Design and Analysis of Algorithms

Dina El-Manakhly, Ph. D.

dina_almnakhly@science.suez.edu.eg

Strategy

- A **Strategy** is an approach or a design for solving a computational problem.

- **Example**
  - Greedy method
  - Dynamic programming
  - Backtracking
  - Branch and bound
  - Brute Force
  - Divide and conquer

Brute Force

- A brute-force algorithm solves a problem in the most simple, direct way.
- Brute Force search is the naive approach (intuitive).
- A brute force algorithm solves a problem through exhaustion: it goes through all possible choices until a solution is found.
- Example:

  If there is a lock of 4-digit PIN. The digits to be chosen from 0-9 then the brute force will be trying all possible combinations one by one like 0001, 0002, 0003, 0004, and so on until we get the right PIN. In the worst case, it will take 10,000 tries to find the right combination.
- Brute force algorithms are simple but very slow.

Some standard algorithms that follow Brute Force algorithm

- Linear Search.
- Selection Sort.
- Merging Problem.

Selection Sort

- **Problem Definition:** Given an array A=(a_1, a_2,…, a_n) of n elements. Sorting the array is rearrangement the elements of the array such that $a_i \leq a_{i+1}$, $1 \leq i \leq n$-1.

Examples

Example 1: Given A=(2,4,9,6,3,10,7,1)

Sort(A)=(1,2,3,4,6,7,9,10)

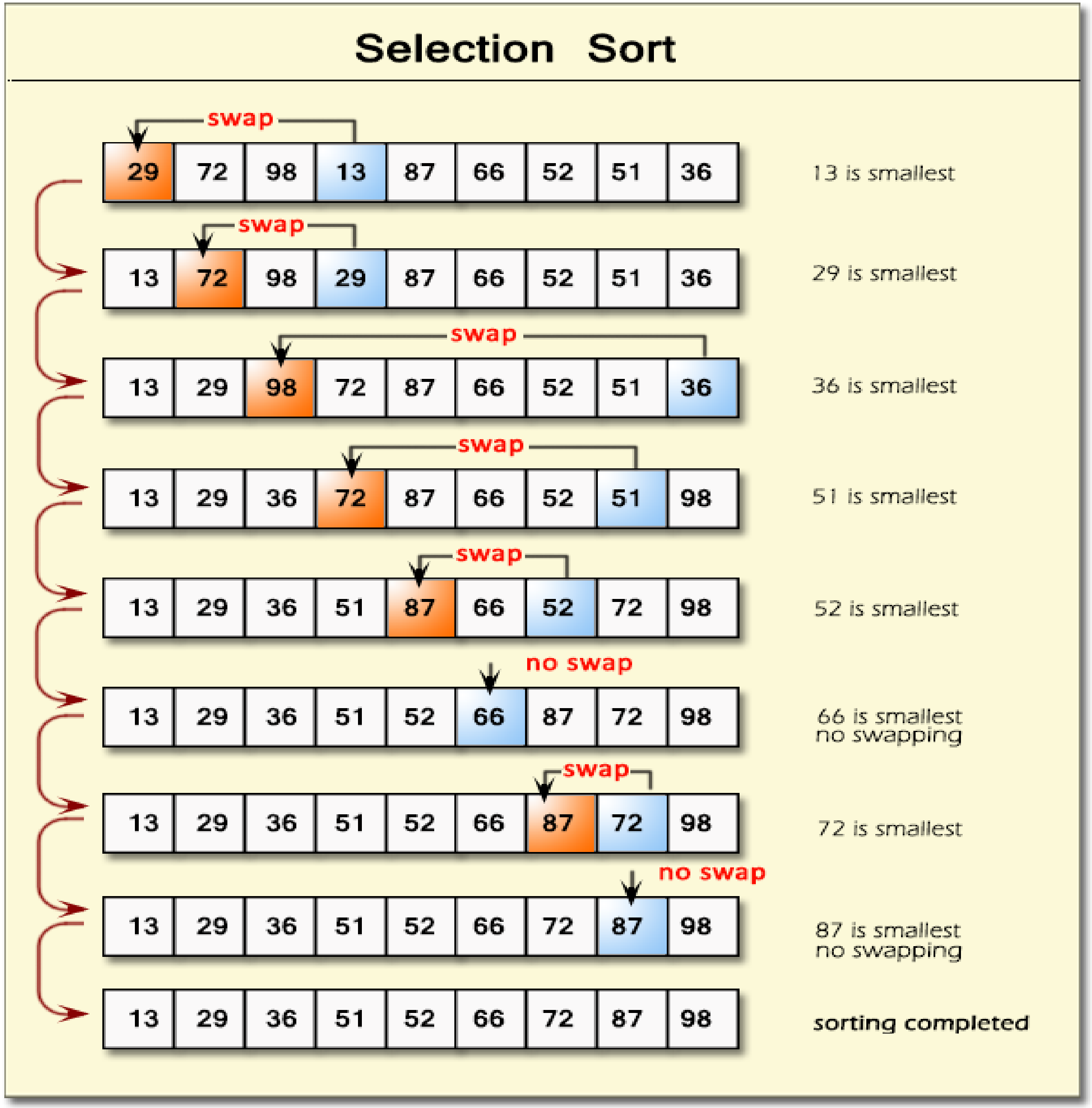
Example 2: Given A=(2,4,9,6,9,10,9,1)

Sort(A)=(1,2,4,6,9,9,9,10)

Selection Sort

The main idea of selection sort algorithm as follows:

- First, we find the minimum element of the array A and store it in a_1.
- Next, we find the minimum of the remaining n-1 elements and store it in a_2.
- We continue this way until the second largest element is stored in a_{n-1} and the largest element of A is stored in a_n.



Selection Sort Pseudo Code

```
for i = 1 to n − 1 do
    min = i
    for j = i + 1 to n do
        // Find the index of the i^th smallest element
        if A[j] < A[min] then
            min = j
        end if
    end for
    Swap A[min] and A[i]
end for
```

Merge two sorted arrays Problem

- **Problem Definition:** Given two sorted arrays $A=(a_1, a_2,\dots, a_n)$ and $B=(b_1, b_2,\dots, b_m)$ of $\boldsymbol{n}$ and $\boldsymbol{m}$ elements respectively. Merging the two sorted arrays is an array $C=(c_1, c_2,\dots, c_{n+m})$ of $\boldsymbol{n+m}$ elements such that:

(i) $c_i \in C$ belongs to A or B, $\forall\ 1 \le i \le n+m$.

(ii) a_i and b_j appear exactly once in C, $\forall\ 1 \le i \le n$ *and* $1 \le j \le m$.

Examples

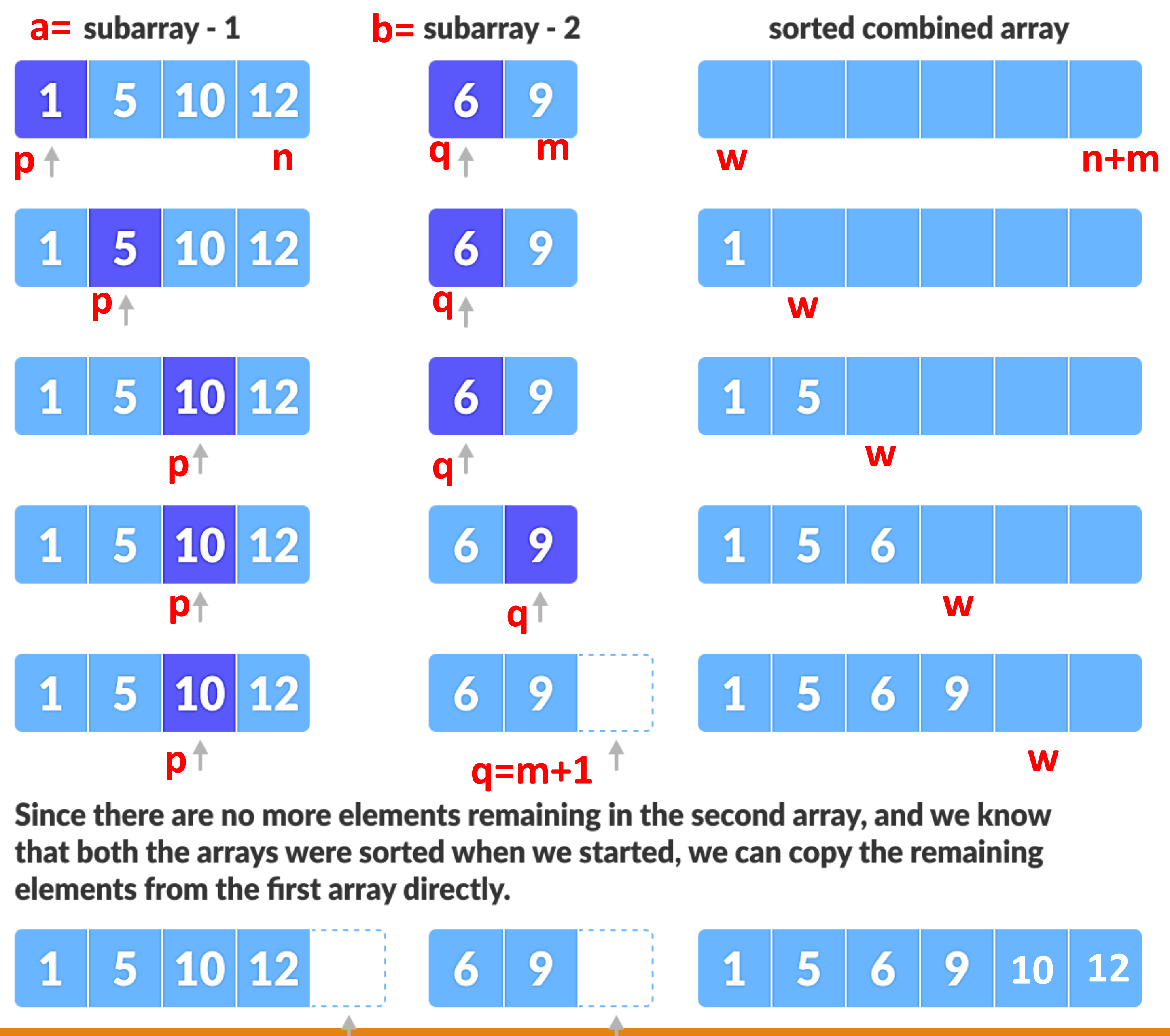
Example 1: Given A=(1,3,4,5,10) and *B*=(2,3,3,7,8)

Merge(A,B)=(1,2,3,3,3,4,5,7,8,10)

Merge two sorted arrays Problem

Main Idea: The main idea of merging algorithm as follows:

- We maintain two pointers **p** and **q** that initially point to a_1 and b_1 respectively.
- In each step, we compare the elements a_p and b_q . If a_p is less than or equal b_q then append a_p to the array C at position w. Then increment p and w by 1. Otherwise, append b_q to the array C at position w. Then increment q and w by 1.
- This process ends when p=n+1 or q=m+1. In case of p=n+1, we append the remaining elements B(q…m) to C(w…n+m). In the second case (q=m+1), we append A(p...n) to array C(w..n+m).



Pseudo Code

Algorithm: Merging

Input: Two sorted arrays $A=(a_1, a_2,\ldots, a_n)$ and $B=(b_1, b_2,\ldots, b_m)$ of n and m elements respectively.

Output: Sorted array $C=(c_1, c_2,\ldots, c_{n+m})$ s.t. (i) $c_i \in C$ belongs to A or B, $\forall\ 1 \le i \le n+m$. (ii) a_i and b_j appear exactly once in C, $\forall\ 1 \le i \le n$ and $1 \le j \le m$.

Begin

1. $p=q=w=1$

2. While $p \le n$ and $q \le m$ do

    if $a_p \le b_q$ Then

        $c_w = a_p$, p=p+1, w=w+1

    else $c_w = b_q$, q=q+1, w=w+1

3. If p > n then $C(c_w, c_{w+1},\ldots, c_{n+m})=B(b_q, b_{q+1},\ldots, b_m)$

    if q> m then $C(c_w, c_{w+1},\ldots, c_{n+m})=A(a_p, a_{p+1},\ldots, a_n)$

End.

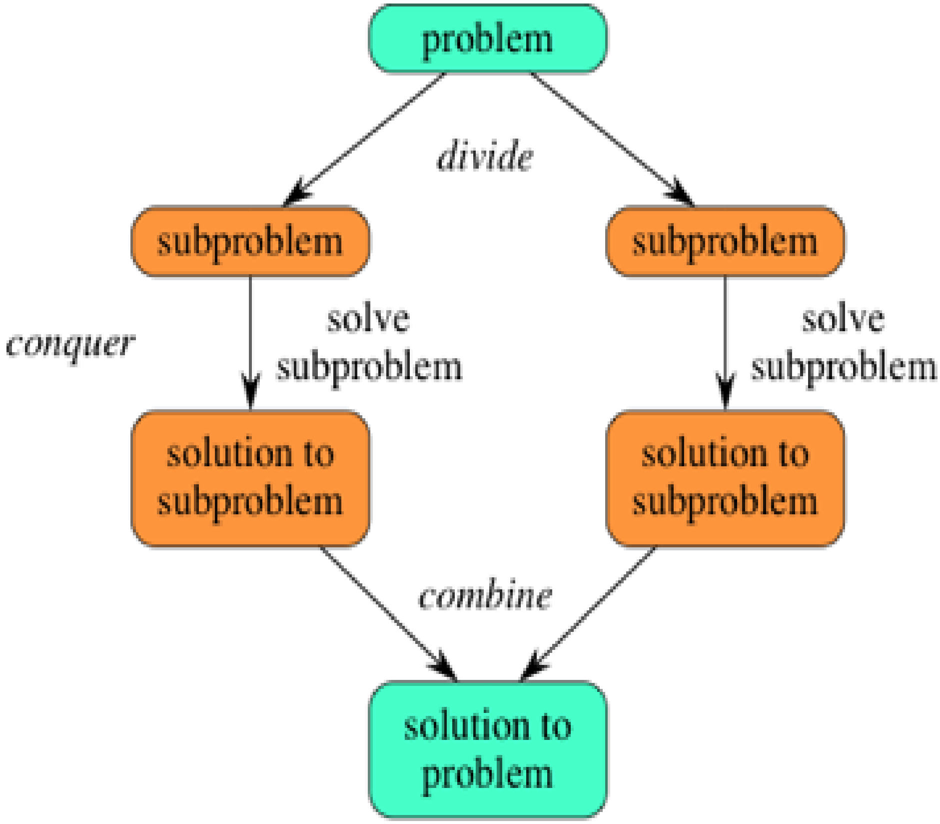
Assignment 1

- Design an algorithm using brute force approach to compute 2^n.

Divide and conquer

Divide and conquer strategy involves three steps :

1. **Divide** the given problem into sub-problems of same type. This step involves breaking the problem into smaller sub-problems. Sub-problems should represent a part of the original problem. This step generally takes a recursive approach to divide the problem until no sub-problem is further divisible.

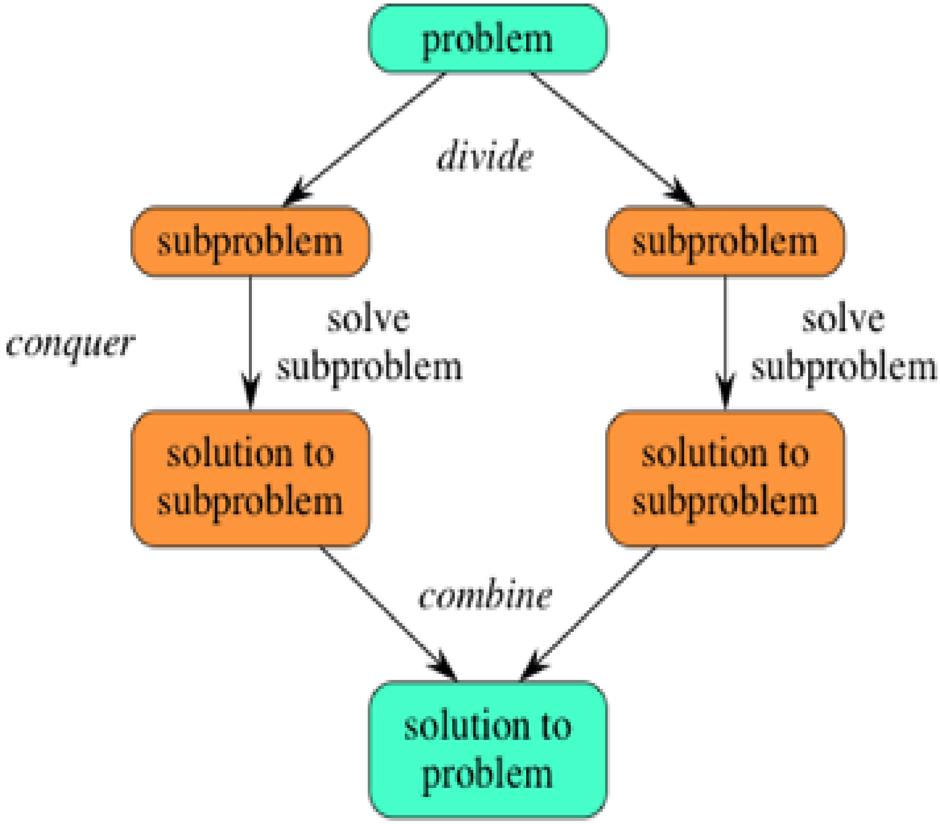


Divide and conquer

Divide and conquer strategy involves three steps :

2. Conquer the sub-problems by solving them recursively. If the sub-problem sizes are small enough, just solve the sub-problems in a straightforward manner.

3. Combine: Appropriately combine the answers. When the smaller sub-problems are solved, this stage recursively combines them until they formulate a solution of the original problem.

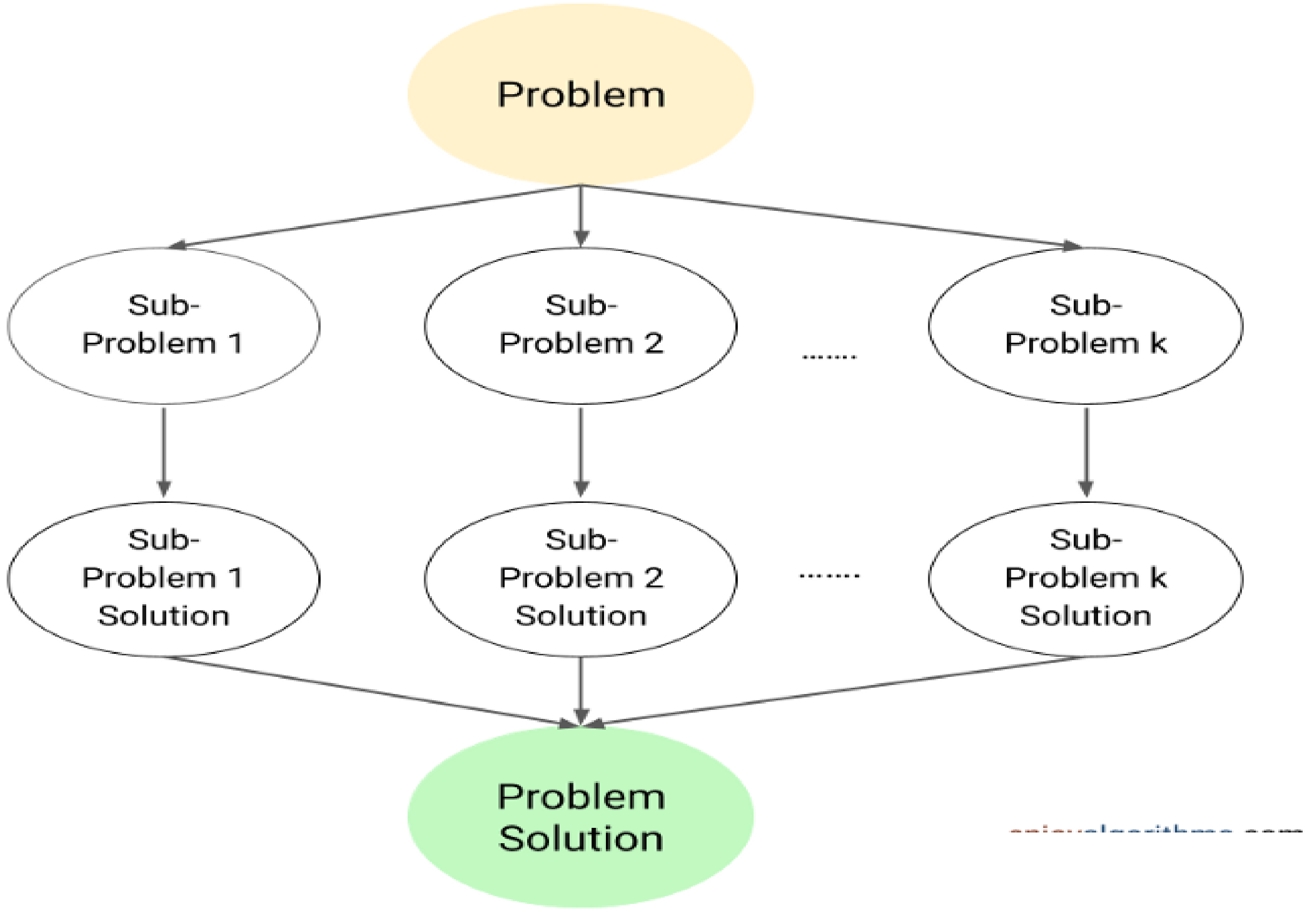


Divide and conquer

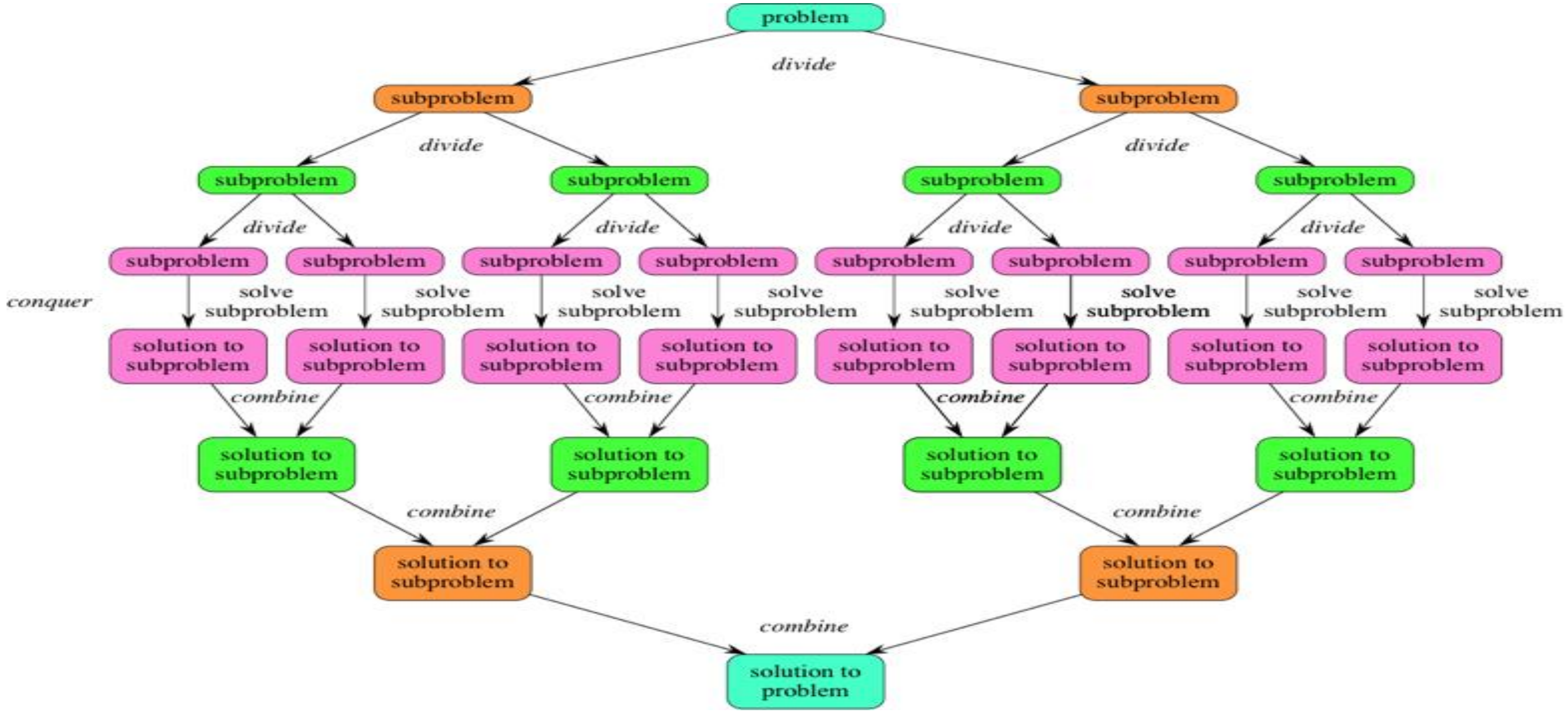
Divide
Dividing the problem into smaller sub-problems

Conquer
Solving each sub-problems recursively

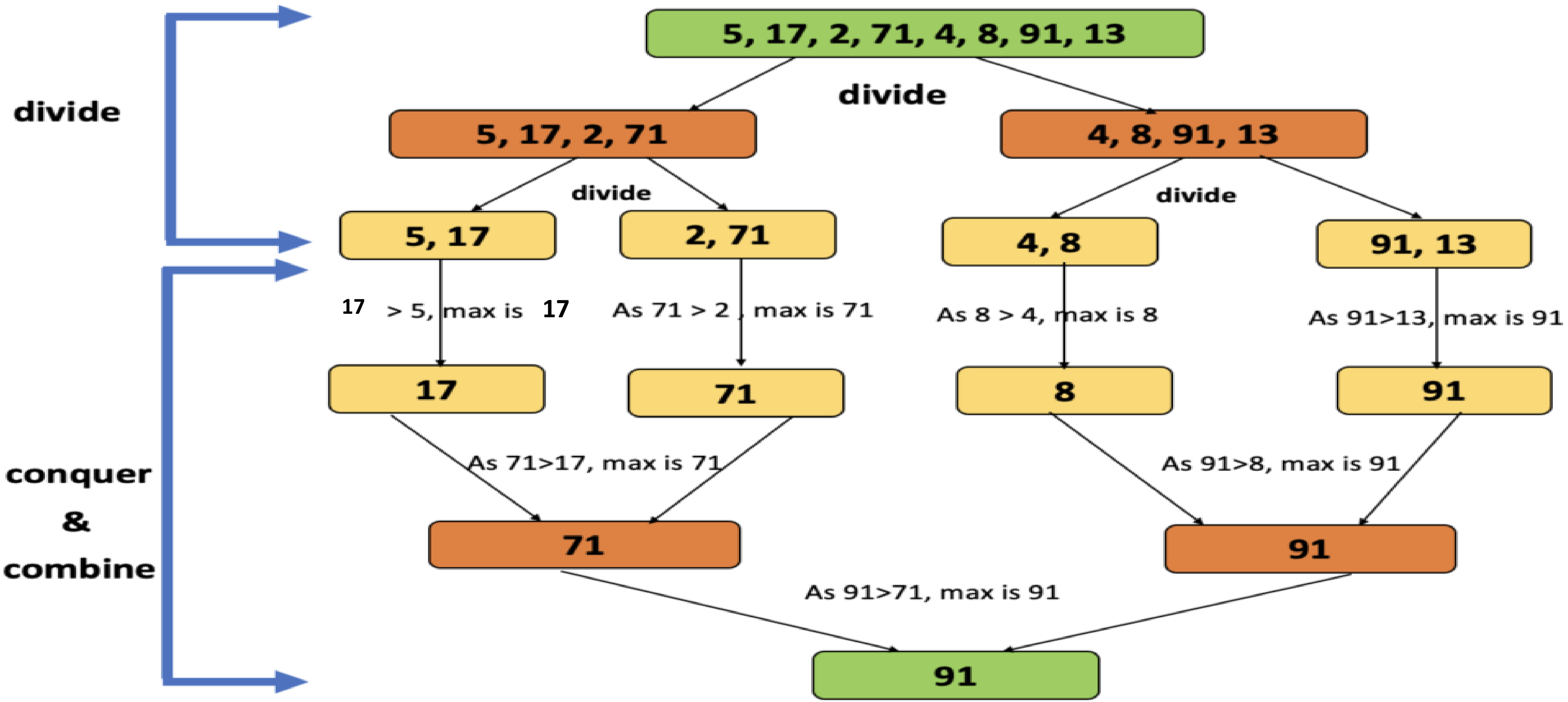
Combine
Combining sub-problem solutions to build the original problem solution



Divide and conquer

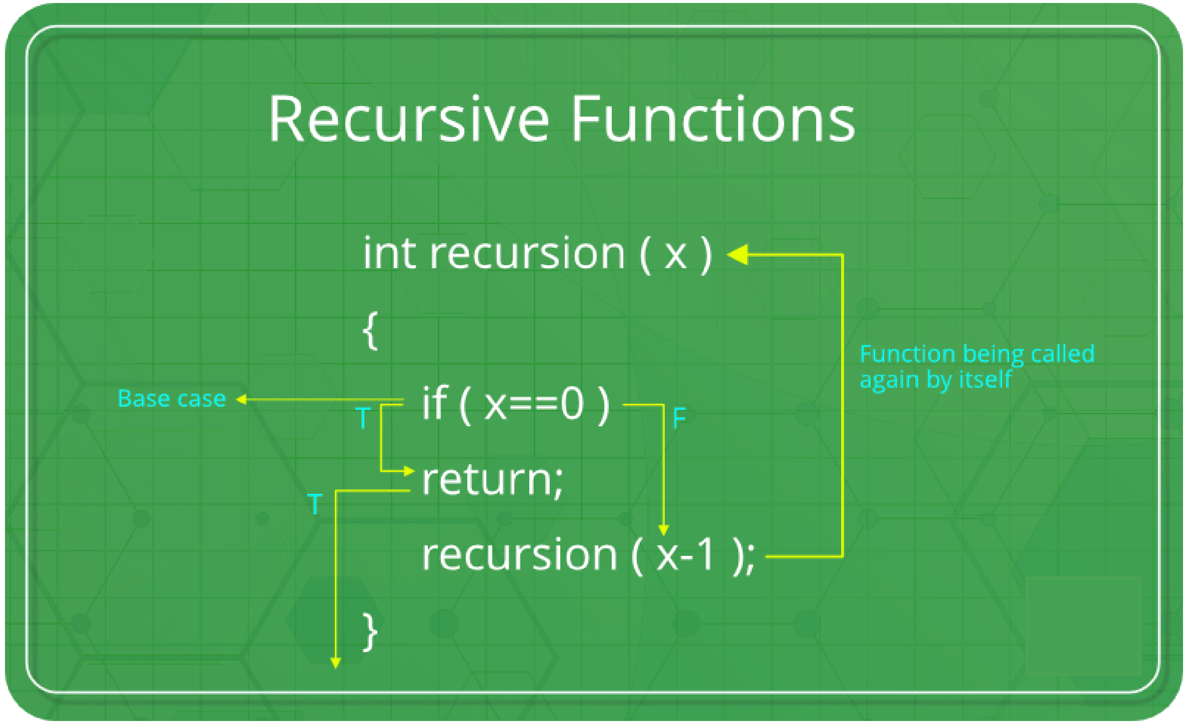


Divide and conquer (Find The Maximum)



Recursive Functions

- A recursive function is a function in code that refers to itself for execution.



Advantages of Divide and Conquer Algorithm

- ***Solving difficult problems***: It is a powerful method for solving difficult problems. Dividing the problem into sub-problems so that sub-problems can be combined again is a major difficulty in designing a new algorithm. For many such problem this algorithm provides a simple solution.
- The **Tower of Hanoi** was one of the biggest mathematical puzzles. But the divide and conquer algorithm has successfully been able to solve it recursively.
- The divide and conquer divides the problem into sub-problems which can run parallel at the same time. Thus, this algorithm works on **parallelism**. *Parallelism* allows us to solve the sub-problems independently, this allows for execution in multi-processor machines.
- ***Memory access***: It naturally tend to make efficient use of memory caches. This is because once a sub-problem is small, it and all its all its sub-problems can be solved within the cache, without accessing the slower main memory. The divide and conquer strategy makes use of cache memory because of the repeated use of variables in recursion. Executing problems in the cache memory is faster than the main memory.

Disadvantages of Divide and Conquer Algorithm

- The divide and conquer technique uses recursion. Recursion in turn leads to lots of space complexity because it makes use of the stack.

- The implementation of divide and conquer requires high memory management.

- The system may crash in case the recursion is not performed properly.

Assignment 2 (two weeks)

- Write a divide-and-conquer algorithm for the Tower of Hanoi problem

Some standard algorithms that follow Divide and Conquer algorithm

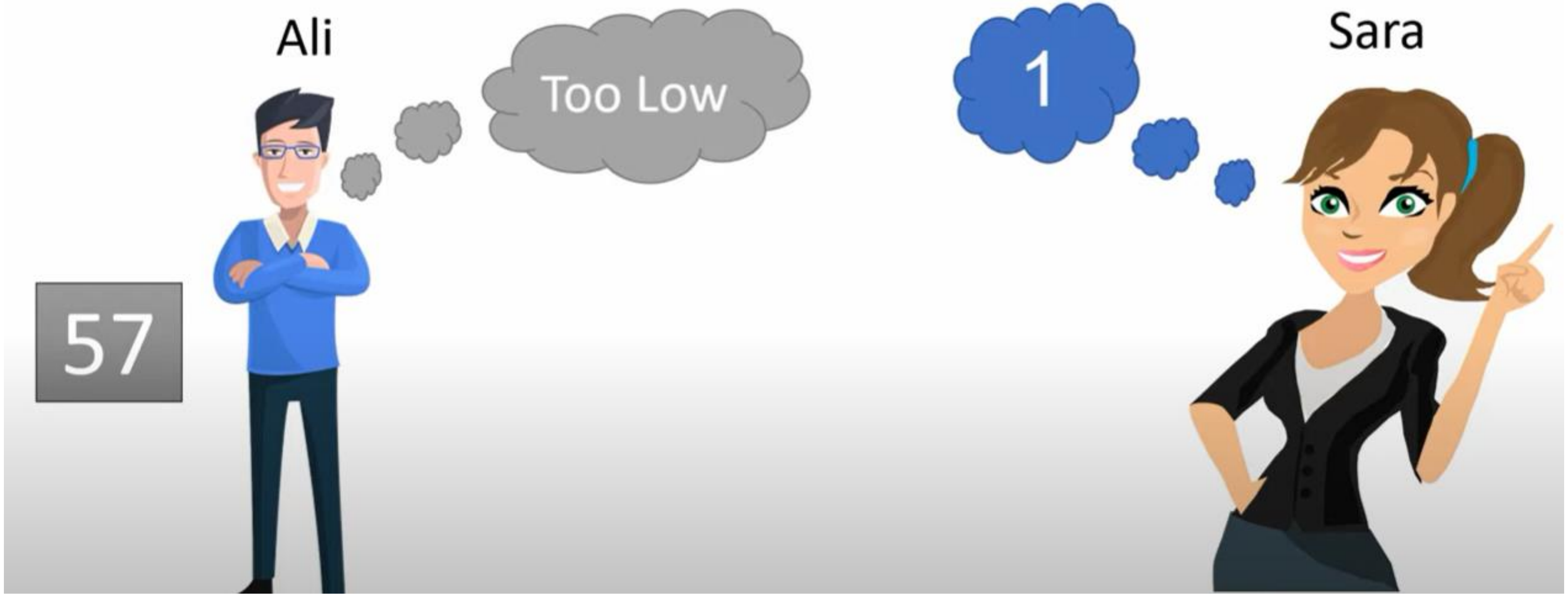
- $\boxed{\text{Binary Search}}$
- Merge Sort
- Quick Sort
- Closest Pair of Points
- Strassen’s Algorithm (matrix multiplication)
- Finding maximum and minimum

Guess the number from 0 to 100 [Traditional Search]

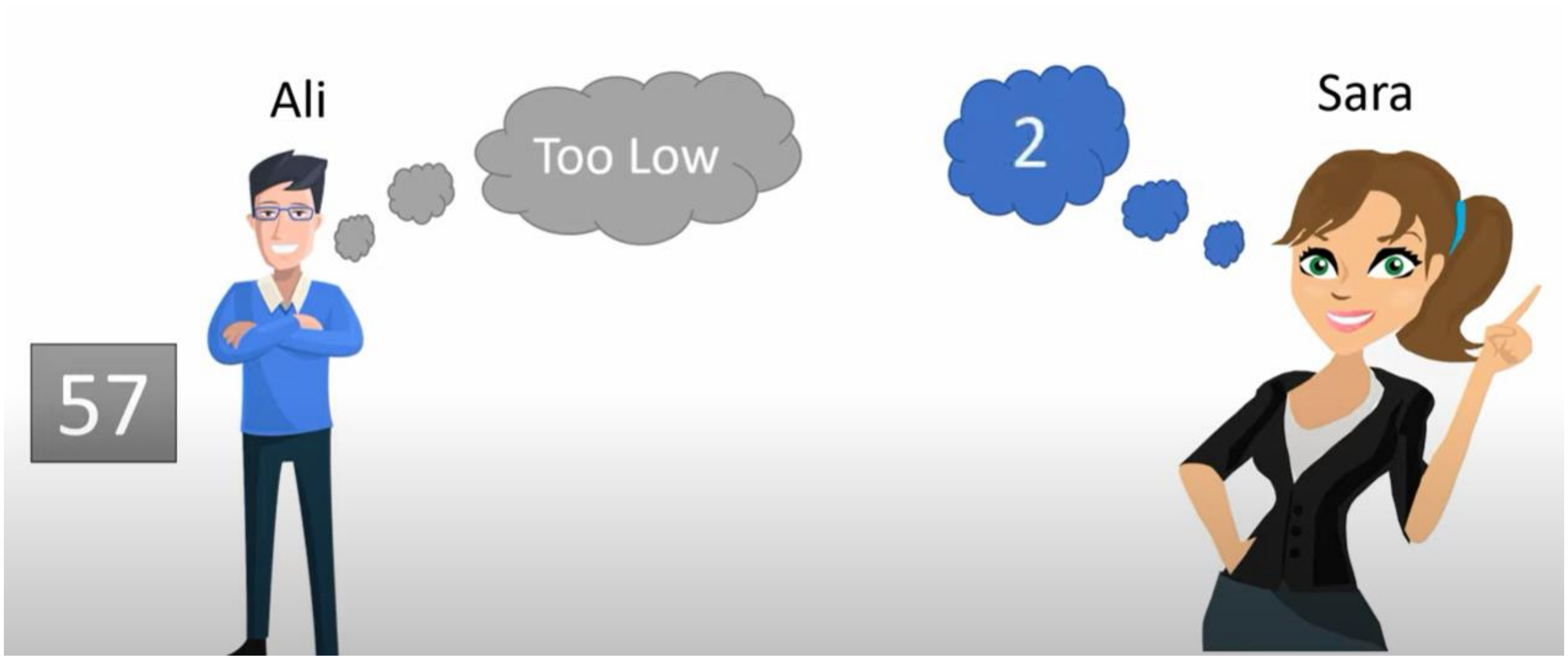
Guess the number [Traditional Search]

Guess the number [Traditional Search]

Guess the number [Traditional Search]

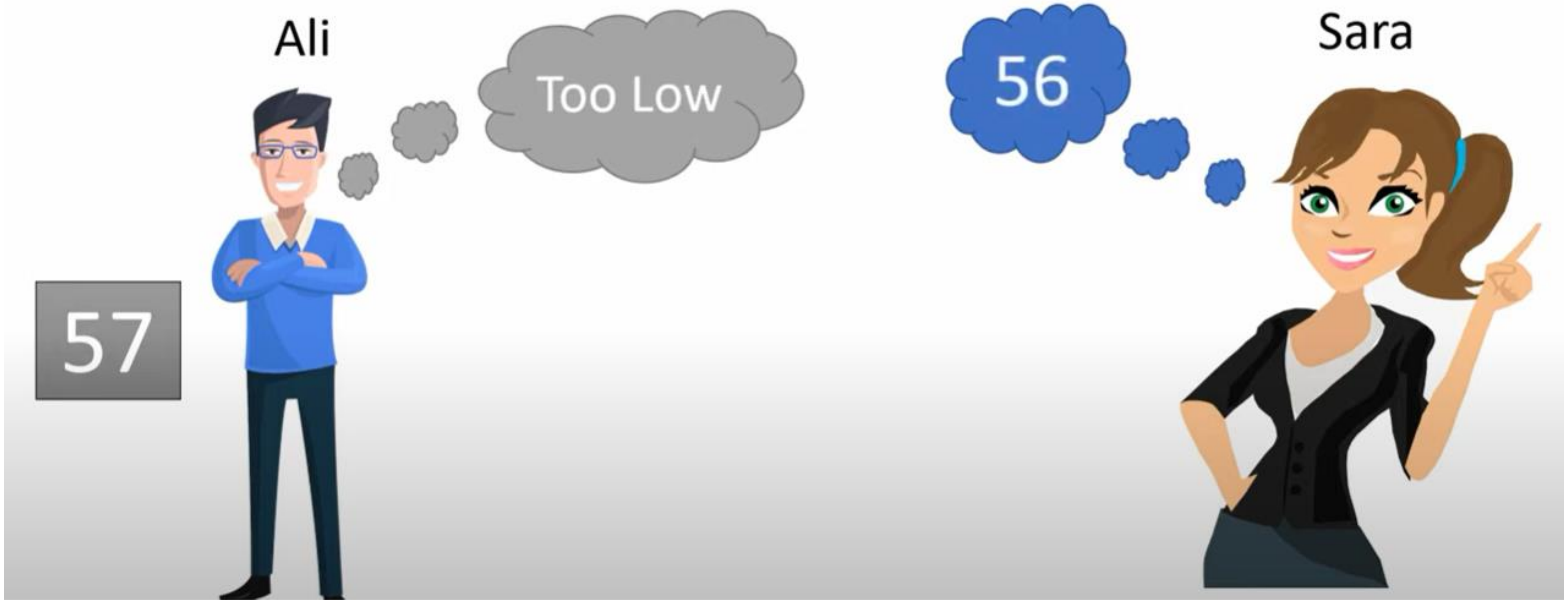


Guess the number [Traditional Search]



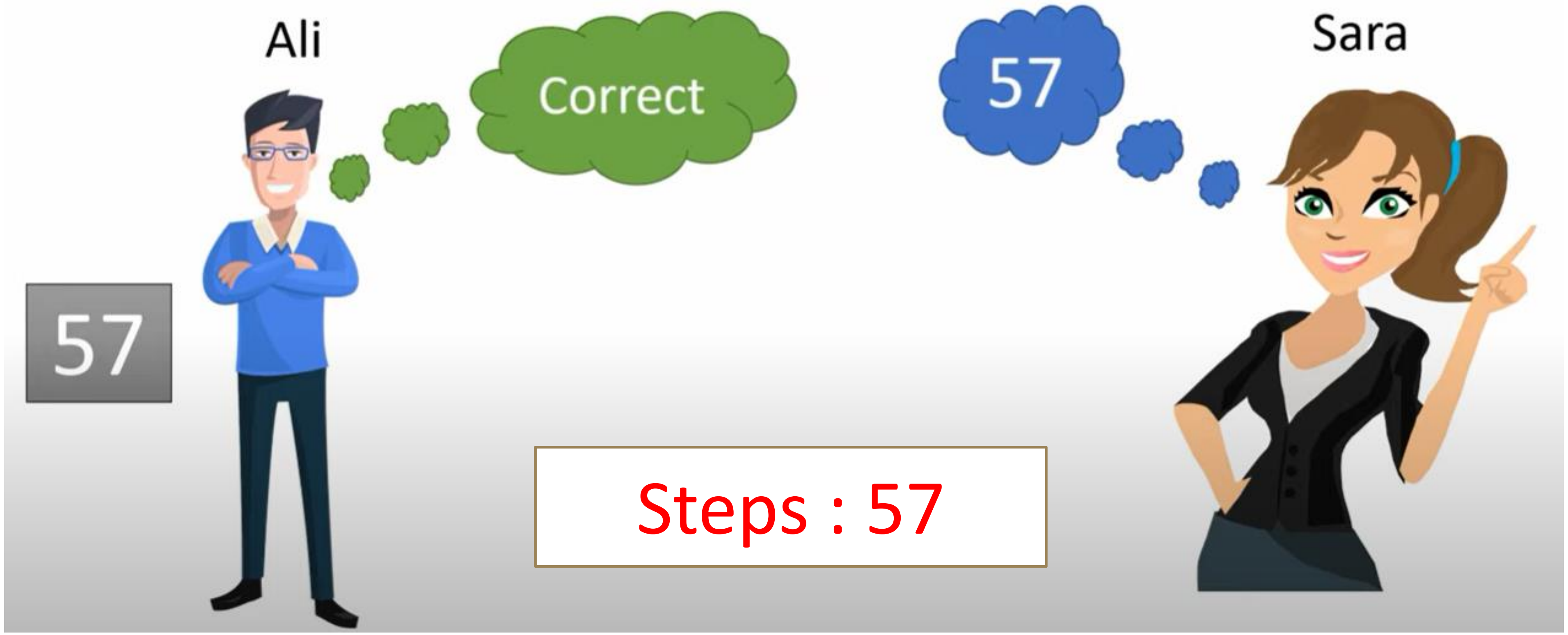
Guess the number [Traditional Search]

Guess the number [Traditional Search]



Guess the number [Traditional Search]

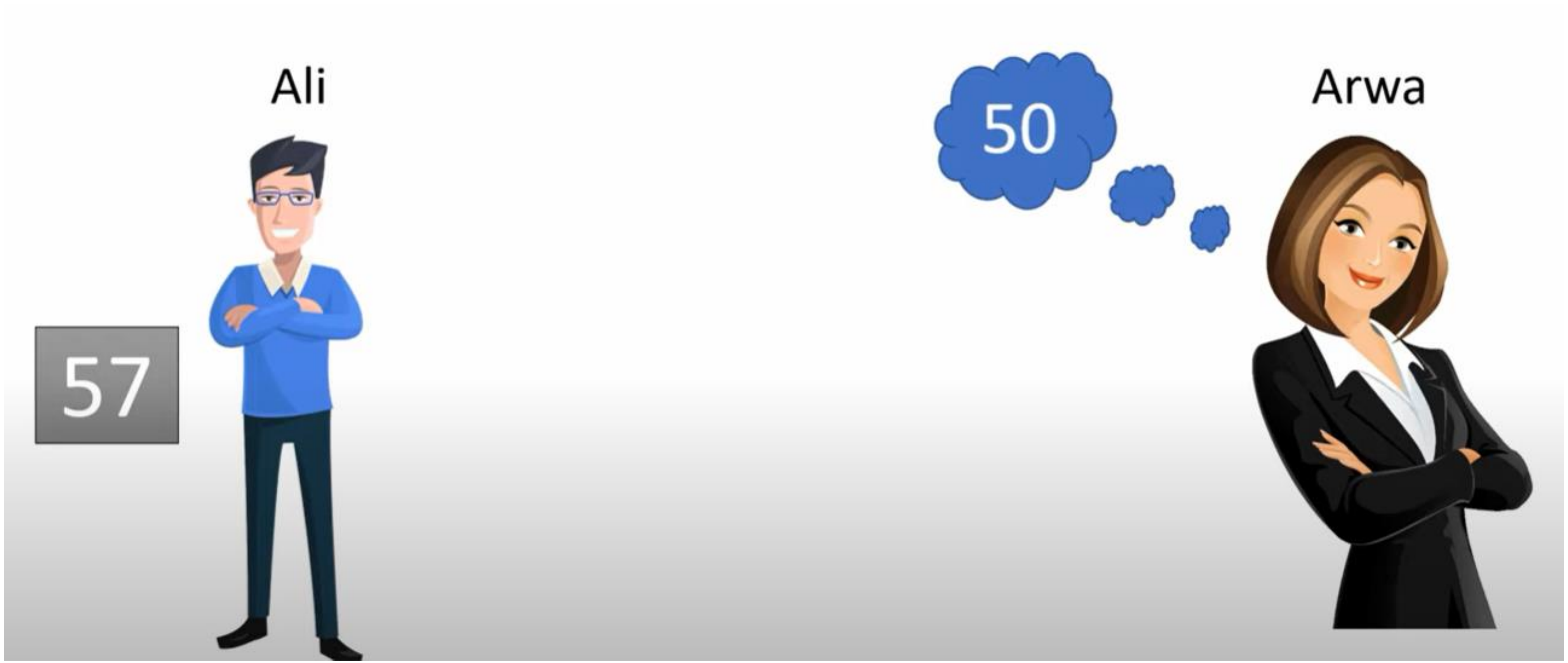
Sara follows the linear Search method



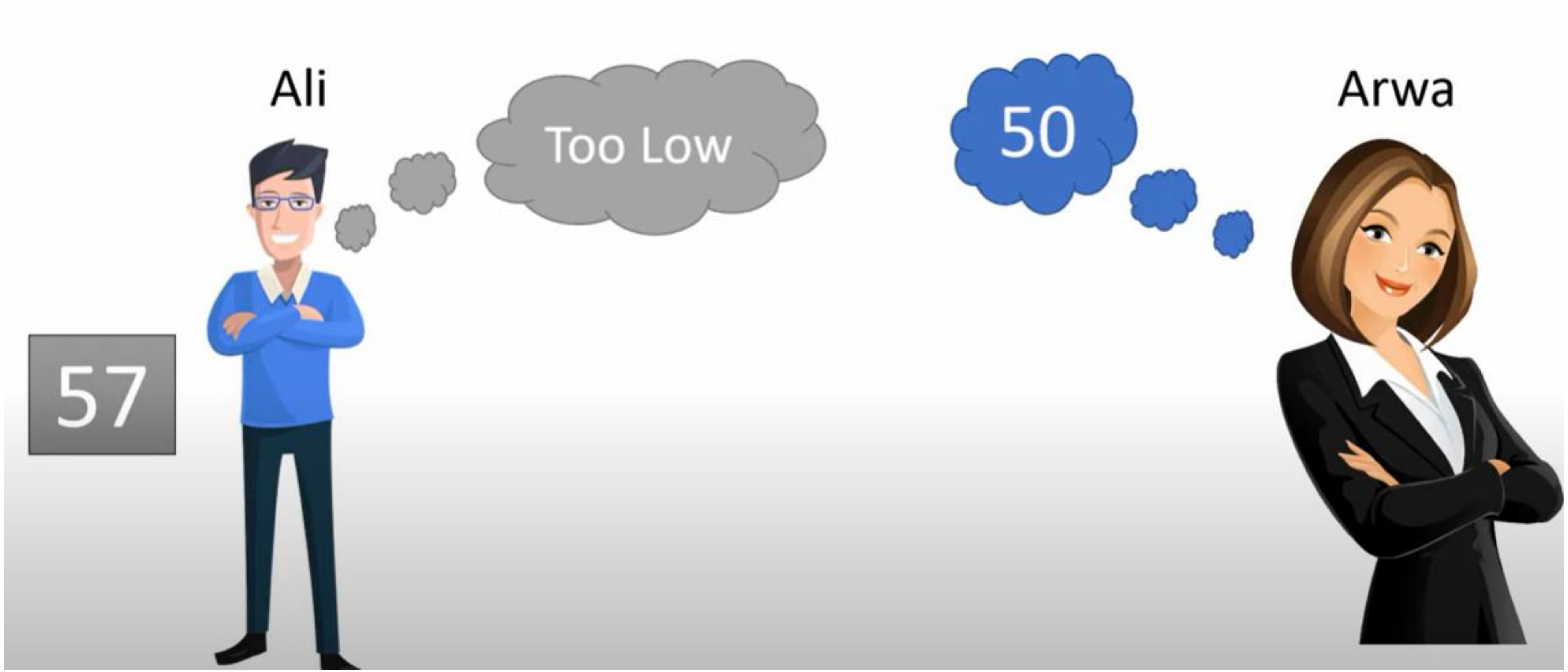
Guess the number from 0 to 100 [Binary Search]

Guess the number [Binary Search]

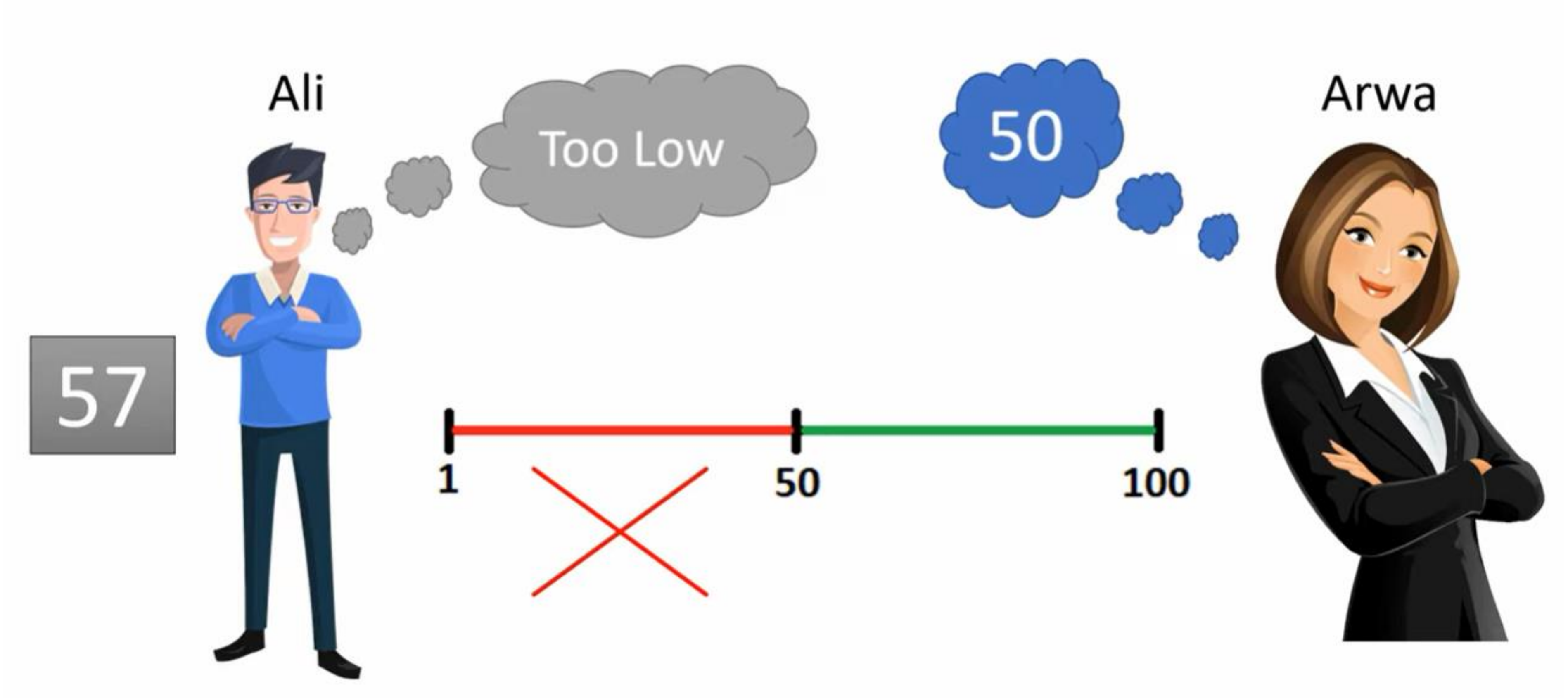
Guess the number [Binary Search]



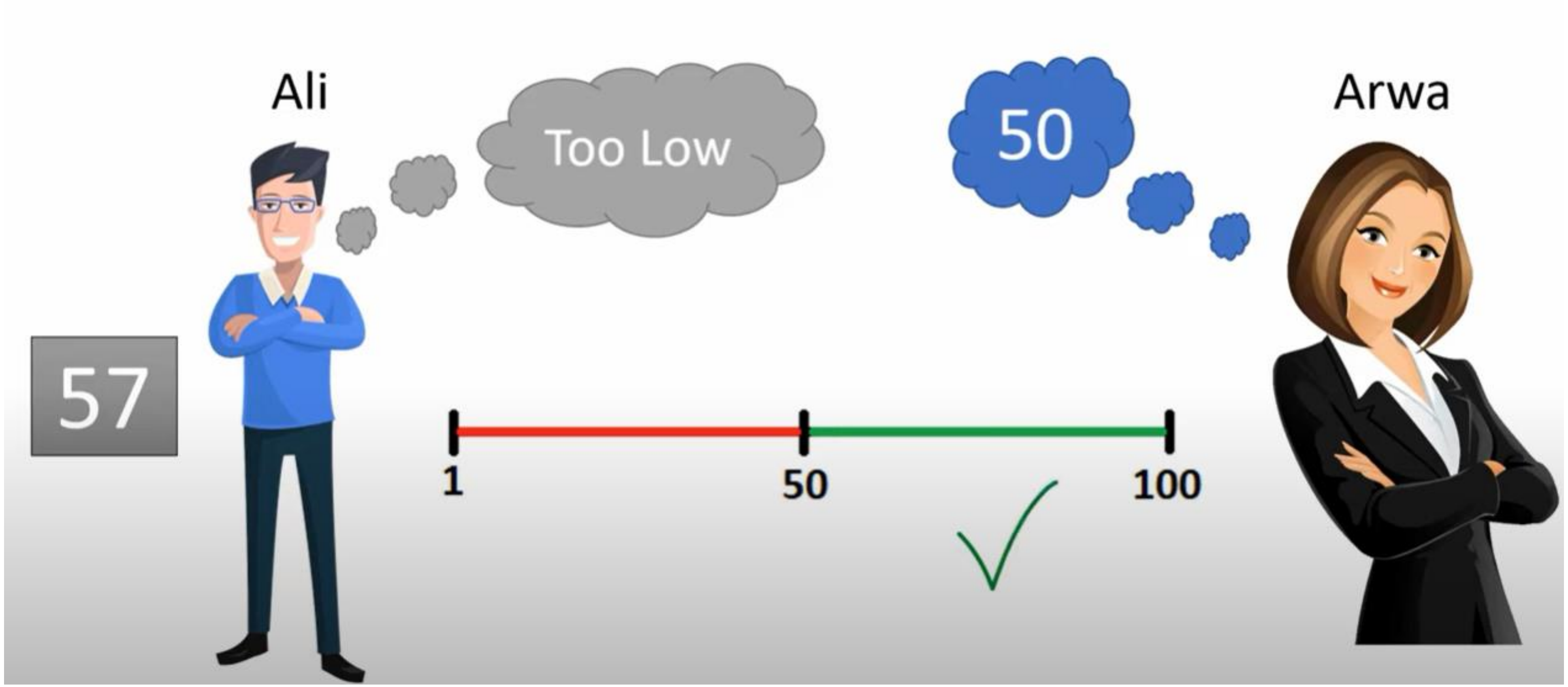
Guess the number [Binary Search]



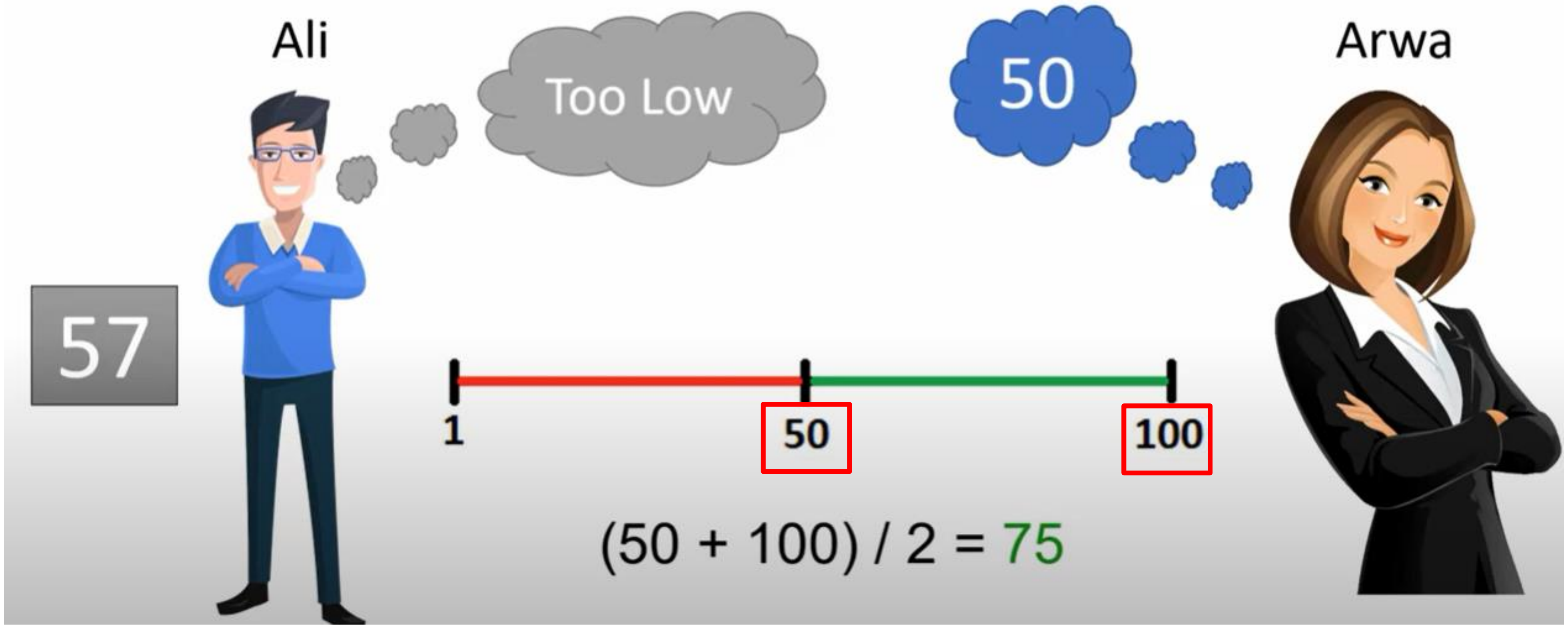
Guess the number [Binary Search]



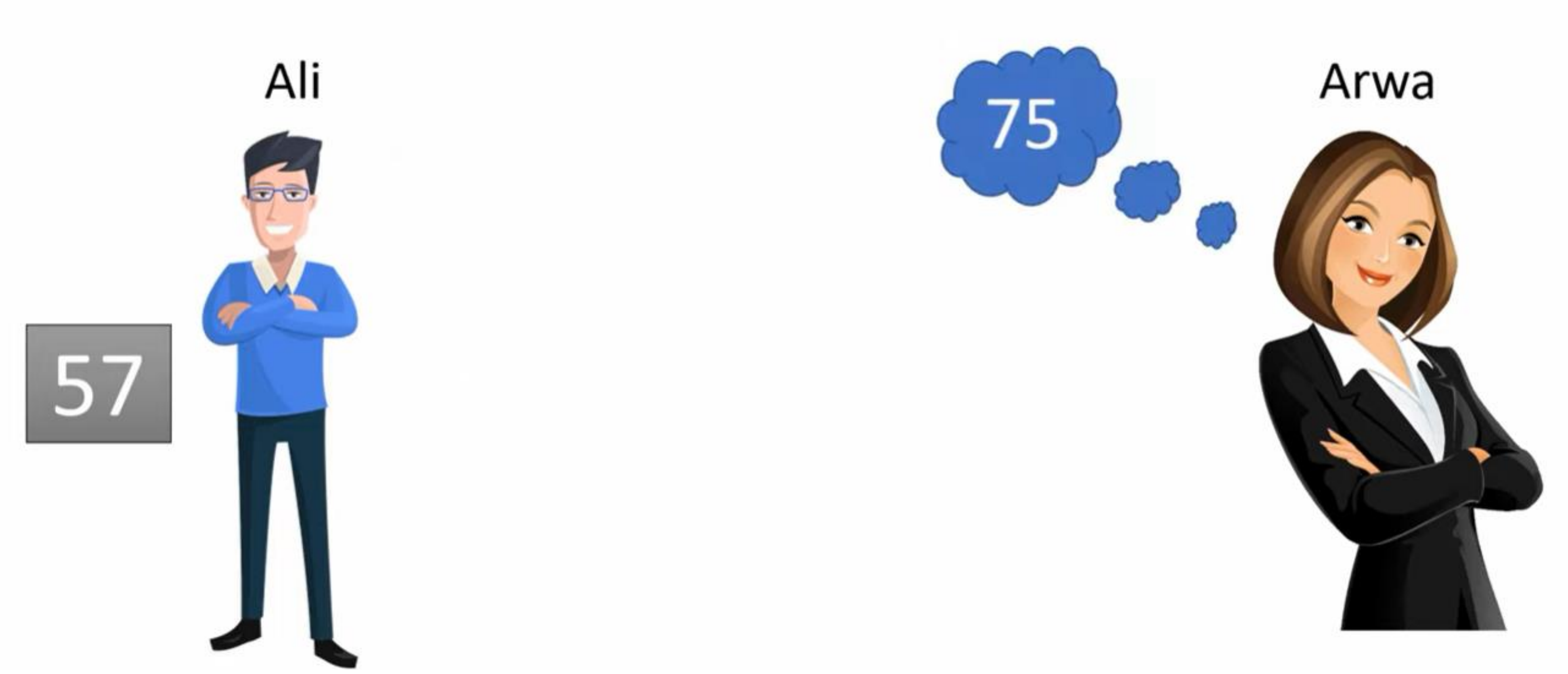
Guess the number [Binary Search]



Guess the number [Binary Search]

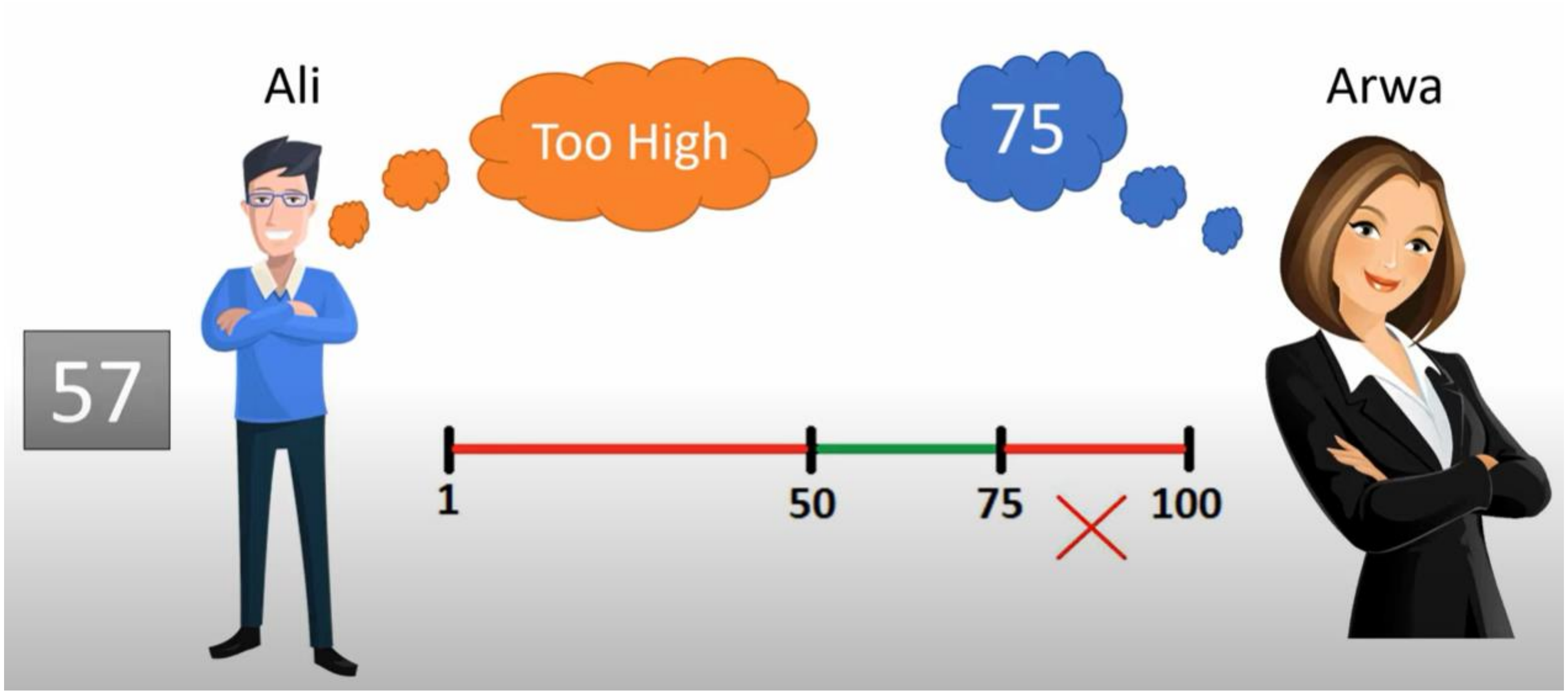


Guess the number [Binary Search]

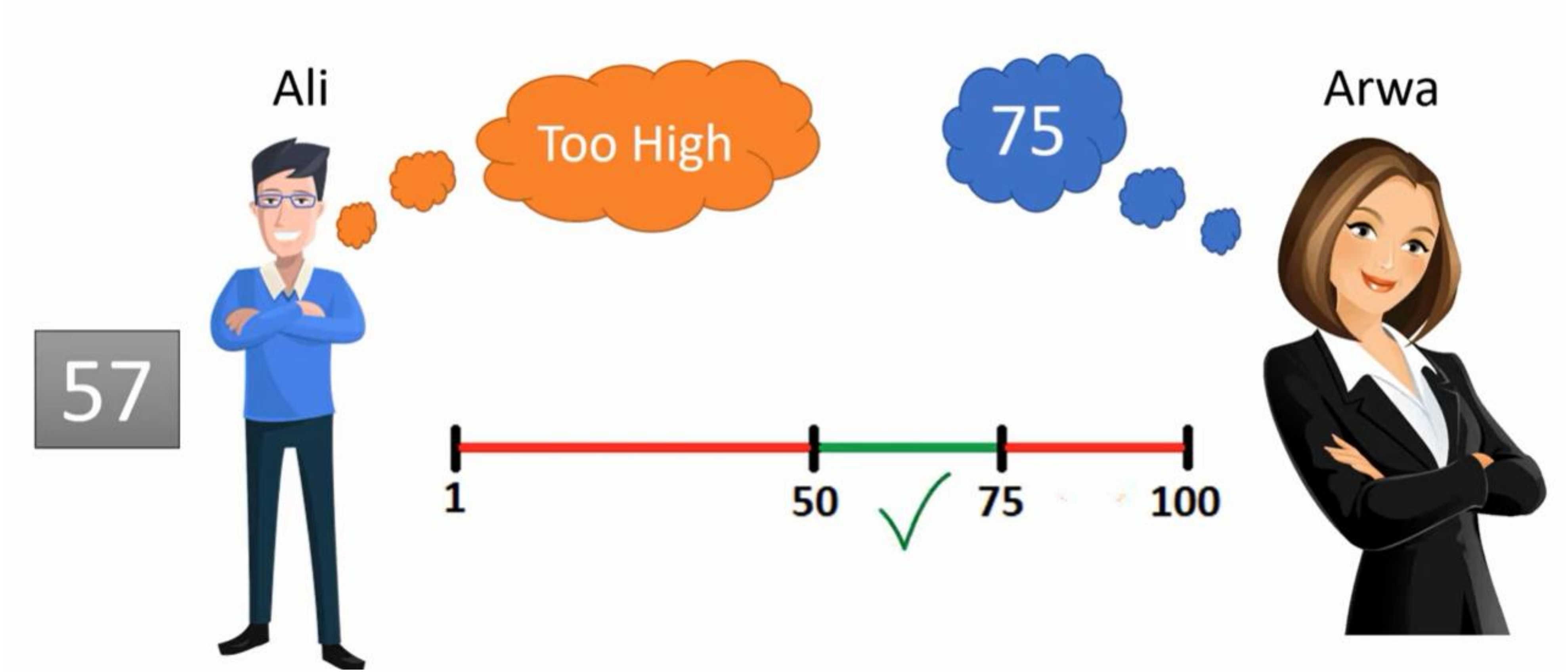


Guess the number [Binary Search]

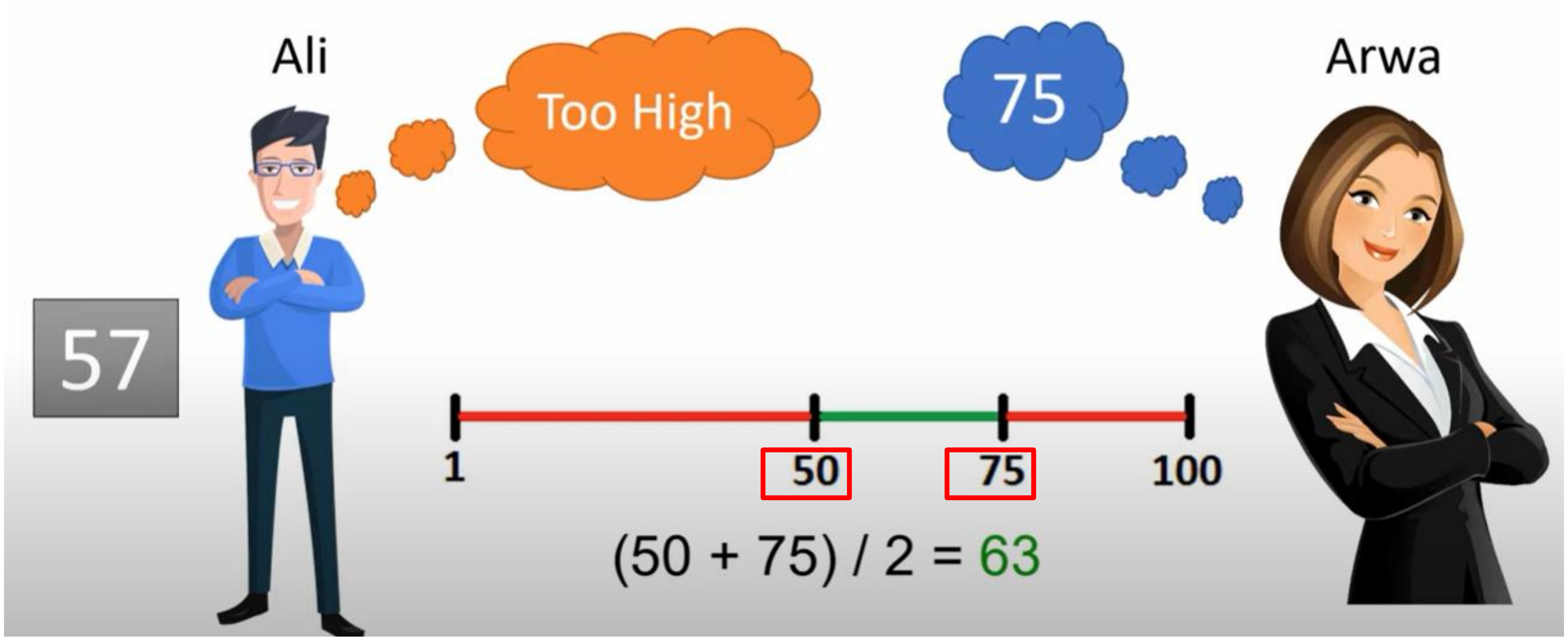
Guess the number [Binary Search]



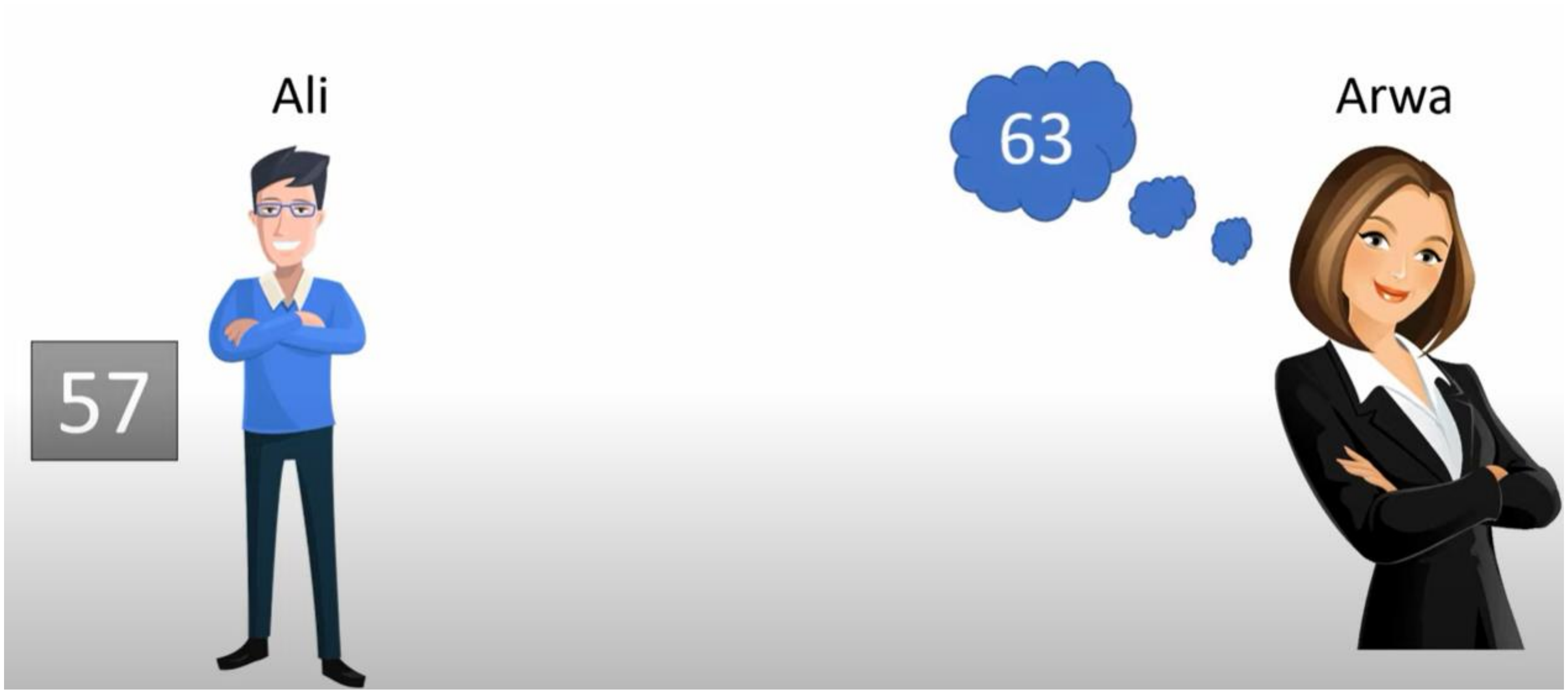
Guess the number [Binary Search]



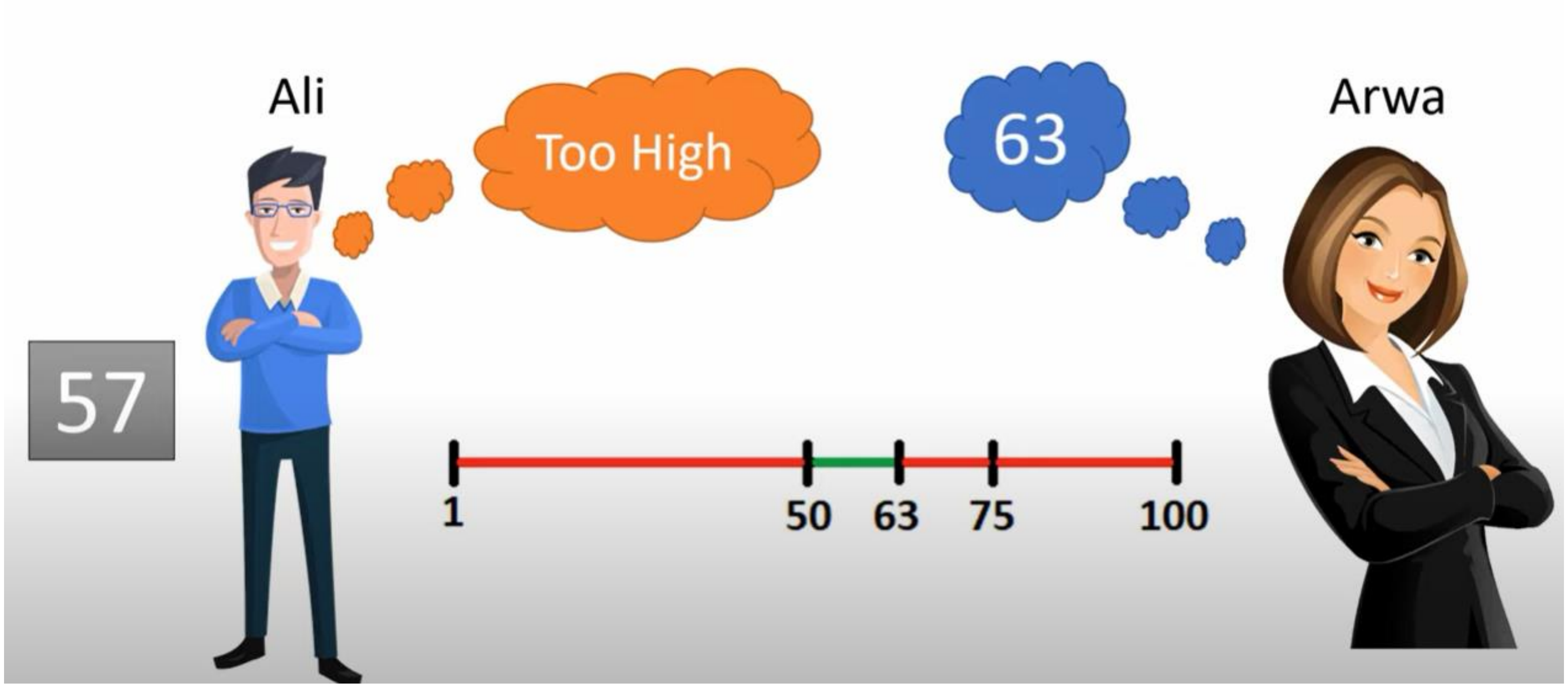
Guess the number [Binary Search]



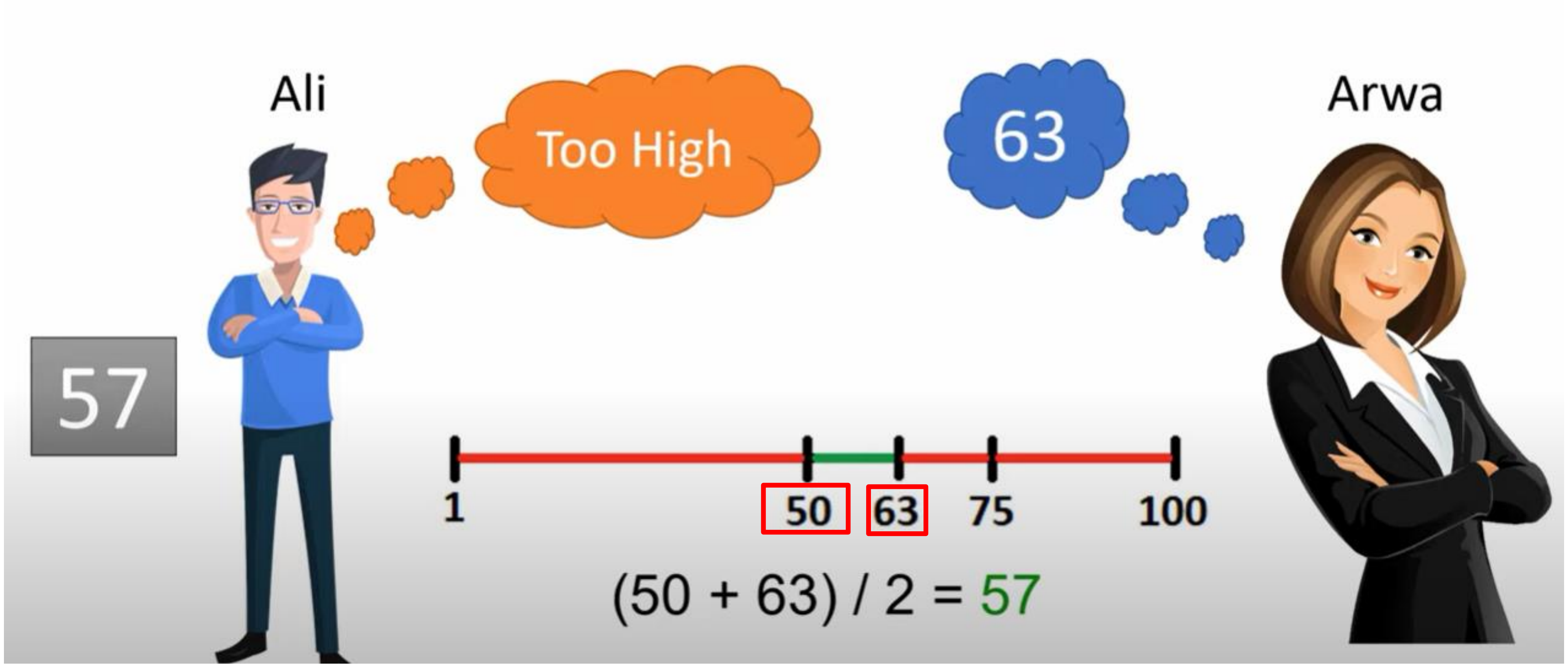
Guess the number [Binary Search]



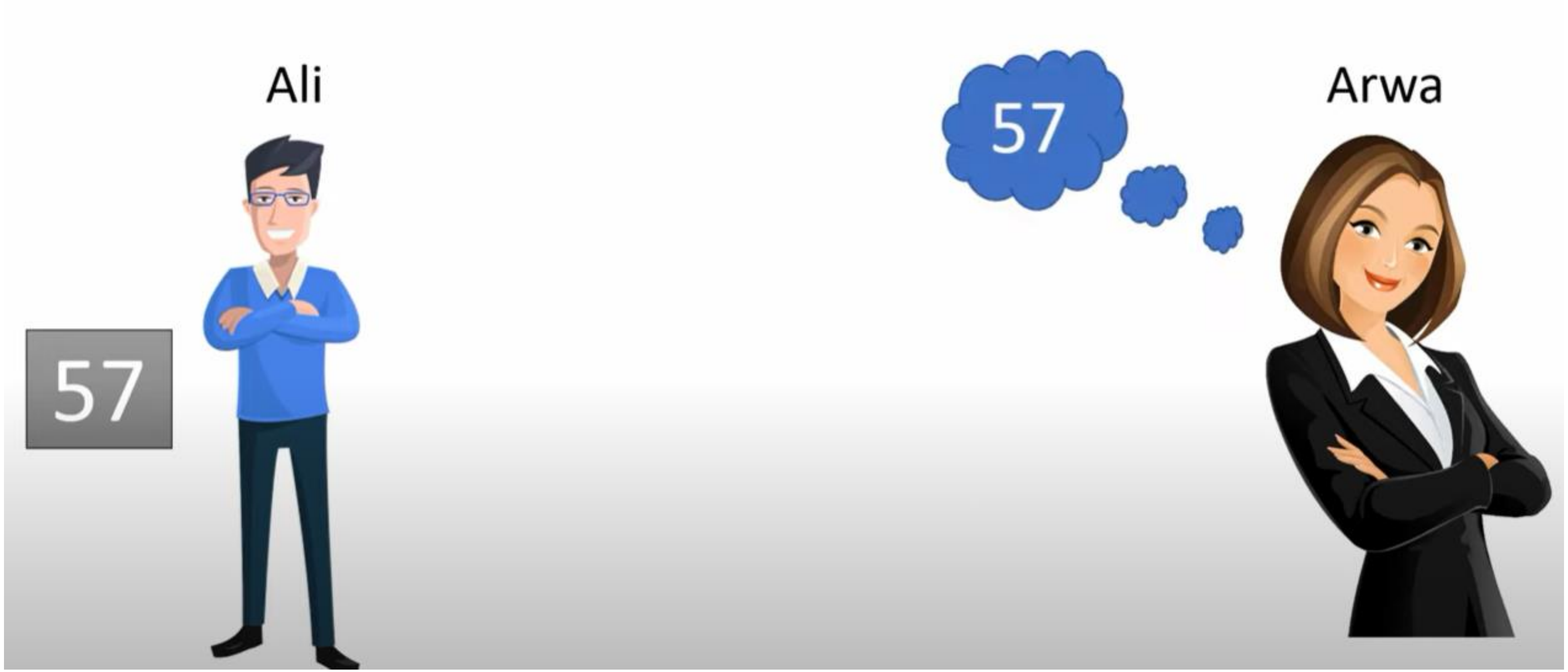
Guess the number [Binary Search]



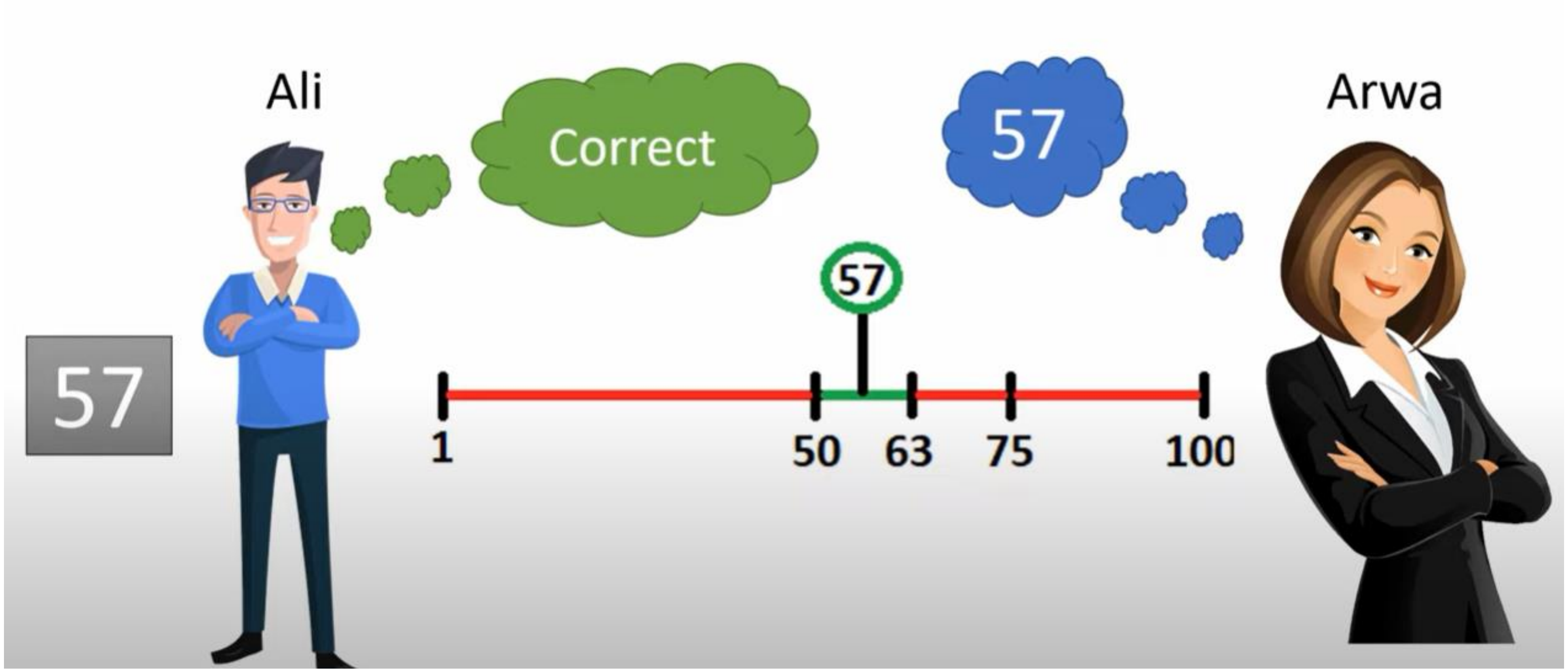
Guess the number [Binary Search]



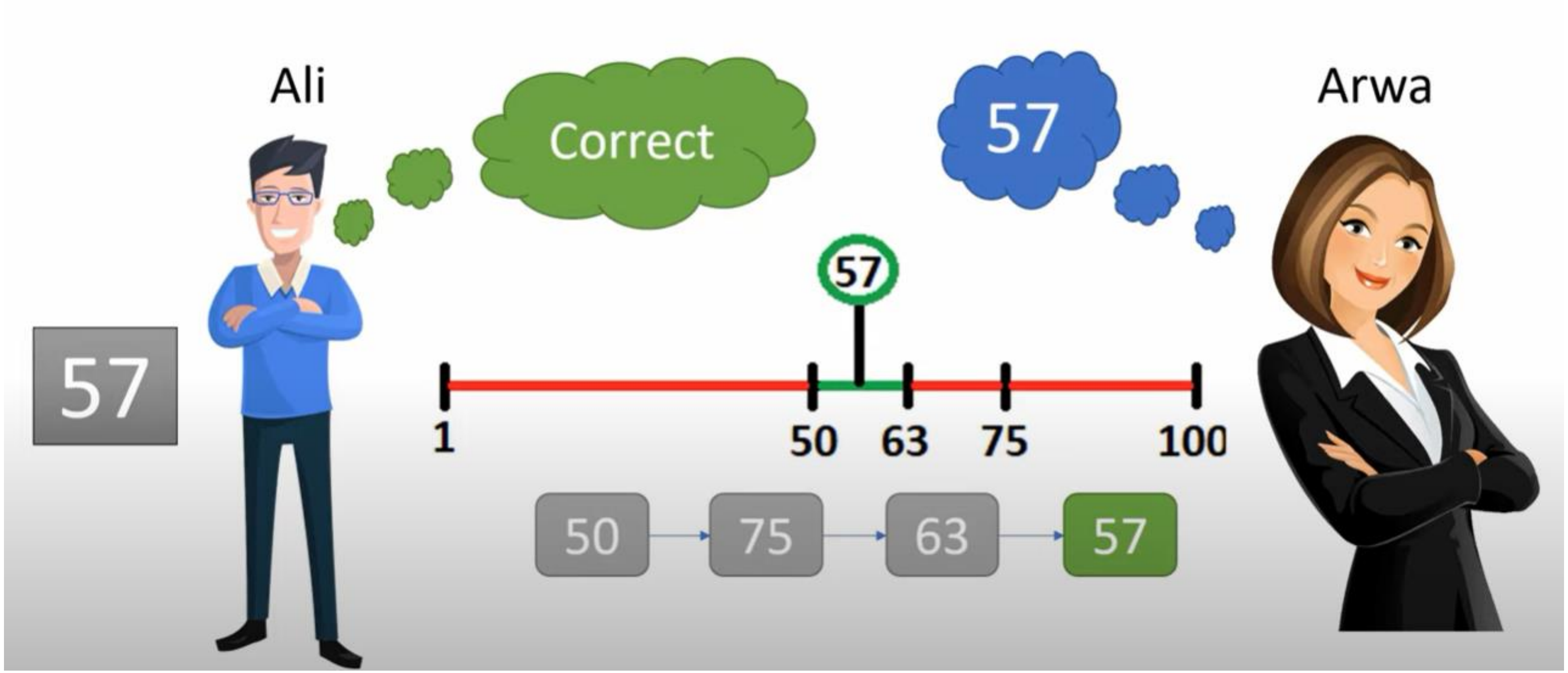
Guess the number [Binary Search]



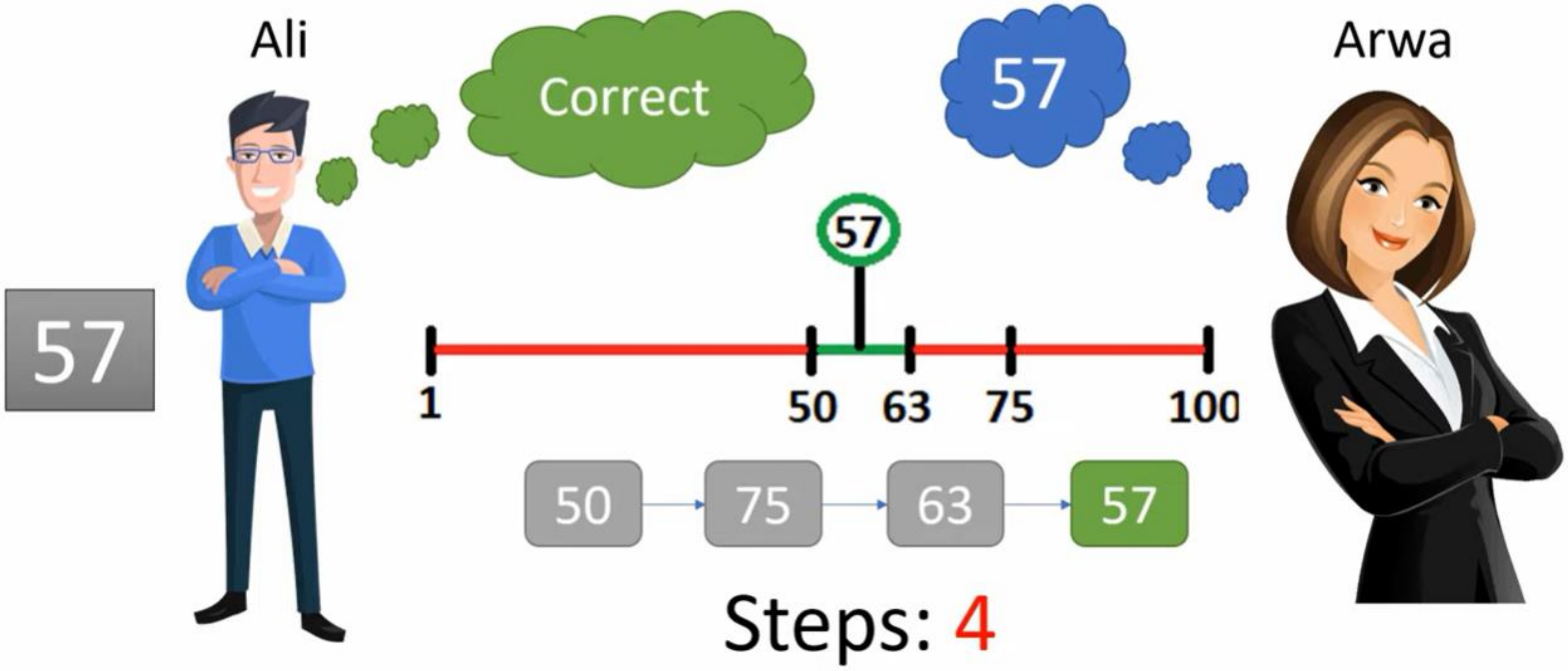
Guess the number [Binary Search]



Guess the number [Binary Search]



Guess the number [Binary Search]



Binary Search

- **Problem Definition:** Given a sorted array $A=(a_1, a_2,\dots, a_n)$ of n elements in non-decreasing order and an element k. Find the position of k in A, j, if $k=a_j$. Otherwise, return zero.

Examples

Example 1: Given A=(2,4,6,7,10,17,20) and k=7 then search(A,k)=4

Example 2: Given A=(2,4,6,7,10,17,20) and k=6 then search(A,k)=3

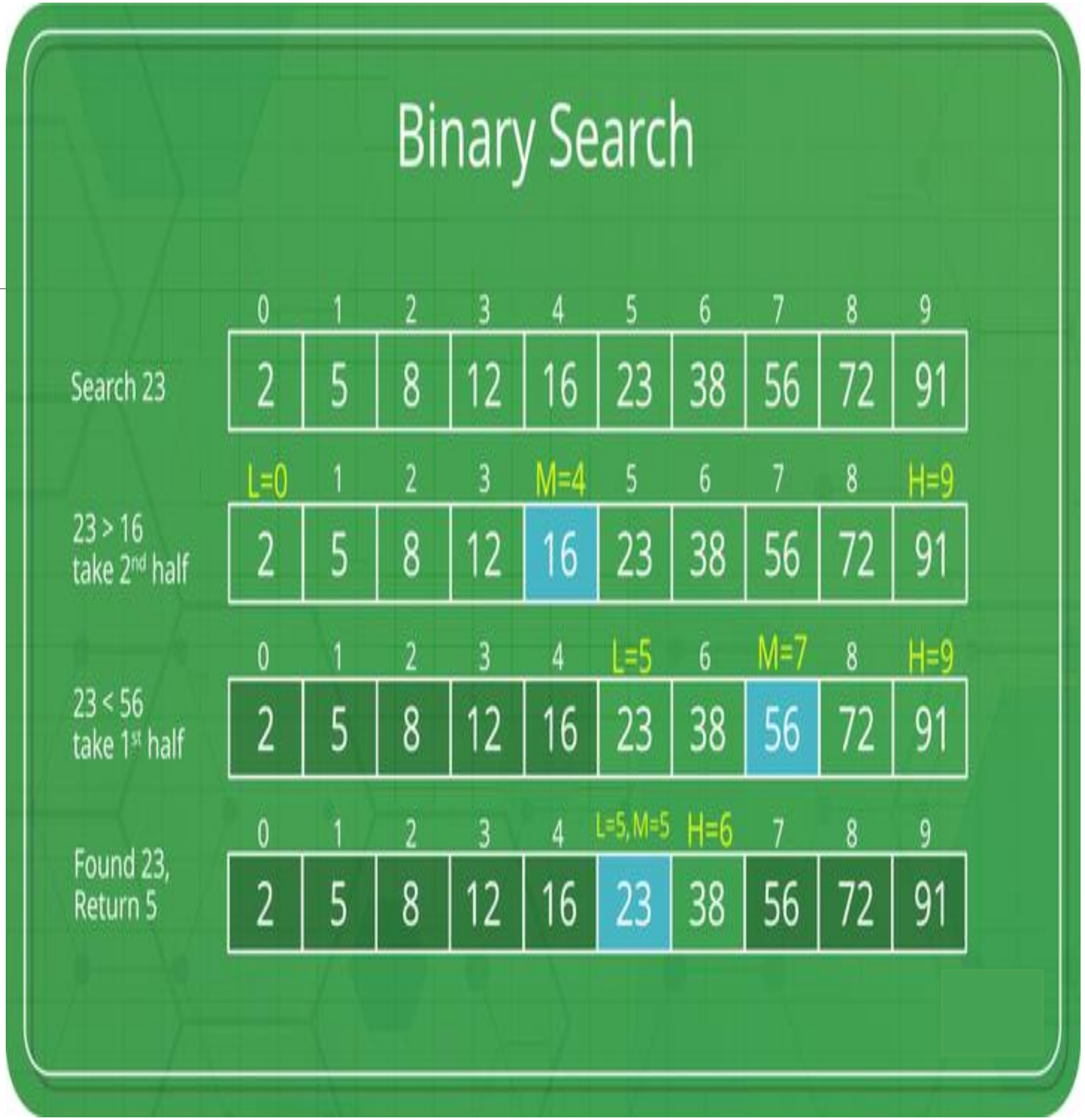
Example 3: Given A=(2,4,6,7,10,17,20) and k=9 then search(A,k)=0

Binary Search

Main Idea:

- We compare a given element k with the middle element in the sorted array $A(1...n)$.

- $m=[(L+H)/2]$, L is the index of first element of A and H is the index of the last element of A.
- If $k = a_m$ then the element k is exist in the array A and return m.
- If $k < a_m$ then we discard $A(m...H)$ and we repeat the same process on $A(L...m\text{-}1)$. Similarly,
- if $k > a_m$ then we discard $A(L...m)$ and we repeat the same process on $A(m+1..H)$.



Pseudo Code

Algorithm: BinarySearch(A(L...H),k)

Begin

if $L > H$ then return 0

else

$m = \lfloor (L+H)/2 \rfloor$

if $k = a_m$ then return m

else if $k < a_m$ then return BinarySearch($A(L.....m$-1),k)

else return BinarySearch($A(m$+1.....$H)$,k)

End.

Recursion

Time complexity of linear search and binary search

	Linear Search	Binary Search
Time Complexity	$O(n)$	$O(\log_2 n)$
$n = 10$	$10ms$	$3ms$
$n = 1000$	$1\ sec$	$10\ ms \approx 0.01\ sec$
$n = 10^6$	$16.6\ min$	$19\ ms \approx 0.02\ sec$
$n = 10^9$	$11\ day$	$30\ ms \approx 0.03\ sec$