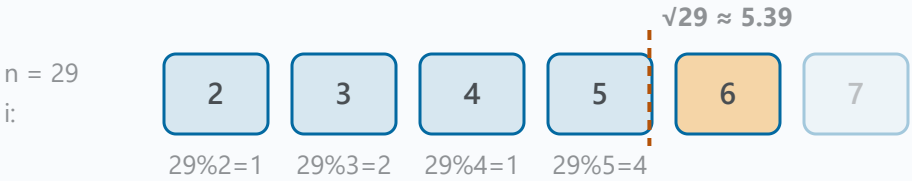
```
1 read n
2 if n < 2 then print "not prime" and stop
3 i = 2
4 while i × i ≤ n
5     if n % i == 0 then print "not prime" and stop
      i = i + 1
7 print "prime"
```

VARIABLES

n = 29 i = 6

OUTPUT

Prime Number Step by Step (n = 29)



Condition: **6×6 = 36 ≤ 29 → false**: i has passed √29, so there is no need to go on.

Step 16 / 17

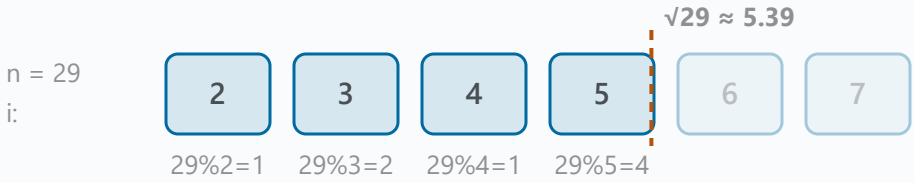
```
1 read n
2 if n < 2 then print "not prime" and stop
3 i = 2
▶ while i × i ≤ n
5     if n % i == 0 then print "not prime" and
  stop
6     i = i + 1
7 print "prime"
```

VARIABLES

`n = 29` `i = 6` `i×i = 36`

OUTPUT

Prime Number Step by Step (n = 29)



tried 2, 3, 4, 5; 6 and beyond are unnecessary

No divisor found → **prime**. Only 4 divisors tried (instead of 2…28). For n = 21, i = 3 gives 21 % 3 = 0 → not prime.

Step 17 / 17

```
1 read n
2 if n < 2 then print "not prime" and stop
3 i = 2
4 while i × i ≤ n
5     if n % i == 0 then print "not prime" and stop
6     i = i + 1
▶ print "prime"
```

VARIABLES

`n = 29` `i = 6`

OUTPUT

```
prime
```

VERIFY

Tracing an Algorithm

- Record each variable's value at every step in a **trace table**.
- Try normal, boundary and invalid inputs separately.
- When expected and actual results differ, find the first step where they diverge.

Trace table for factorial, n = 4 (`i`: value at the start of the round, `result`: value at the end):

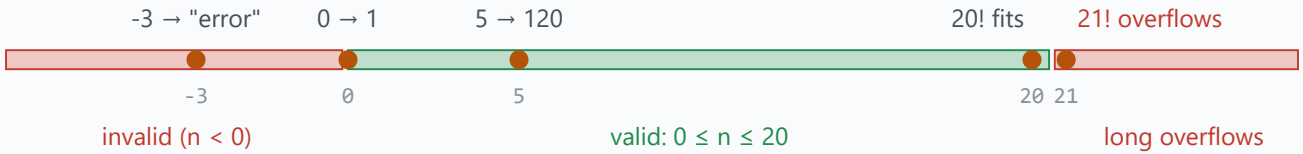
Step	i	result	i ≤ n
start	2	1	—
round 1	2	2	true
round 2	3	6	true
round 3	4	24	true
exit	5	24	5 ≤ 4 false

One passing test does not prove an algorithm correct for every input.

VERIFY

Test Cases: Factorial

Type	Goal	Input	Expected
Normal	Check typical use	n = 5	120
Boundary	Try the lower bound	n = 0	1 (the loop never runs)
Invalid	Check error handling	n = -3	"error"
Large	Detect overflow	n = 21	21! = 51,090,942,171,709,440,000 → does not fit in a 64-bit integer



A 64-bit integer (Java `long`) holds at most 9,223,372,036,854,775,807; 20! = 2,432,902,008,176,640,000 fits, 21! does not. Hence the constraint 0 ≤ n ≤ 20.

ANALYSIS

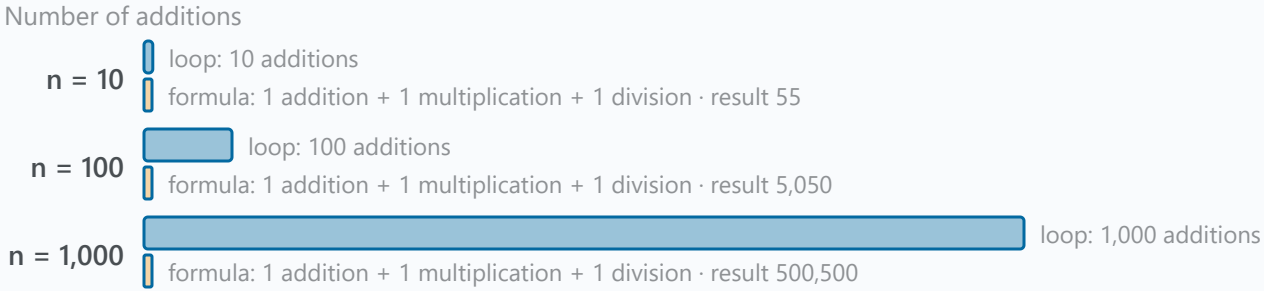
Same Problem, Two Algorithms: 1 + 2 + … + n

```
sum = 0
i = 1
while i <= n
    sum = sum + i
    i = i + 1
```

A: a loop, n rounds

```
sum = n * (n + 1) / 2
```

B: Gauss's formula, one step



Both are **correct**, but as n grows A does more work while B's stays constant: **efficiency** matters as much as correctness.

ANALYSIS

Correctness and Efficiency

- Correctness means the algorithm produces the expected result.
- Time complexity describes how the operation count grows with input size, written in big-O notation: O(n), O(log n), O(n^2).
- Space complexity measures the extra memory used.
- On small inputs, the fastest-looking method is not always the clearest one.

n	$\lfloor \log_2 n \rfloor + 1$ (binary search)	n (linear search)	n^2 (simple sort)
10	4	10	100
100	7	100	10,000
1,000	10	1,000	1,000,000
1,000,000	20	1,000,000	1,000,000,000,000

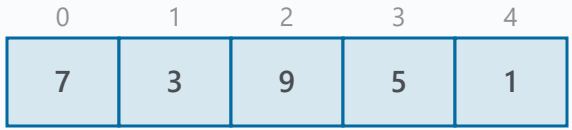
Linear Search

```java
int linearSearch(int[] arr, int target) {
    for (int i = 0; i < arr.length; i++) {
        if (arr[i] == target) return i;
    }
    return -1;
}
```

- Compares elements from first to last; the data need not be sorted.
- Stops at the first match; returns -1 if there is none.
- Worst case examines all n elements: O(n).

The code is in Java.

Linear Search Step by Step (target = 5)



Call `linearSearch(arr, 5)`. The array is not sorted, so we check one element at a time from the start.

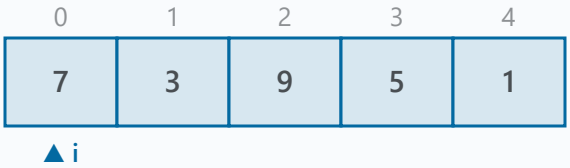
Step 1 / 9

```
▶ int linearSearch(int[] arr, int target) {
2     for (int i = 0; i < arr.length; i++) {
3         if (arr[i] == target) return i;
4     }
5     return -1;
6 }
```

VARIABLES

`arr = [7, 3, 9, 5, 1]` `target = 5`

Linear Search Step by Step (target = 5)



i = 0: 0 < 5 → true.

Step 2 / 9

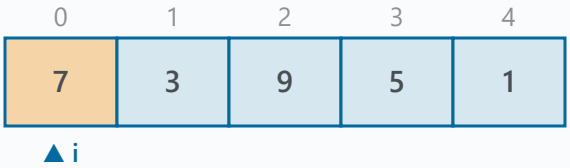
```
1 int linearSearch(int[] arr, int target) {
▶     for (int i = 0; i < arr.length; i++) {
3         if (arr[i] == target) return i;
4     }
5     return -1;
6 }
```

VARIABLES

arr = [7, 3, 9, 5, 1] target = 5

i = 0

Linear Search Step by Step (target = 5)



arr[0] = 7 == 5 → **false**.

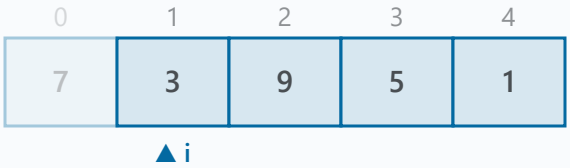
Step 3 / 9

```
1 int linearSearch(int[] arr, int target) {
2     for (int i = 0; i < arr.length; i++) {
▶         if (arr[i] == target) return i;
4     }
5     return -1;
6 }
```

VARIABLES

`arr = [7, 3, 9, 5, 1]` `target = 5` `i = 0`

Linear Search Step by Step (target = 5)



i = 1: 1 < 5 → true.

Step 4 / 9

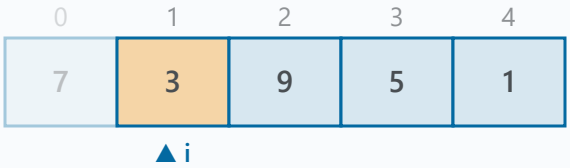
```
1 int linearSearch(int[] arr, int target) {
    for (int i = 0; i < arr.length; i++) {
3       if (arr[i] == target) return i;
4   }
5   return -1;
6 }
```

VARIABLES

arr = [7, 3, 9, 5, 1] target = 5

i = 1

Linear Search Step by Step (target = 5)



arr[1] = 3 == 5 → **false**.

Step 5 / 9

```
1 int linearSearch(int[] arr, int target) {
2     for (int i = 0; i < arr.length; i++) {
▶         if (arr[i] == target) return i;
4     }
5     return -1;
6 }
```

VARIABLES

`arr = [7, 3, 9, 5, 1]` `target = 5` `i = 1`

Linear Search Step by Step (target = 5)



i = 2: 2 < 5 → true.

Step 6 / 9

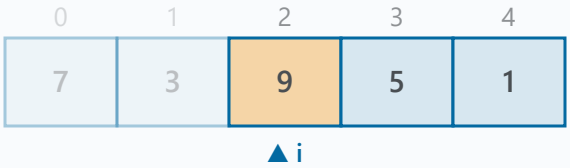
```
1 int linearSearch(int[] arr, int target) {
▶       for (int i = 0; i < arr.length; i++) {
3           if (arr[i] == target) return i;
4       }
5       return -1;
6   }
```

VARIABLES

arr = [7, 3, 9, 5, 1] target = 5

i = 2

Linear Search Step by Step (target = 5)



arr[2] = 9 == 5 → **false**.

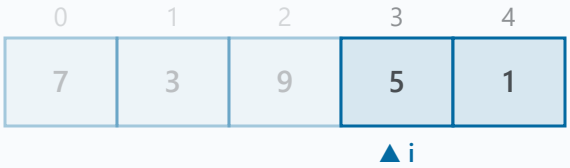
Step 7 / 9

```
1 int linearSearch(int[] arr, int target) {
2     for (int i = 0; i < arr.length; i++) {
          if (arr[i] == target) return i;
4     }
5     return -1;
6 }
```

VARIABLES

`arr = [7, 3, 9, 5, 1]` `target = 5` `i = 2`

Linear Search Step by Step (target = 5)



i = 3: 3 < 5 → true.

Step 8 / 9

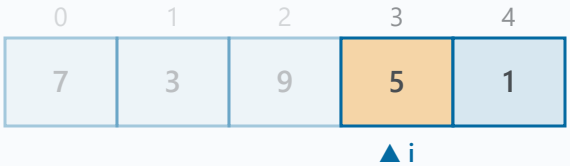
```
1 int linearSearch(int[] arr, int target) {
▶       for (int i = 0; i < arr.length; i++) {
3           if (arr[i] == target) return i;
4       }
5       return -1;
6   }
```

VARIABLES

arr = [7, 3, 9, 5, 1] target = 5

i = 3

Linear Search Step by Step (target = 5)



arr[3] = 5 == 5 → **true** → **return 3**. That took 4 comparisons; for target 4 it would return -1 after 5.

Step 9 / 9

```
1 int linearSearch(int[] arr, int target) {
2     for (int i = 0; i < arr.length; i++) {
▶         if (arr[i] == target) return i;
4     }
5     return -1;
6 }
```

VARIABLES

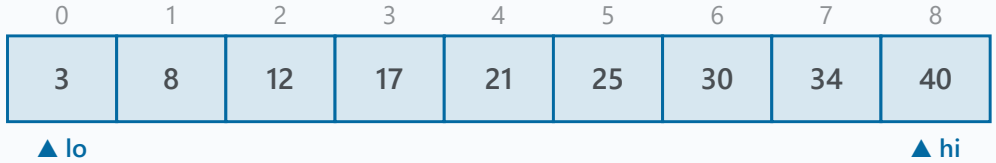
`arr = [7, 3, 9, 5, 1]` `target = 5` `i = 3`

Binary Search

```java
int binarySearch(int[] arr, int target) {
    int lo = 0, hi = arr.length - 1;
    while (lo <= hi) {
        int mid = lo + (hi - lo) / 2;
        if (arr[mid] == target) return mid;
        if (arr[mid] < target) lo = mid + 1;
        else hi = mid - 1;
    }
    return -1;
}
```

- Works only on **sorted** data; each step halves the range.
- Worst case $\lfloor \log_2 n \rfloor + 1$ rounds: O(log n).
- `lo + (hi - lo) / 2` gives the same result as `(lo + hi) / 2` but does not overflow for very large indices.
- Sorting first just to search once can raise the total cost.

Binary Search Step by Step (target = 34)



The search range is the whole array: **lo = 0**, **hi = 8**.

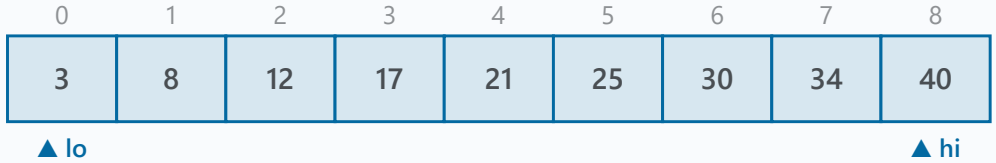
Step 1 / 12

```
1  int binarySearch(int[] arr, int target) {
▶      int lo = 0, hi = arr.length - 1;
3      while (lo <= hi) {
4          int mid = lo + (hi - lo) / 2;
5          if (arr[mid] == target) return mid;
6          if (arr[mid] < target) lo = mid + 1;
7          else hi = mid - 1;
8      }
9      return -1;
10 }
```

VARIABLES

`target = 34` `lo = 0` `hi = 8`

Binary Search Step by Step (target = 34)



0 ≤ 8 → true: the range is not empty, round 1.

Step 2 / 12

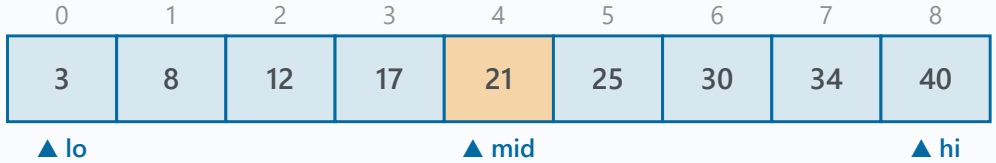
```
1  int binarySearch(int[] arr, int target) {
2      int lo = 0, hi = arr.length - 1;
▶      while (lo <= hi) {
4          int mid = lo + (hi - lo) / 2;
5          if (arr[mid] == target) return mid;
6          if (arr[mid] < target) lo = mid + 1;
7          else hi = mid - 1;
8      }
9      return -1;
10 }
```

VARIABLES

`target = 34` `lo = 0` `hi = 8`

`round = 1`

Binary Search Step by Step (target = 34)



mid = 0 + (8 − 0) / 2 = 4, arr[4] = 21.

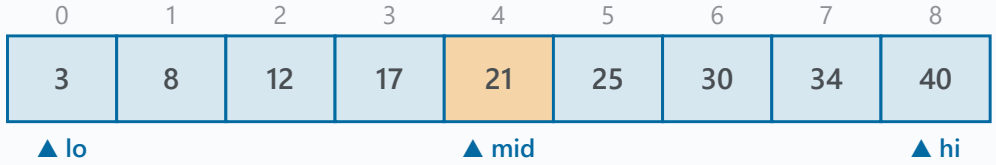
Step 3 / 12

```
1  int binarySearch(int[] arr, int target) {
2      int lo = 0, hi = arr.length - 1;
3      while (lo <= hi) {
▶          int mid = lo + (hi - lo) / 2;
5          if (arr[mid] == target) return mid;
6          if (arr[mid] < target) lo = mid + 1;
7          else hi = mid - 1;
8      }
9      return -1;
10 }
```

VARIABLES

`target = 34` `lo = 0` `hi = 8` `mid = 4` `round = 1`

Binary Search Step by Step (target = 34)



21 == 34 → false.

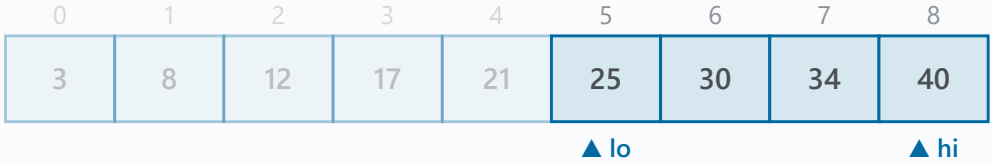
Step 4 / 12

```
1  int binarySearch(int[] arr, int target) {
2      int lo = 0, hi = arr.length - 1;
3      while (lo <= hi) {
4          int mid = lo + (hi - lo) / 2;
▶          if (arr[mid] == target) return mid;
6          if (arr[mid] < target) lo = mid + 1;
7          else hi = mid - 1;
8      }
9      return -1;
10 }
```

VARIABLES

`target = 34` `lo = 0` `hi = 8` `mid = 4` `round = 1`

Binary Search Step by Step (target = 34)



21 < 34 → the target is in the right half: **lo = 5**. The left half is gone.

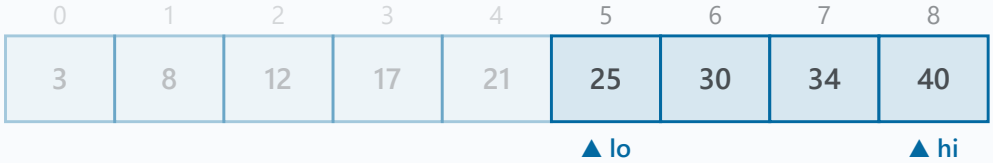
Step 5 / 12

```
1  int binarySearch(int[] arr, int target) {
2      int lo = 0, hi = arr.length - 1;
3      while (lo <= hi) {
4          int mid = lo + (hi - lo) / 2;
5          if (arr[mid] == target) return mid;
▶          if (arr[mid] < target) lo = mid + 1;
7          else hi = mid - 1;
8      }
9      return -1;
10 }
```

VARIABLES

`target = 34` `lo = 5` `hi = 8` `mid = 4` `round = 1`

Binary Search Step by Step (target = 34)



5 ≤ 8 → true: the range is not empty, round 2.

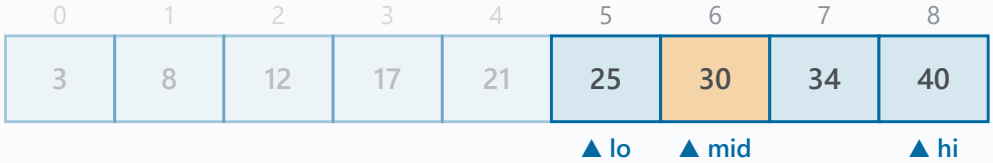
Step 6 / 12

```
1  int binarySearch(int[] arr, int target) {
2      int lo = 0, hi = arr.length - 1;
▶      while (lo <= hi) {
4          int mid = lo + (hi - lo) / 2;
5          if (arr[mid] == target) return mid;
6          if (arr[mid] < target) lo = mid + 1;
7          else hi = mid - 1;
8      }
9      return -1;
10 }
```

VARIABLES

target = 34 | lo = 5 | hi = 8 | round = 2

Binary Search Step by Step (target = 34)



mid = 5 + (8 − 5) / 2 = 6, arr[6] = 30.

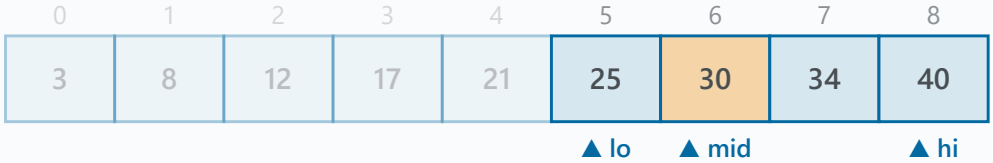
Step 7 / 12

```
1  int binarySearch(int[] arr, int target) {
2      int lo = 0, hi = arr.length - 1;
3      while (lo <= hi) {
▶          int mid = lo + (hi - lo) / 2;
5          if (arr[mid] == target) return mid;
6          if (arr[mid] < target) lo = mid + 1;
7          else hi = mid - 1;
8      }
9      return -1;
10 }
```

VARIABLES

target = 34 | lo = 5 | hi = 8 | mid = 6 | round = 2

Binary Search Step by Step (target = 34)



30 == 34 → false.

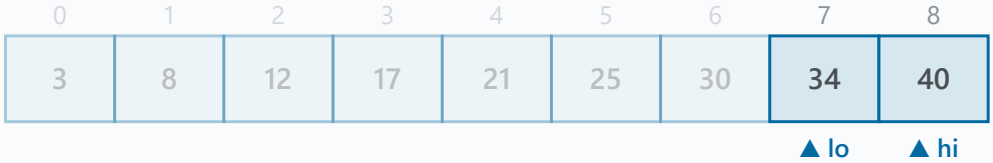
Step 8 / 12

```
1  int binarySearch(int[] arr, int target) {
2      int lo = 0, hi = arr.length - 1;
3      while (lo <= hi) {
4          int mid = lo + (hi - lo) / 2;
▶          if (arr[mid] == target) return mid;
6          if (arr[mid] < target) lo = mid + 1;
7          else hi = mid - 1;
8      }
9      return -1;
10 }
```

VARIABLES

target = 34 | lo = 5 | hi = 8 | mid = 6 | round = 2

Binary Search Step by Step (target = 34)



30 < 34 → the target is in the right half: **lo = 7**. The left half is gone.

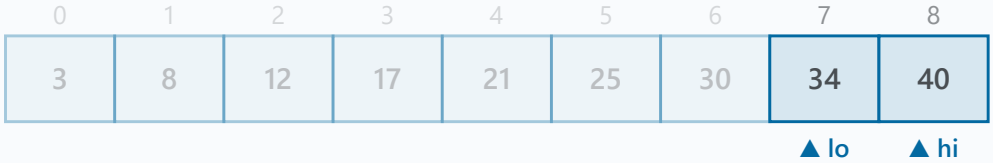
Step 9 / 12

```
1  int binarySearch(int[] arr, int target) {
2      int lo = 0, hi = arr.length - 1;
3      while (lo <= hi) {
4          int mid = lo + (hi - lo) / 2;
5          if (arr[mid] == target) return mid;
▶          if (arr[mid] < target) lo = mid + 1;
7          else hi = mid - 1;
8      }
9      return -1;
10 }
```

VARIABLES

`target = 34` `lo = 7` `hi = 8` `mid = 6` `round = 2`

Binary Search Step by Step (target = 34)



7 ≤ 8 → true: the range is not empty, round 3.

Step 10 / 12

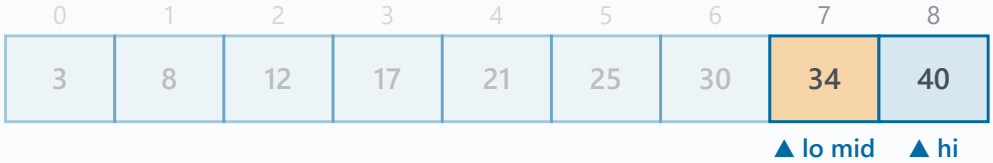
```
1  int binarySearch(int[] arr, int target) {
2      int lo = 0, hi = arr.length - 1;
▶      while (lo <= hi) {
4          int mid = lo + (hi - lo) / 2;
5          if (arr[mid] == target) return mid;
6          if (arr[mid] < target) lo = mid + 1;
7          else hi = mid - 1;
8      }
9      return -1;
10 }
```

VARIABLES

`target = 34` `lo = 7` `hi = 8`

`round = 3`

Binary Search Step by Step (target = 34)



mid = 7 + (8 − 7) / 2 = 7, arr[7] = 34.

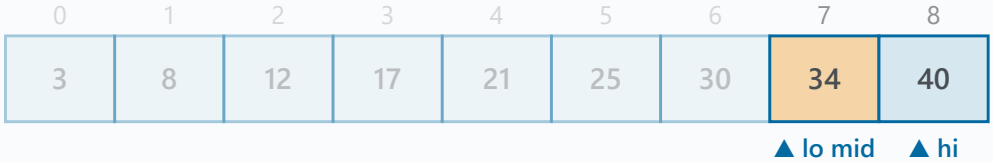
Step 11 / 12

```
1  int binarySearch(int[] arr, int target) {
2      int lo = 0, hi = arr.length - 1;
3      while (lo <= hi) {
▶          int mid = lo + (hi - lo) / 2;
5          if (arr[mid] == target) return mid;
6          if (arr[mid] < target) lo = mid + 1;
7          else hi = mid - 1;
8      }
9      return -1;
10 }
```

VARIABLES

target = 34 | lo = 7 | hi = 8 | mid = 7 | round = 3

Binary Search Step by Step (target = 34)



34 == 34 → return 7. Found in 3 rounds; linear search would have taken 8 comparisons.

Step 12 / 12

```
1  int binarySearch(int[] arr, int target) {
2      int lo = 0, hi = arr.length - 1;
3      while (lo <= hi) {
4          int mid = lo + (hi - lo) / 2;
▶          if (arr[mid] == target) return mid;
6          if (arr[mid] < target) lo = mid + 1;
7          else hi = mid - 1;
8      }
9      return -1;
10 }
```

VARIABLES

`target = 34` `lo = 7` `hi = 8` `mid = 7` `round = 3`

Binary Search: Target Not Found (target = 19)



The search range is the whole array: **lo = 0**, **hi = 7**. The target 19 is **not** in the array (it would sit between 17 and 21).

Step 1 / 15

```
1  int binarySearch(int[] arr, int target) {
▶      int lo = 0, hi = arr.length - 1;
3      while (lo <= hi) {
4          int mid = lo + (hi - lo) / 2;
5          if (arr[mid] == target) return mid;
6          if (arr[mid] < target) lo = mid + 1;
7          else hi = mid - 1;
8      }
9      return -1;
10 }
```

VARIABLES

`target = 19` `lo = 0` `hi = 7`

Binary Search: Target Not Found (target = 19)



0 ≤ 7 → true: the range is not empty, round 1.

Step 2 / 15

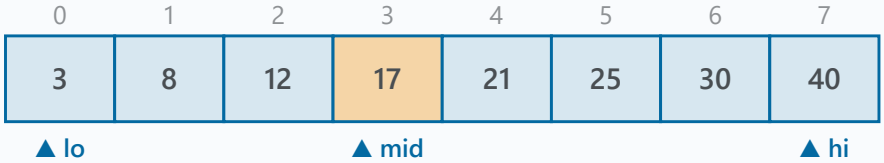
```
1  int binarySearch(int[] arr, int target) {
2      int lo = 0, hi = arr.length - 1;
▶      while (lo <= hi) {
4          int mid = lo + (hi - lo) / 2;
5          if (arr[mid] == target) return mid;
6          if (arr[mid] < target) lo = mid + 1;
7          else hi = mid - 1;
8      }
9      return -1;
10 }
```

VARIABLES

target = 19 | lo = 0 | hi = 7

round = 1

Binary Search: Target Not Found (target = 19)



mid = 0 + (7 − 0) / 2 = 3, arr[3] = 17.

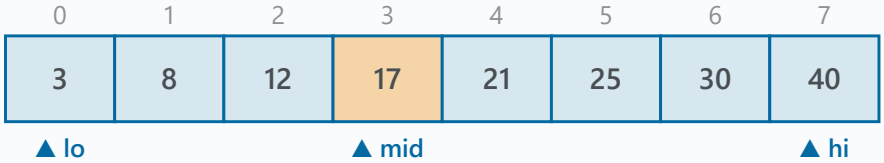
Step 3 / 15

```
1  int binarySearch(int[] arr, int target) {
2      int lo = 0, hi = arr.length - 1;
3      while (lo <= hi) {
▶          int mid = lo + (hi - lo) / 2;
5          if (arr[mid] == target) return mid;
6          if (arr[mid] < target) lo = mid + 1;
7          else hi = mid - 1;
8      }
9      return -1;
10 }
```

VARIABLES

target = 19 | lo = 0 | hi = 7 | mid = 3 | round = 1

Binary Search: Target Not Found (target = 19)



17 == 19 → false.

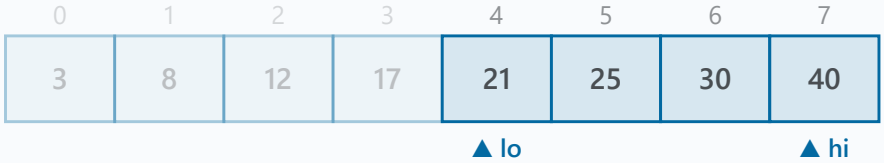
Step 4 / 15

```
1  int binarySearch(int[] arr, int target) {
2      int lo = 0, hi = arr.length - 1;
3      while (lo <= hi) {
4          int mid = lo + (hi - lo) / 2;
▶          if (arr[mid] == target) return mid;
6          if (arr[mid] < target) lo = mid + 1;
7          else hi = mid - 1;
8      }
9      return -1;
10 }
```

VARIABLES

target = 19 | lo = 0 | hi = 7 | mid = 3 | round = 1

Binary Search: Target Not Found (target = 19)



17 < 19 → true: the target can only be in the right half, **lo = 4**. The left half is gone.

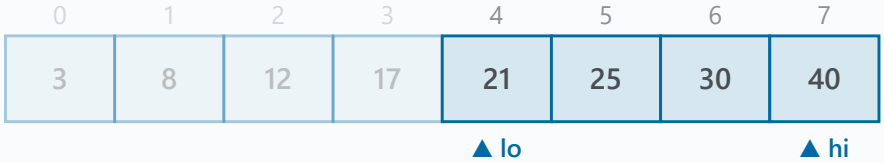
Step 5 / 15

```
1  int binarySearch(int[] arr, int target) {
2      int lo = 0, hi = arr.length - 1;
3      while (lo <= hi) {
4          int mid = lo + (hi - lo) / 2;
5          if (arr[mid] == target) return mid;
▶          if (arr[mid] < target) lo = mid + 1;
7          else hi = mid - 1;
8      }
9      return -1;
10 }
```

VARIABLES

target = 19 | lo = 4 | hi = 7 | mid = 3 | round = 1

Binary Search: Target Not Found (target = 19)



4 ≤ 7 → true: the range is not empty, round 2.

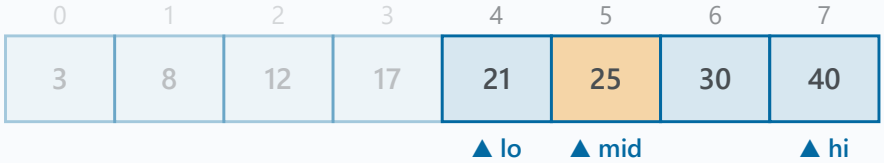
Step 6 / 15

```
1  int binarySearch(int[] arr, int target) {
2      int lo = 0, hi = arr.length - 1;
▶      while (lo <= hi) {
4          int mid = lo + (hi - lo) / 2;
5          if (arr[mid] == target) return mid;
6          if (arr[mid] < target) lo = mid + 1;
7          else hi = mid - 1;
8      }
9      return -1;
10 }
```

VARIABLES

`target = 19` `lo = 4` `hi = 7` `round = 2`

Binary Search: Target Not Found (target = 19)



mid = 4 + (7 − 4) / 2 = 5, arr[5] = 25.

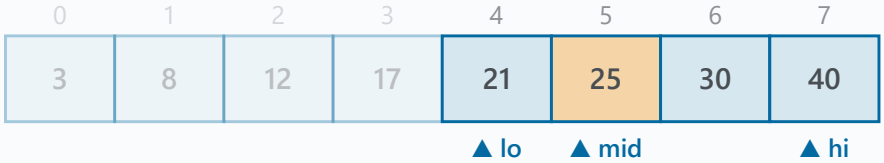
Step 7 / 15

```
1  int binarySearch(int[] arr, int target) {
2      int lo = 0, hi = arr.length - 1;
3      while (lo <= hi) {
▶          int mid = lo + (hi - lo) / 2;
5          if (arr[mid] == target) return mid;
6          if (arr[mid] < target) lo = mid + 1;
7          else hi = mid - 1;
8      }
9      return -1;
10 }
```

VARIABLES

target = 19 | lo = 4 | hi = 7 | mid = 5 | round = 2

Binary Search: Target Not Found (target = 19)



25 == 19 → false.

Step 8 / 15

```
1  int binarySearch(int[] arr, int target) {
2      int lo = 0, hi = arr.length - 1;
3      while (lo <= hi) {
4          int mid = lo + (hi - lo) / 2;
▶          if (arr[mid] == target) return mid;
6          if (arr[mid] < target) lo = mid + 1;
7          else hi = mid - 1;
8      }
9      return -1;
10 }
```

VARIABLES

target = 19 | lo = 4 | hi = 7 | mid = 5 | round = 2

Binary Search: Target Not Found (target = 19)



25 < 19 → false, **else**: the target can only be in the left half, **hi = 4**. The right half is gone.

Step 9 / 15

```
1  int binarySearch(int[] arr, int target) {
2      int lo = 0, hi = arr.length - 1;
3      while (lo <= hi) {
4          int mid = lo + (hi - lo) / 2;
5          if (arr[mid] == target) return mid;
6          if (arr[mid] < target) lo = mid + 1;
▶          else hi = mid - 1;
8      }
9      return -1;
10 }
```

VARIABLES

target = 19 | lo = 4 | hi = 4 | mid = 5 | round = 2

Binary Search: Target Not Found (target = 19)



4 ≤ 4 → true: the range is not empty, round 3.

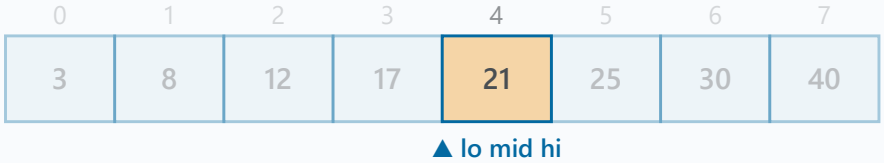
Step 10 / 15

```
1  int binarySearch(int[] arr, int target) {
2      int lo = 0, hi = arr.length - 1;
▶      while (lo <= hi) {
4          int mid = lo + (hi - lo) / 2;
5          if (arr[mid] == target) return mid;
6          if (arr[mid] < target) lo = mid + 1;
7          else hi = mid - 1;
8      }
9      return -1;
10 }
```

VARIABLES

`target = 19` `lo = 4` `hi = 4` `round = 3`

Binary Search: Target Not Found (target = 19)



mid = 4 + (4 − 4) / 2 = 4, arr[4] = 21.

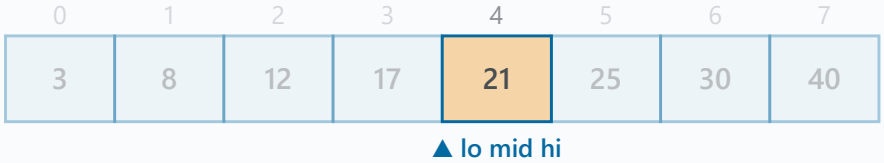
Step 11 / 15

```
1  int binarySearch(int[] arr, int target) {
2      int lo = 0, hi = arr.length - 1;
3      while (lo <= hi) {
▶          int mid = lo + (hi - lo) / 2;
5          if (arr[mid] == target) return mid;
6          if (arr[mid] < target) lo = mid + 1;
7          else hi = mid - 1;
8      }
9      return -1;
10 }
```

VARIABLES

target = 19 | lo = 4 | hi = 4 | mid = 4 | round = 3

Binary Search: Target Not Found (target = 19)



21 == 19 → false.

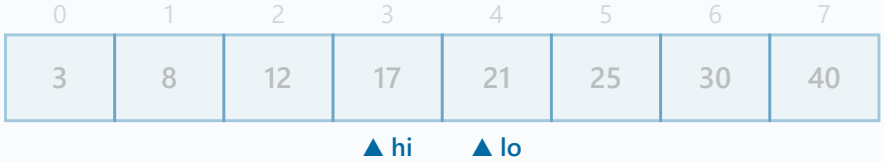
Step 12 / 15

```
1  int binarySearch(int[] arr, int target) {
2      int lo = 0, hi = arr.length - 1;
3      while (lo <= hi) {
4          int mid = lo + (hi - lo) / 2;
▶          if (arr[mid] == target) return mid;
6          if (arr[mid] < target) lo = mid + 1;
7          else hi = mid - 1;
8      }
9      return -1;
10 }
```

VARIABLES

`target = 19` `lo = 4` `hi = 4` `mid = 4` `round = 3`

Binary Search: Target Not Found (target = 19)



21 < 19 → false, **else**: the target can only be in the left half, **hi = 3**. The right half is gone.

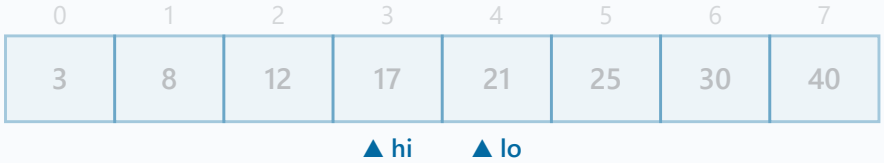
Step 13 / 15

```
1  int binarySearch(int[] arr, int target) {
2      int lo = 0, hi = arr.length - 1;
3      while (lo <= hi) {
4          int mid = lo + (hi - lo) / 2;
5          if (arr[mid] == target) return mid;
6          if (arr[mid] < target) lo = mid + 1;
▶          else hi = mid - 1;
8      }
9      return -1;
10 }
```

VARIABLES

`target = 19` `lo = 4` `hi = 3` `mid = 4` `round = 3`

Binary Search: Target Not Found (target = 19)



4 ≤ 3 → false: lo has passed hi; the range is **empty**, so the loop ends.

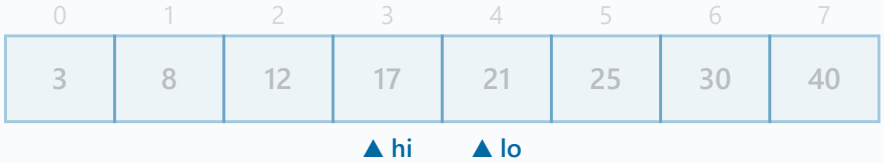
Step 14 / 15

```
1  int binarySearch(int[] arr, int target) {
2      int lo = 0, hi = arr.length - 1;
▶      while (lo <= hi) {
4          int mid = lo + (hi - lo) / 2;
5          if (arr[mid] == target) return mid;
6          if (arr[mid] < target) lo = mid + 1;
7          else hi = mid - 1;
8      }
9      return -1;
10 }
```

VARIABLES

`target = 19` `lo = 4` `hi = 3` `round = 3`

Binary Search: Target Not Found (target = 19)



return -1: target not found. The range emptied in 3 rounds, without checking all 8 elements.

Step 15 / 15

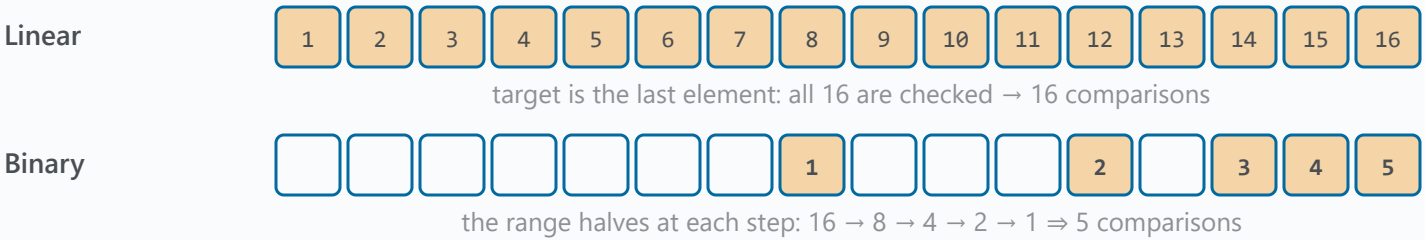
```
1  int binarySearch(int[] arr, int target) {
2      int lo = 0, hi = arr.length - 1;
3      while (lo <= hi) {
4          int mid = lo + (hi - lo) / 2;
5          if (arr[mid] == target) return mid;
6          if (arr[mid] < target) lo = mid + 1;
7          else hi = mid - 1;
8      }
▶      return -1;
10 }
```

VARIABLES

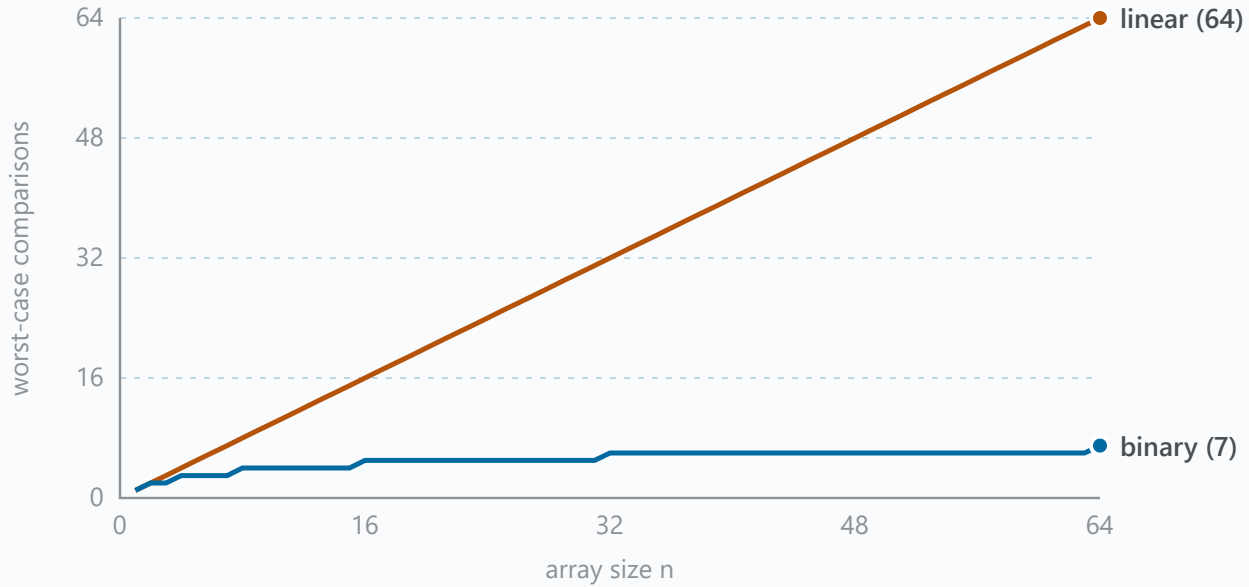
target = 19 | lo = 4 | hi = 3 | round = 3

Comparing the Searches

Property	Linear	Binary
Sorted data	Not needed	Required
Worst-case time	O(n)	O(log n)
Structure	Array or stream	Sorted, random-access structure
Use	Small or unsorted data	Many searches



Searching: How Many Comparisons in the Worst Case?



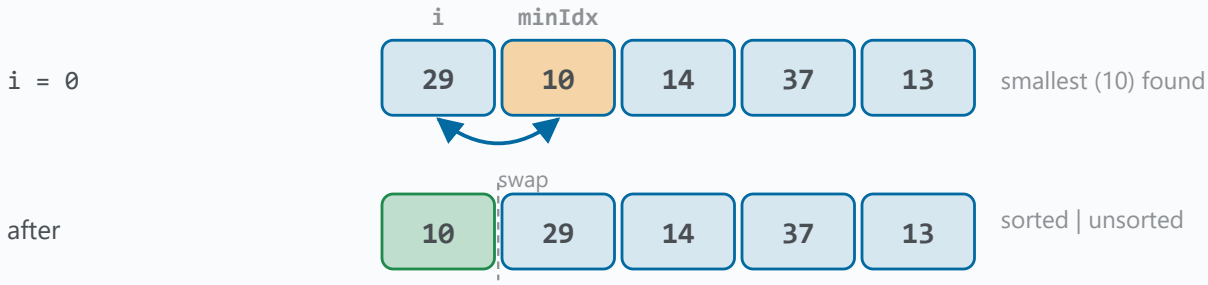
The values were counted by actually running both algorithms with a missing target. For n = 1,000,000, linear search needs **1,000,000** comparisons; binary search only **20** rounds.

SORTING

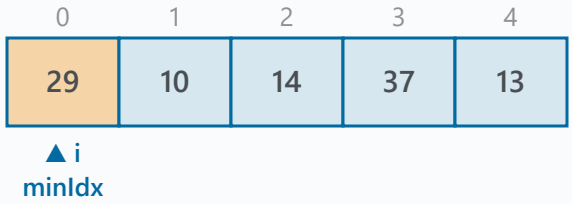
Selection Sort

- Find the smallest element in the unsorted part.
- Swap it into the first position of that part.
- Always n(n−1)/2 comparisons → $O(n^2)$; at most n − 1 swaps; O(1) extra memory.

```
for i = 0 to n-2
    minIdx = i
    for j = i+1 to n-1
        if arr[j] < arr[minIdx] then minIdx = j
    swap arr[i] and arr[minIdx]
```



Selection Sort Step by Step



i = 0: the unsorted part is 0…4. Candidate minimum: arr[0] = 29.

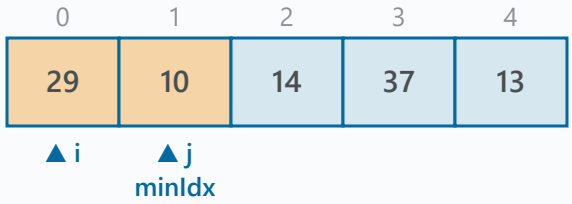
Step 1 / 18

```
▶ for i = 0 to n-2
▶     minIdx = i
3     for j = i+1 to n-1
4         if arr[j] < arr[minIdx] then minIdx =
  j
5     swap arr[i] and arr[minIdx]
```

VARIABLES

`arr = [29, 10, 14, 37, 13]` `i = 0` `minIdx = 0`

Selection Sort Step by Step



arr[1] = 10 < arr[0] = 29 → **true** → **minIdx = 1**.

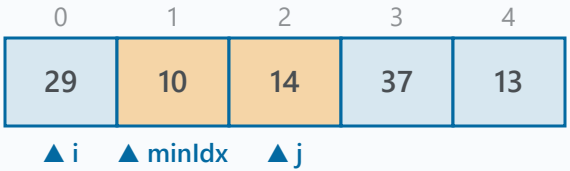
Step 2 / 18

```
1 for i = 0 to n-2
2     minIdx = i
      for j = i+1 to n-1
          if arr[j] < arr[minIdx] then minIdx =
  j
5     swap arr[i] and arr[minIdx]
```

VARIABLES

arr = [29, 10, 14, 37, 13] i = 0 j = 1 minIdx = 1 comps = 1

Selection Sort Step by Step



arr[2] = 14 < arr[1] = 10 → **false**.

Step 3 / 18

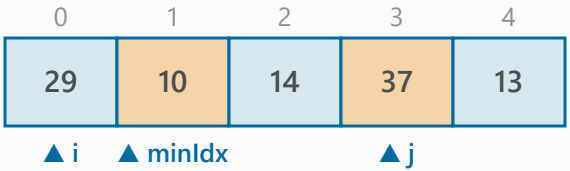
```
1 for i = 0 to n-2
2     minIdx = i
▶     for j = i+1 to n-1
▶         if arr[j] < arr[minIdx] then minIdx = j
5     swap arr[i] and arr[minIdx]
```

VARIABLES

`arr = [29, 10, 14, 37, 13]` `i = 0`

`j = 2` `minIdx = 1` `comps = 2`

Selection Sort Step by Step



arr[3] = 37 < arr[1] = 10 → **false**.

Step 4 / 18

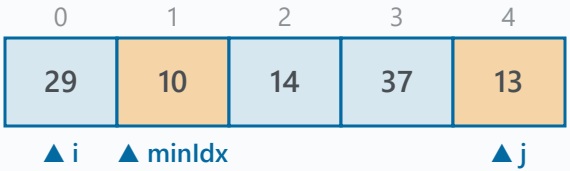
```
1 for i = 0 to n-2
2     minIdx = i
▶     for j = i+1 to n-1
▶         if arr[j] < arr[minIdx] then minIdx = j
5     swap arr[i] and arr[minIdx]
```

VARIABLES

`arr = [29, 10, 14, 37, 13]` `i = 0`

`j = 3` `minIdx = 1` `comps = 3`

Selection Sort Step by Step



arr[4] = 13 < arr[1] = 10 → **false**.

Step 5 / 18

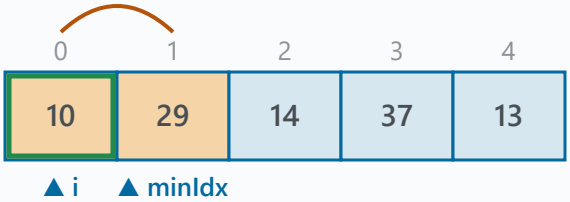
```
1 for i = 0 to n-2
2     minIdx = i
▶     for j = i+1 to n-1
▶         if arr[j] < arr[minIdx] then minIdx =
  j
5     swap arr[i] and arr[minIdx]
```

VARIABLES

arr = [29, 10, 14, 37, 13] i = 0

j = 4 minIdx = 1 comps = 4

Selection Sort Step by Step



Swap arr[0] and arr[1] → 10 is in place.

Step 6 / 18

```
1 for i = 0 to n-2
2     minIdx = i
3     for j = i+1 to n-1
4         if arr[j] < arr[minIdx] then minIdx =
  j
▶     swap arr[i] and arr[minIdx]
```

VARIABLES

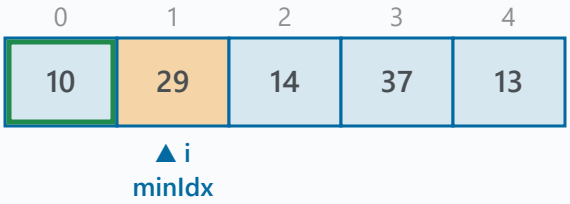
arr = [10, 29, 14, 37, 13]

i = 0

minIdx = 1

comps = 4

Selection Sort Step by Step



i = 1: the unsorted part is 1…4. Candidate minimum: arr[1] = 29.

Step 7 / 18

```
▶ for i = 0 to n-2
▶     minIdx = i
3     for j = i+1 to n-1
4         if arr[j] < arr[minIdx] then minIdx =
  j
5     swap arr[i] and arr[minIdx]
```

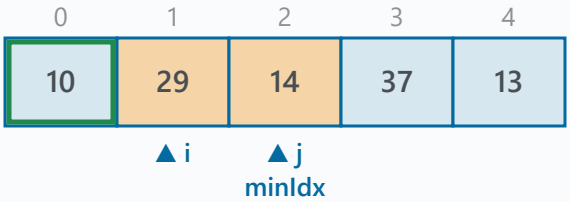
VARIABLES

arr = [10, 29, 14, 37, 13]

i = 1

minIdx = 1

Selection Sort Step by Step



arr[2] = 14 < arr[1] = 29 → **true** → **minIdx = 2**.

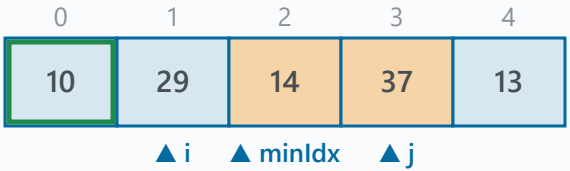
Step 8 / 18

```
1 for i = 0 to n-2
2     minIdx = i
▶     for j = i+1 to n-1
▶         if arr[j] < arr[minIdx] then minIdx =
  j
5     swap arr[i] and arr[minIdx]
```

VARIABLES

arr = [10, 29, 14, 37, 13] i = 1 j = 2 minIdx = 2 comps = 5

Selection Sort Step by Step



arr[3] = 37 < arr[2] = 14 → **false**.

Step 9 / 18

```
1 for i = 0 to n-2
2     minIdx = i
▶     for j = i+1 to n-1
▶         if arr[j] < arr[minIdx] then minIdx = j
5     swap arr[i] and arr[minIdx]
```

VARIABLES

arr = [10, 29, 14, 37, 13] | i = 1 | j = 3 | minIdx = 2 | comps = 6

Selection Sort Step by Step



arr[4] = 13 < arr[2] = 14 → **true** → **minIdx = 4**.

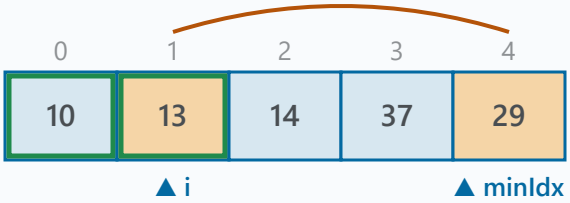
Step 10 / 18

```
1 for i = 0 to n-2
2     minIdx = i
▶     for j = i+1 to n-1
▶         if arr[j] < arr[minIdx] then minIdx =
  j
5     swap arr[i] and arr[minIdx]
```

VARIABLES

arr = [10, 29, 14, 37, 13] i = 1 j = 4 minIdx = 4 comps = 7

Selection Sort Step by Step



Swap arr[1] and arr[4] → 13 is in place.

Step 11 / 18

```
1 for i = 0 to n-2
2     minIdx = i
3     for j = i+1 to n-1
4         if arr[j] < arr[minIdx] then minIdx =
  j
▶     swap arr[i] and arr[minIdx]
```

VARIABLES

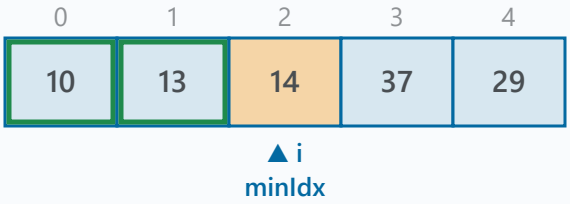
arr = [10, 13, 14, 37, 29]

i = 1

minIdx = 4

comps = 7

Selection Sort Step by Step



i = 2: the unsorted part is 2…4. Candidate minimum: arr[2] = 14.

Step 12 / 18

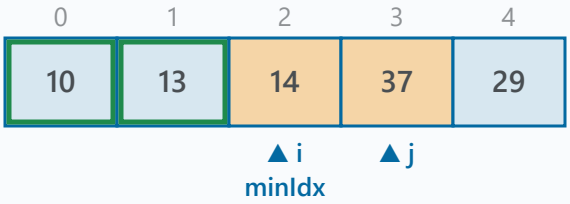
```
▶ for i = 0 to n-2
▶     minIdx = i
3     for j = i+1 to n-1
4         if arr[j] < arr[minIdx] then minIdx =
  j
5     swap arr[i] and arr[minIdx]
```

VARIABLES

arr = [10, 13, 14, 37, 29] i = 2

minIdx = 2

Selection Sort Step by Step



arr[3] = 37 < arr[2] = 14 → **false**.

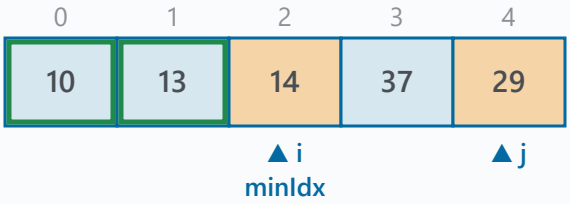
Step 13 / 18

```
1 for i = 0 to n-2
2     minIdx = i
▶     for j = i+1 to n-1
▶         if arr[j] < arr[minIdx] then minIdx =
  j
5     swap arr[i] and arr[minIdx]
```

VARIABLES

arr = [10, 13, 14, 37, 29] i = 2 j = 3 minIdx = 2 comps = 8

Selection Sort Step by Step



arr[4] = 29 < arr[2] = 14 → **false**.

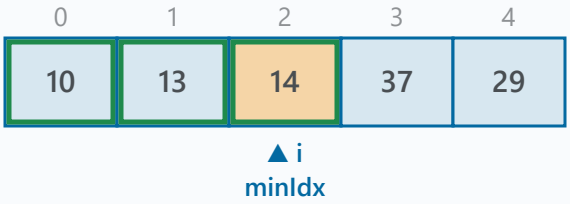
Step 14 / 18

```
1 for i = 0 to n-2
2     minIdx = i
▶     for j = i+1 to n-1
▶         if arr[j] < arr[minIdx] then minIdx =
  j
5     swap arr[i] and arr[minIdx]
```

VARIABLES

arr = [10, 13, 14, 37, 29] i = 2 j = 4 minIdx = 2 comps = 9

Selection Sort Step by Step



The minimum is already in place (minIdx = i), so the swap has no effect.

Step 15 / 18

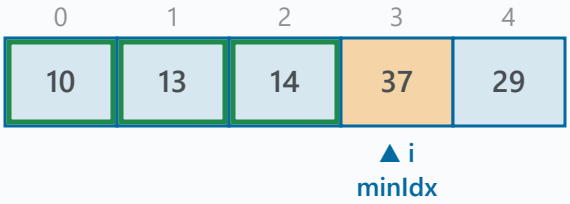
```
1 for i = 0 to n-2
2     minIdx = i
3     for j = i+1 to n-1
4         if arr[j] < arr[minIdx] then minIdx =
  j
▶     swap arr[i] and arr[minIdx]
```

VARIABLES

`arr = [10, 13, 14, 37, 29]` `i = 2`

`minIdx = 2` `comps = 9`

Selection Sort Step by Step



i = 3: the unsorted part is 3…4. Candidate minimum: arr[3] = 37.

Step 16 / 18

```
▶ for i = 0 to n-2
▶     minIdx = i
3     for j = i+1 to n-1
4         if arr[j] < arr[minIdx] then minIdx =
  j
5     swap arr[i] and arr[minIdx]
```

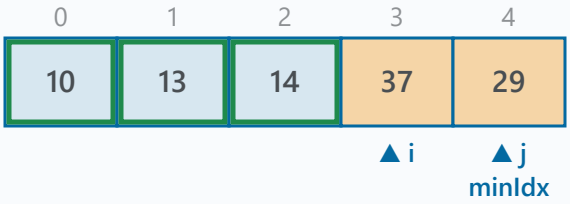
VARIABLES

arr = [10, 13, 14, 37, 29]

i = 3

minIdx = 3

Selection Sort Step by Step



arr[4] = 29 < arr[3] = 37 → **true** → **minIdx = 4**.

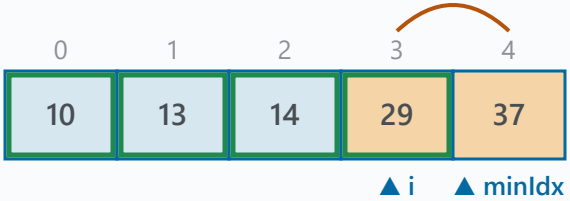
Step 17 / 18

```
1 for i = 0 to n-2
2     minIdx = i
▶     for j = i+1 to n-1
▶         if arr[j] < arr[minIdx] then minIdx =
  j
5     swap arr[i] and arr[minIdx]
```

VARIABLES

arr = [10, 13, 14, 37, 29] | i = 3 | j = 4 | minIdx = 4 | comps = 10

Selection Sort Step by Step



Swap arr[3] and arr[4] → 29 is in place. Sorted, with 10 comparisons = n(n−1)/2.

Step 18 / 18

```
1 for i = 0 to n-2
2     minIdx = i
3     for j = i+1 to n-1
4         if arr[j] < arr[minIdx] then minIdx =
  j
▶     swap arr[i] and arr[minIdx]
```

VARIABLES

`arr = [10, 13, 14, 29, 37]` `i = 3`

`minIdx = 4` `comps = 10`

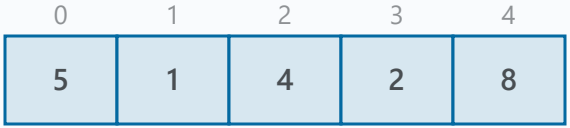
SORTING

Bubble Sort

- Compares adjacent elements and swaps them if they are out of order.
- Each pass carries the largest remaining element to the end; the next pass is one element shorter.
- If a pass makes no swaps, the array is sorted and we stop.
- $O(n^2)$ in general; best case (already sorted) is a single pass: O(n).

```
round = 0
repeat
    swapped = false
    for j = 0 to n-2-round
        if arr[j] > arr[j+1] then
            swap arr[j] and arr[j+1]
            swapped = true
    round = round + 1
until swapped is false
```

Bubble Sort Step by Step (pass 1)



round = 0. Adjacent pairs are compared from left to right.

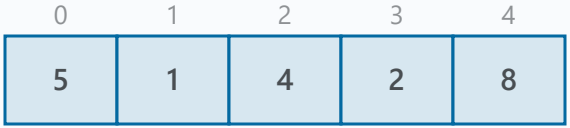
Step 1 / 11

```
▶ round = 0
2 repeat
3     swapped = false
4     for j = 0 to n-2-round
5         if arr[j] > arr[j+1] then
6             swap arr[j] and arr[j+1]
7             swapped = true
8     round = round + 1
9 until swapped is false
```

VARIABLES

`arr = [5, 1, 4, 2, 8]` `round = 0`

Bubble Sort Step by Step (pass 1)



Pass 1: swapped = false; j = 0…3.

Step 2 / 11

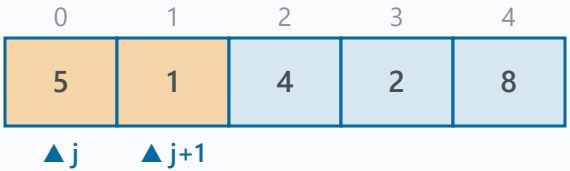
```
1 round = 0
2 repeat
▶     swapped = false
4     for j = 0 to n-2-round
5         if arr[j] > arr[j+1] then
6             swap arr[j] and arr[j+1]
7             swapped = true
8     round = round + 1
9 until swapped is false
```

VARIABLES

`arr = [5, 1, 4, 2, 8]` `round = 0`

`swapped = false`

Bubble Sort Step by Step (pass 1)



arr[0] = 5 > arr[1] = 1 → **true** → swap.

Step 3 / 11

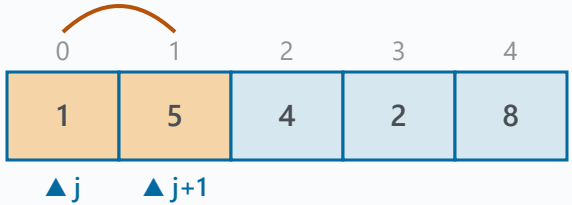
```
1 round = 0
2 repeat
3     swapped = false
4     for j = 0 to n-2-round
▶         if arr[j] > arr[j+1] then
6             swap arr[j] and arr[j+1]
7             swapped = true
8     round = round + 1
9 until swapped is false
```

VARIABLES

`arr = [5, 1, 4, 2, 8]` `round = 0`

`j = 0` `swapped = false`

Bubble Sort Step by Step (pass 1)



Swapped: 5 moved one step right; **swapped = true**.

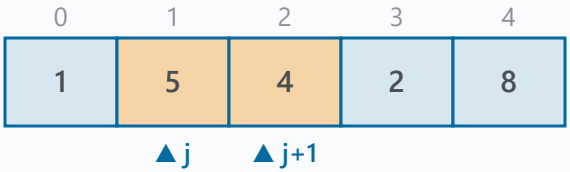
Step 4 / 11

```
1 round = 0
2 repeat
3     swapped = false
4     for j = 0 to n-2-round
5         if arr[j] > arr[j+1] then
▶             swap arr[j] and arr[j+1]
▶             swapped = true
8     round = round + 1
9 until swapped is false
```

VARIABLES

`arr = [1, 5, 4, 2, 8]` `round = 0` `j = 0` `swapped = true`

Bubble Sort Step by Step (pass 1)



arr[1] = 5 > arr[2] = 4 → **true** → swap.

Step 5 / 11

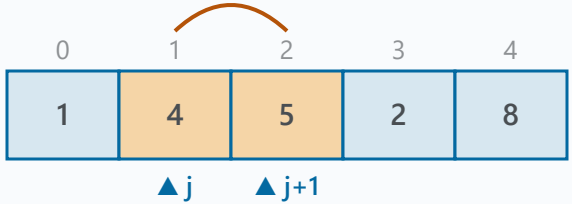
```
1 round = 0
2 repeat
3     swapped = false
4     for j = 0 to n-2-round
▶         if arr[j] > arr[j+1] then
6             swap arr[j] and arr[j+1]
7             swapped = true
8     round = round + 1
9 until swapped is false
```

VARIABLES

`arr = [1, 5, 4, 2, 8]` `round = 0`

`j = 1` `swapped = true`

Bubble Sort Step by Step (pass 1)



Swapped: 5 moved one step right; **swapped = true**.

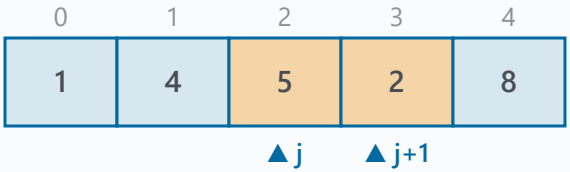
Step 6 / 11

```
1 round = 0
2 repeat
3     swapped = false
4     for j = 0 to n-2-round
5         if arr[j] > arr[j+1] then
▶             swap arr[j] and arr[j+1]
▶             swapped = true
8     round = round + 1
9 until swapped is false
```

VARIABLES

`arr = [1, 4, 5, 2, 8]` `round = 0` `j = 1` `swapped = true`

Bubble Sort Step by Step (pass 1)



arr[2] = 5 > arr[3] = 2 → **true** → swap.

Step 7 / 11

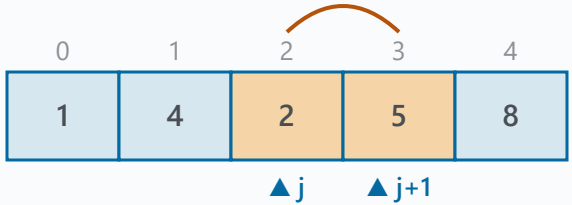
```
1 round = 0
2 repeat
3     swapped = false
4     for j = 0 to n-2-round
▶         if arr[j] > arr[j+1] then
6             swap arr[j] and arr[j+1]
7             swapped = true
8     round = round + 1
9 until swapped is false
```

VARIABLES

`arr = [1, 4, 5, 2, 8]` `round = 0`

`j = 2` `swapped = true`

Bubble Sort Step by Step (pass 1)



Swapped: 5 moved one step right; **swapped = true**.

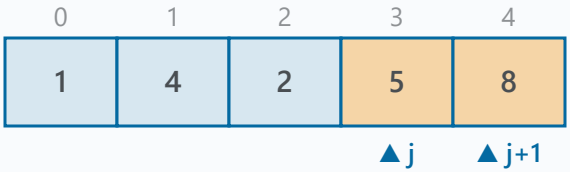
Step 8 / 11

```
1 round = 0
2 repeat
3     swapped = false
4     for j = 0 to n-2-round
5         if arr[j] > arr[j+1] then
▶             swap arr[j] and arr[j+1]
▶             swapped = true
8     round = round + 1
9 until swapped is false
```

VARIABLES

`arr = [1, 4, 2, 5, 8]` `round = 0` `j = 2` `swapped = true`

Bubble Sort Step by Step (pass 1)



arr[3] = 5 > arr[4] = 8 → **false**, already in order.

Step 9 / 11

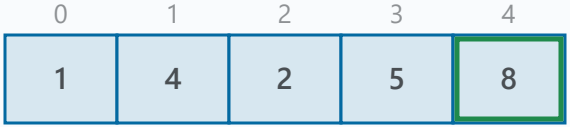
```
1 round = 0
2 repeat
3     swapped = false
4     for j = 0 to n-2-round
▶         if arr[j] > arr[j+1] then
6             swap arr[j] and arr[j+1]
7             swapped = true
8     round = round + 1
9 until swapped is false
```

VARIABLES

`arr = [1, 4, 2, 5, 8]` `round = 0`

`j = 3` `swapped = true`

Bubble Sort Step by Step (pass 1)



Pass done: **8** is in its final place (green border). **round = 1**.

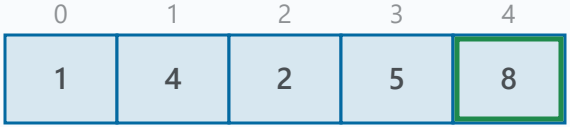
Step 10 / 11

```
1 round = 0
2 repeat
3     swapped = false
4     for j = 0 to n-2-round
5         if arr[j] > arr[j+1] then
6             swap arr[j] and arr[j+1]
7             swapped = true
▶     round = round + 1
9 until swapped is false
```

VARIABLES

`arr = [1, 4, 2, 5, 8]` `round = 1` `swapped = true`

Bubble Sort Step by Step (pass 1)



swapped = true → another pass is needed.

Step 11 / 11

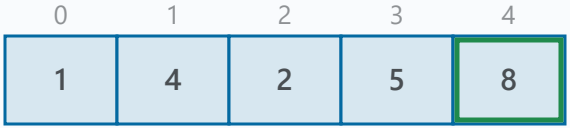
```
1 round = 0
2 repeat
3     swapped = false
4     for j = 0 to n-2-round
5         if arr[j] > arr[j+1] then
6             swap arr[j] and arr[j+1]
7             swapped = true
8     round = round + 1
▶ until swapped is false
```

VARIABLES

`arr = [1, 4, 2, 5, 8]` `round = 1`

`swapped = true`

Bubble Sort Step by Step (cont.: pass 2)



Pass 2: swapped = false; j = 0…2.

Step 1 / 7

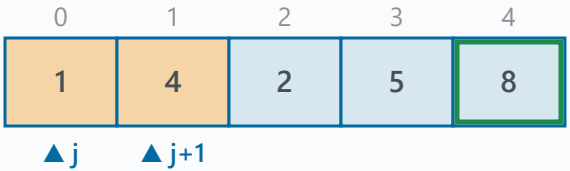
```
1 round = 0
2 repeat
▶     swapped = false
4     for j = 0 to n-2-round
5         if arr[j] > arr[j+1] then
6             swap arr[j] and arr[j+1]
7             swapped = true
8     round = round + 1
9 until swapped is false
```

VARIABLES

`arr = [1, 4, 2, 5, 8]` `round = 1`

`swapped = false`

Bubble Sort Step by Step (cont.: pass 2)



arr[0] = 1 > arr[1] = 4 → **false**, already in order.

Step 2 / 7

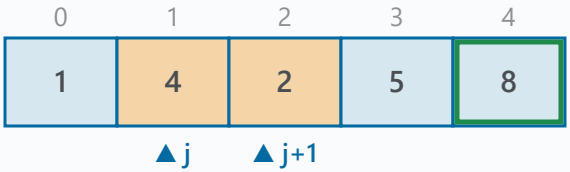
```
1 round = 0
2 repeat
3     swapped = false
4     for j = 0 to n-2-round
          if arr[j] > arr[j+1] then
6             swap arr[j] and arr[j+1]
7             swapped = true
8     round = round + 1
9 until swapped is false
```

VARIABLES

`arr = [1, 4, 2, 5, 8]` `round = 1`

`j = 0` `swapped = false`

Bubble Sort Step by Step (cont.: pass 2)



arr[1] = 4 > arr[2] = 2 → **true** → swap.

Step 3 / 7

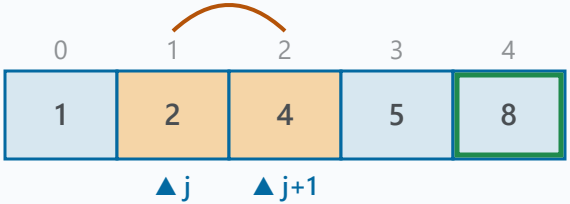
```
1 round = 0
2 repeat
3     swapped = false
4     for j = 0 to n-2-round
▶         if arr[j] > arr[j+1] then
6             swap arr[j] and arr[j+1]
7             swapped = true
8     round = round + 1
9 until swapped is false
```

VARIABLES

`arr = [1, 4, 2, 5, 8]` `round = 1`

`j = 1` `swapped = false`

Bubble Sort Step by Step (cont.: pass 2)



Swapped: 4 moved one step right; **swapped = true**.

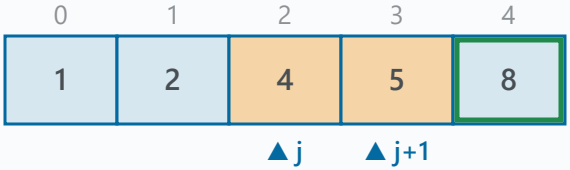
Step 4 / 7

```
1 round = 0
2 repeat
3     swapped = false
4     for j = 0 to n-2-round
5         if arr[j] > arr[j+1] then
▶             swap arr[j] and arr[j+1]
▶             swapped = true
8     round = round + 1
9 until swapped is false
```

VARIABLES

`arr = [1, 2, 4, 5, 8]` `round = 1` `j = 1` `swapped = true`

Bubble Sort Step by Step (cont.: pass 2)



arr[2] = 4 > arr[3] = 5 → **false**, already in order.

Step 5 / 7

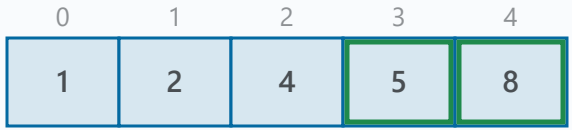
```
1 round = 0
2 repeat
3     swapped = false
4     for j = 0 to n-2-round
▶         if arr[j] > arr[j+1] then
6             swap arr[j] and arr[j+1]
7             swapped = true
8     round = round + 1
9 until swapped is false
```

VARIABLES

`arr = [1, 2, 4, 5, 8]` `round = 1`

`j = 2` `swapped = true`

Bubble Sort Step by Step (cont.: pass 2)



Pass done: **5** is in its final place (green border). **round = 2**.

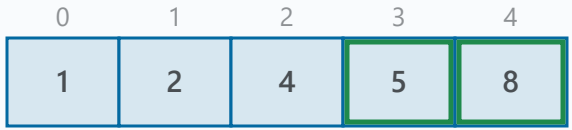
Step 6 / 7

```
1 round = 0
2 repeat
3     swapped = false
4     for j = 0 to n-2-round
5         if arr[j] > arr[j+1] then
6             swap arr[j] and arr[j+1]
7             swapped = true
▶     round = round + 1
9 until swapped is false
```

VARIABLES

`arr = [1, 2, 4, 5, 8]` `round = 2` `swapped = true`

Bubble Sort Step by Step (cont.: pass 2)



swapped = true → another pass is needed.

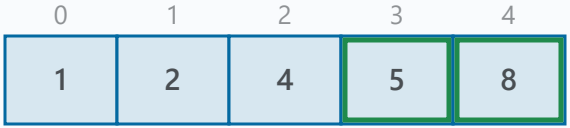
Step 7 / 7

```
1 round = 0
2 repeat
3     swapped = false
4     for j = 0 to n-2-round
5         if arr[j] > arr[j+1] then
6             swap arr[j] and arr[j+1]
7             swapped = true
8     round = round + 1
▶ until swapped is false
```

VARIABLES

`arr = [1, 2, 4, 5, 8]` `round = 2` `swapped = true`

Bubble Sort Step by Step (cont.: pass 3)



Pass 3: swapped = false; j = 0…1.

Step 1 / 5

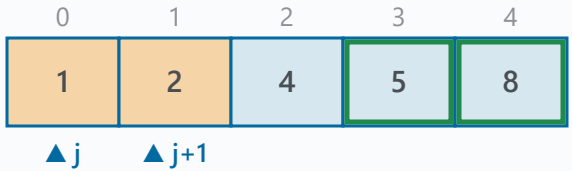
```
1 round = 0
2 repeat
      swapped = false
4     for j = 0 to n-2-round
5         if arr[j] > arr[j+1] then
6             swap arr[j] and arr[j+1]
7             swapped = true
8     round = round + 1
9 until swapped is false
```

VARIABLES

`arr = [1, 2, 4, 5, 8]` `round = 2`

`swapped = false`

Bubble Sort Step by Step (cont.: pass 3)



arr[0] = 1 > arr[1] = 2 → **false**, already in order.

Step 2 / 5

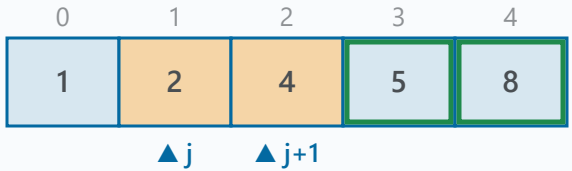
```
1 round = 0
2 repeat
3     swapped = false
4     for j = 0 to n-2-round
          if arr[j] > arr[j+1] then
6             swap arr[j] and arr[j+1]
7             swapped = true
8     round = round + 1
9 until swapped is false
```

VARIABLES

`arr = [1, 2, 4, 5, 8]` `round = 2`

`j = 0` `swapped = false`

Bubble Sort Step by Step (cont.: pass 3)



arr[1] = 2 > arr[2] = 4 → **false**, already in order.

Step 3 / 5

```
1 round = 0
2 repeat
3     swapped = false
4     for j = 0 to n-2-round
          if arr[j] > arr[j+1] then
6             swap arr[j] and arr[j+1]
7             swapped = true
8     round = round + 1
9 until swapped is false
```

VARIABLES

`arr = [1, 2, 4, 5, 8]` `round = 2`

`j = 1` `swapped = false`

Bubble Sort Step by Step (cont.: pass 3)



Pass done: **4** is in its final place (green border). **round = 3**.

Step 4 / 5

```
1 round = 0
2 repeat
3     swapped = false
4     for j = 0 to n-2-round
5         if arr[j] > arr[j+1] then
6             swap arr[j] and arr[j+1]
7             swapped = true
▶     round = round + 1
9 until swapped is false
```

VARIABLES

`arr = [1, 2, 4, 5, 8]` `round = 3` `swapped = false`

Bubble Sort Step by Step (cont.: pass 3)



swapped = false → no swaps at all, so the array is sorted; **stop**. 9 comparisons in total.

Step 5 / 5

```
1 round = 0
2 repeat
3     swapped = false
4     for j = 0 to n-2-round
5         if arr[j] > arr[j+1] then
6             swap arr[j] and arr[j+1]
7             swapped = true
8     round = round + 1
▶ until swapped is false
```

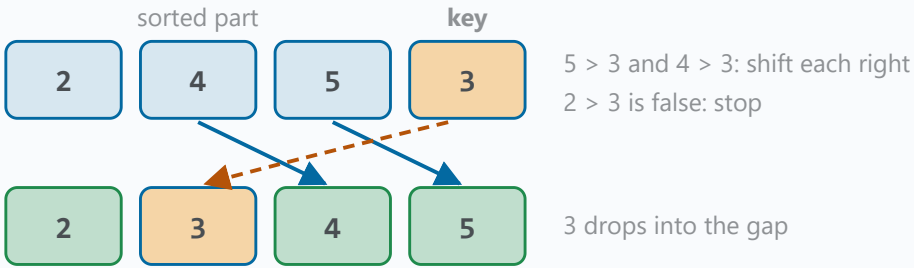
VARIABLES

`arr = [1, 2, 4, 5, 8]` `round = 3` `swapped = false`

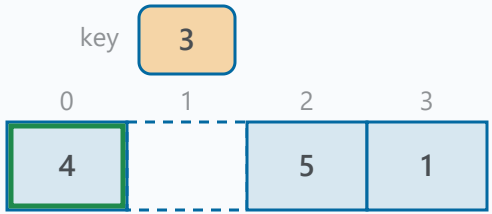
Insertion Sort

- Like arranging playing cards in your hand: the left part is always sorted.
- Take the next element (the **key**), shift larger ones one step right, and drop the key into the gap.
- Worst case (reverse order) $O(n^2)$; on a sorted array each key needs one comparison: $O(n)$. $O(1)$ extra memory.
- Very fast on small or nearly sorted data.

```
for i = 1 to n-1
    key = arr[i]
    j = i - 1
    while j >= 0 and arr[j] > key
        arr[j+1] = arr[j]
        j = j - 1
    arr[j+1] = key
```



Insertion Sort Step by Step



i = 1: take key = arr[1] = **3**, leaving a gap. The left part (0…0) is sorted. j = 0.

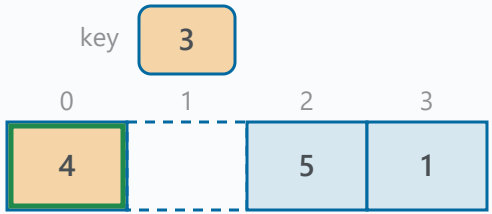
Step 1 / 17

```
▶ for i = 1 to n-1
▶     key = arr[i]
▶     j = i - 1
4     while j >= 0 and arr[j] > key
5         arr[j+1] = arr[j]
6         j = j - 1
7     arr[j+1] = key
```

VARIABLES

`arr = [4, 3, 5, 1]` `i = 1` `key = 3` `j = 0`

Insertion Sort Step by Step



j = 0 ≥ 0 and arr[0] = 4 > 3 → **true**.

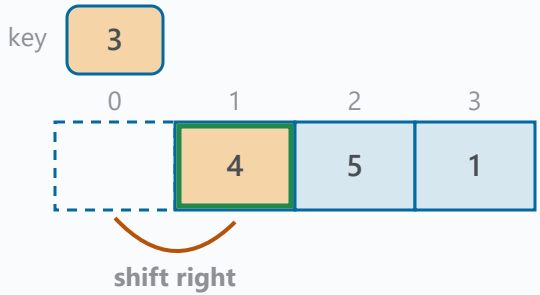
Step 2 / 17

```
1 for i = 1 to n-1
2     key = arr[i]
3     j = i - 1
▶     while j >= 0 and arr[j] > key
5         arr[j+1] = arr[j]
6         j = j - 1
7     arr[j+1] = key
```

VARIABLES

`arr = [4, _, 5, 1]` `i = 1` `key = 3` `j = 0`

Insertion Sort Step by Step



4 moved one step right; the gap is now cell 0. **j = -1**.

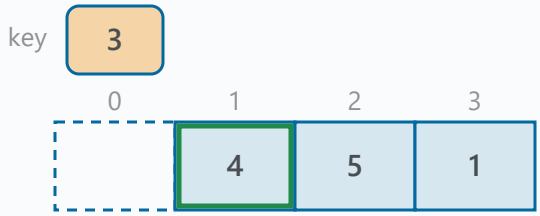
Step 3 / 17

```
1 for i = 1 to n-1
2     key = arr[i]
3     j = i - 1
4     while j >= 0 and arr[j] > key
          arr[j+1] = arr[j]
          j = j - 1
7     arr[j+1] = key
```

VARIABLES

`arr = [_, 4, 5, 1]` `i = 1` `key = 3` `j = -1`

Insertion Sort Step by Step



j = -1 < 0 → **false** (we hit the left end), so the loop ends.

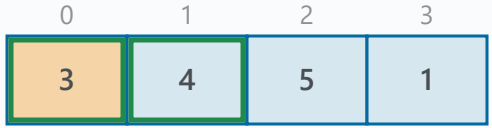
Step 4 / 17

```
1 for i = 1 to n-1
2     key = arr[i]
3     j = i - 1
▶     while j >= 0 and arr[j] > key
5         arr[j+1] = arr[j]
6         j = j - 1
7     arr[j+1] = key
```

VARIABLES

`arr = [_, 4, 5, 1]` `i = 1` `key = 3` `j = -1`

Insertion Sort Step by Step



arr[0] = 3: the key drops into the gap. Now 0…1 is sorted.

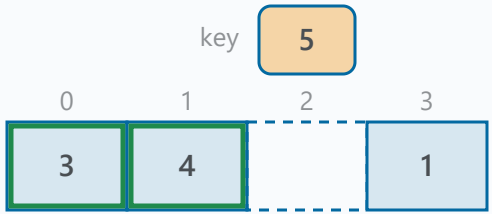
Step 5 / 17

```
1 for i = 1 to n-1
2     key = arr[i]
3     j = i - 1
4     while j >= 0 and arr[j] > key
5         arr[j+1] = arr[j]
6         j = j - 1
▶     arr[j+1] = key
```

VARIABLES

`arr = [3, 4, 5, 1]` `i = 1` `key = 3` `j = -1`

Insertion Sort Step by Step



i = 2: take key = arr[2] = **5**, leaving a gap. The left part (0…1) is sorted. j = 1.

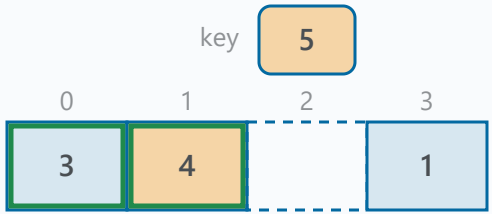
Step 6 / 17

```
for i = 1 to n-1
    key = arr[i]
    j = i - 1
    while j >= 0 and arr[j] > key
        arr[j+1] = arr[j]
        j = j - 1
    arr[j+1] = key
```

VARIABLES

arr = [3, 4, 5, 1] i = 2 key = 5 j = 1

Insertion Sort Step by Step



j = 1 ≥ 0 and arr[1] = 4 > 5 → **false**, so the loop ends.


```
1 for i = 1 to n-1
2     key = arr[i]
3     j = i - 1
▶     while j >= 0 and arr[j] > key
5         arr[j+1] = arr[j]
6         j = j - 1
7     arr[j+1] = key
```

VARIABLES

`arr = [3, 4, _, 1]` `i = 2` `key = 5` `j = 1`

Insertion Sort Step by Step



arr[2] = 5: the key drops into the gap. Now 0…2 is sorted.

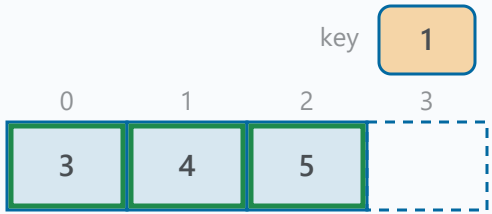
Step 8 / 17

```
1 for i = 1 to n-1
2     key = arr[i]
3     j = i - 1
4     while j >= 0 and arr[j] > key
5         arr[j+1] = arr[j]
6         j = j - 1
▶     arr[j+1] = key
```

VARIABLES

arr = [3, 4, 5, 1] i = 2 key = 5 j = 1

Insertion Sort Step by Step



i = 3: take key = arr[3] = **1**, leaving a gap. The left part (0…2) is sorted. j = 2.

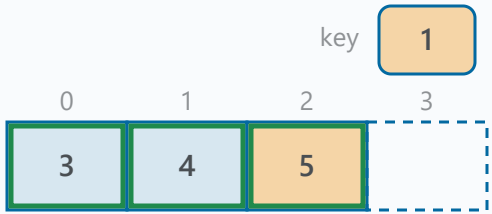
Step 9 / 17

```
for i = 1 to n-1
    key = arr[i]
    j = i - 1
    while j >= 0 and arr[j] > key
        arr[j+1] = arr[j]
        j = j - 1
    arr[j+1] = key
```

VARIABLES

arr = [3, 4, 5, 1] | i = 3 | key = 1 | j = 2

Insertion Sort Step by Step



j = 2 ≥ 0 and arr[2] = 5 > 1 → **true**.

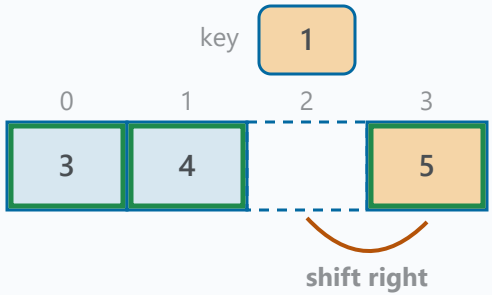
Step 10 / 17

```
1 for i = 1 to n-1
2     key = arr[i]
3     j = i - 1
▶     while j >= 0 and arr[j] > key
5         arr[j+1] = arr[j]
6         j = j - 1
7     arr[j+1] = key
```

VARIABLES

`arr = [3, 4, 5, _]` `i = 3` `key = 1` `j = 2`

Insertion Sort Step by Step



5 moved one step right; the gap is now cell 2. **j = 1**.

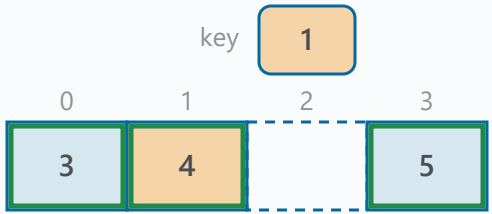
Step 11 / 17

```
1 for i = 1 to n-1
2     key = arr[i]
3     j = i - 1
4     while j >= 0 and arr[j] > key
          arr[j+1] = arr[j]
          j = j - 1
7     arr[j+1] = key
```

VARIABLES

`arr = [3, 4, _, 5]` `i = 3` `key = 1` `j = 1`

Insertion Sort Step by Step



j = 1 ≥ 0 and arr[1] = 4 > 1 → **true**.

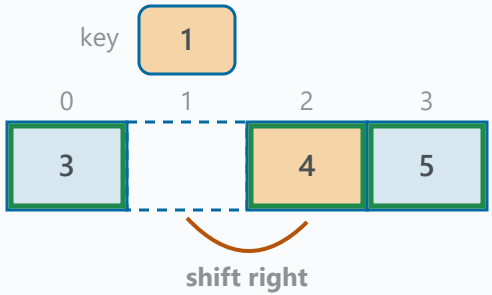
Step 12 / 17

```
1 for i = 1 to n-1
2     key = arr[i]
3     j = i - 1
▶     while j >= 0 and arr[j] > key
5         arr[j+1] = arr[j]
6         j = j - 1
7     arr[j+1] = key
```

VARIABLES

`arr = [3, 4, _, 5]` `i = 3` `key = 1` `j = 1`

Insertion Sort Step by Step



4 moved one step right; the gap is now cell 1. **j = 0**.

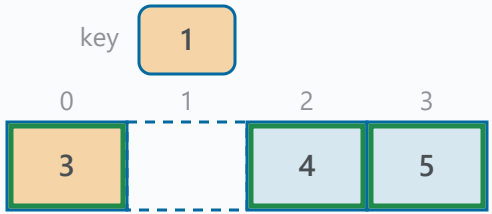
Step 13 / 17

```
1 for i = 1 to n-1
2     key = arr[i]
3     j = i - 1
4     while j >= 0 and arr[j] > key
          arr[j+1] = arr[j]
          j = j - 1
7     arr[j+1] = key
```

VARIABLES

`arr = [3, _, 4, 5]` `i = 3` `key = 1` `j = 0`

Insertion Sort Step by Step



j = 0 ≥ 0 and arr[0] = 3 > 1 → **true**.

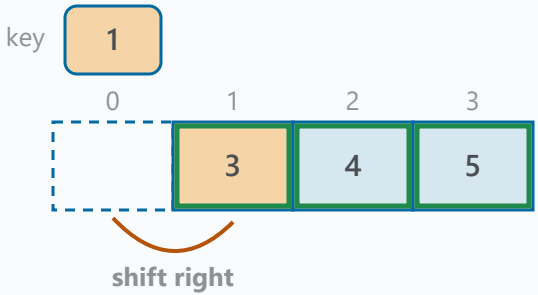
Step 14 / 17

```
1 for i = 1 to n-1
2     key = arr[i]
3     j = i - 1
▶     while j >= 0 and arr[j] > key
5         arr[j+1] = arr[j]
6         j = j - 1
7     arr[j+1] = key
```

VARIABLES

`arr = [3, _, 4, 5]` `i = 3` `key = 1` `j = 0`

Insertion Sort Step by Step



3 moved one step right; the gap is now cell 0. **j = -1**.

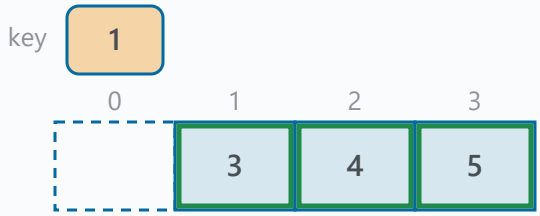
Step 15 / 17

```
1 for i = 1 to n-1
2     key = arr[i]
3     j = i - 1
4     while j >= 0 and arr[j] > key
          arr[j+1] = arr[j]
          j = j - 1
7     arr[j+1] = key
```

VARIABLES

`arr = [_, 3, 4, 5]` `i = 3` `key = 1` `j = -1`

Insertion Sort Step by Step



j = -1 < 0 → **false** (we hit the left end), so the loop ends.

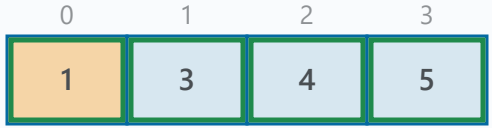
Step 16 / 17

```
1 for i = 1 to n-1
2     key = arr[i]
3     j = i - 1
▶     while j >= 0 and arr[j] > key
5         arr[j+1] = arr[j]
6         j = j - 1
7     arr[j+1] = key
```

VARIABLES

`arr = [_, 3, 4, 5]` `i = 3` `key = 1` `j = -1`

Insertion Sort Step by Step



arr[0] = 1: the key drops into the gap. Now 0…3 is sorted. Done, with 5 comparisons.

Step 17 / 17

```
1 for i = 1 to n-1
2     key = arr[i]
3     j = i - 1
4     while j >= 0 and arr[j] > key
5         arr[j+1] = arr[j]
6         j = j - 1
▶     arr[j+1] = key
```

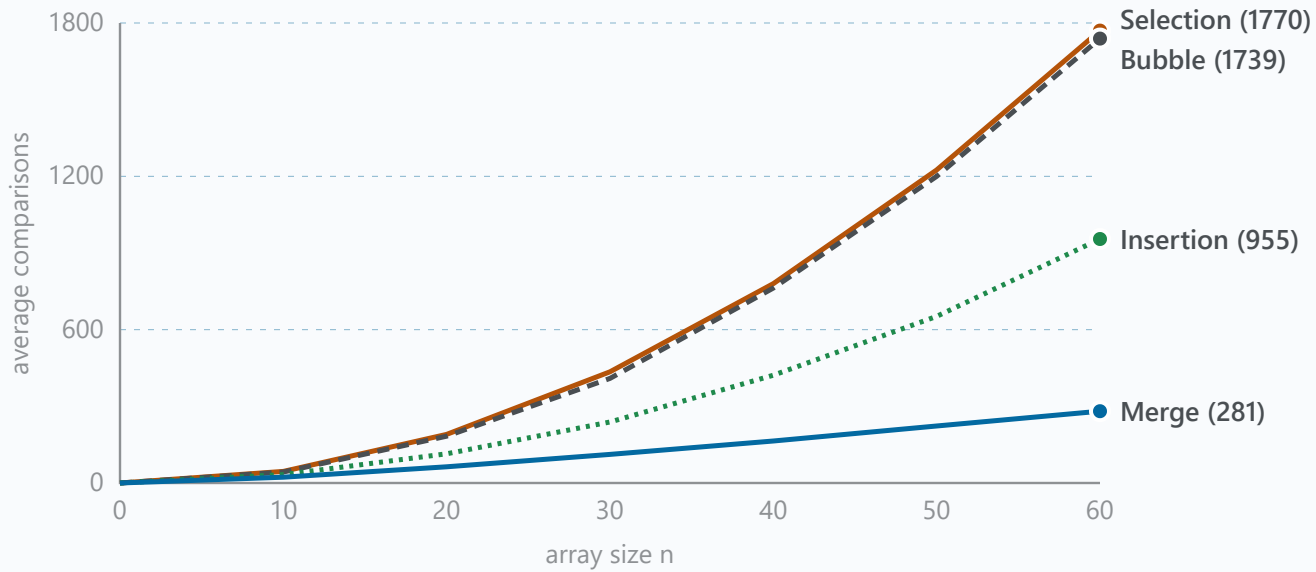
VARIABLES

arr = [1, 3, 4, 5] i = 3 key = 1 j = -1

Comparing the Sorts

Algorithm	Key idea	Worst	Best	Extra space
Selection	Select the smallest and place it	$O(n^2)$	$O(n^2)$	O(1)
Bubble	Swap neighbors	$O(n^2)$	O(n) (with early exit)	O(1)
Insertion	Insert into the sorted part	$O(n^2)$	O(n)	O(1)
Merge	Split, sort, merge	O(n log n)	O(n log n)	O(n)

Sorting: How Many Comparisons?



For each n, 20 random arrays were actually sorted by all four algorithms and the averages plotted. At n = 60, selection sort made 1770 comparisons; merge sort only 281.

SUMMARY

Chapter Summary

- An algorithm is a clear, finite set of steps that turns input into output.
- Pseudocode writes an algorithm without committing to a language.
- Hand tracing and a variety of test cases help assess correctness.
- Linear search needs no sorting; binary search halves the range on sorted data.
- Besides correctness, algorithms are compared by time and memory cost.

SUMMARY

Review Questions

1. What belongs in a good problem definition?
2. Why write pseudocode before choosing a programming language?
3. How does the factorial algorithm handle an input of 0?
4. Why does binary search need sorted data?
5. What does comparing O(n) with O(log n) tell us?
6. What is the key difference between selection sort and bubble sort?

SUMMARY

Exercises

1. Write pseudocode that prints three numbers (a, b, c) in ascending order; build a trace table for a = 9, b = 2, c = 5.
2. Write an algorithm that sums the **even** numbers from 1 to n. Trace it for n = 0 and n = 7.
3. Change the factorial algorithm to warn when n = 21 is entered (constraint $0 \le n \le 20$).
4. Sort [6, 2, 8, 3] by hand with selection, bubble and insertion sort; count the comparisons for each.
5. Use binary search to look for 27 and 4 in [2, 5, 9, 14, 20, 27, 33]; write down lo, hi and mid in every round.
6. Write an algorithm that sums the digits of a number (e.g. 472 → 13) using % and /.

▶ **Hints**

References

- Cormen, Leiserson, Rivest and Stein, Introduction to Algorithms, 4th ed., MIT Press, 2022.
- Levitin, Introduction to the Design and Analysis of Algorithms, 3rd ed., Pearson, 2012.
- Sedgewick and Wayne, Algorithms, 4th ed., Addison-Wesley, 2011 (**book site**).
- Sedgewick and Wayne, Computer Science: An Interdisciplinary Approach, Addison-Wesley, 2016: **4.1 Performance**, **4.2 Sorting and Searching**.
- Oracle, Java SE 21 API: **java.util.Arrays** (sort, binarySearch).

The End

Chapter 4.1: Algorithms

Previous: Chapter 3.3: Microsoft PowerPoint · Next: Chapter 4.2: Flowcharts