Design and Analysis of Algorithms

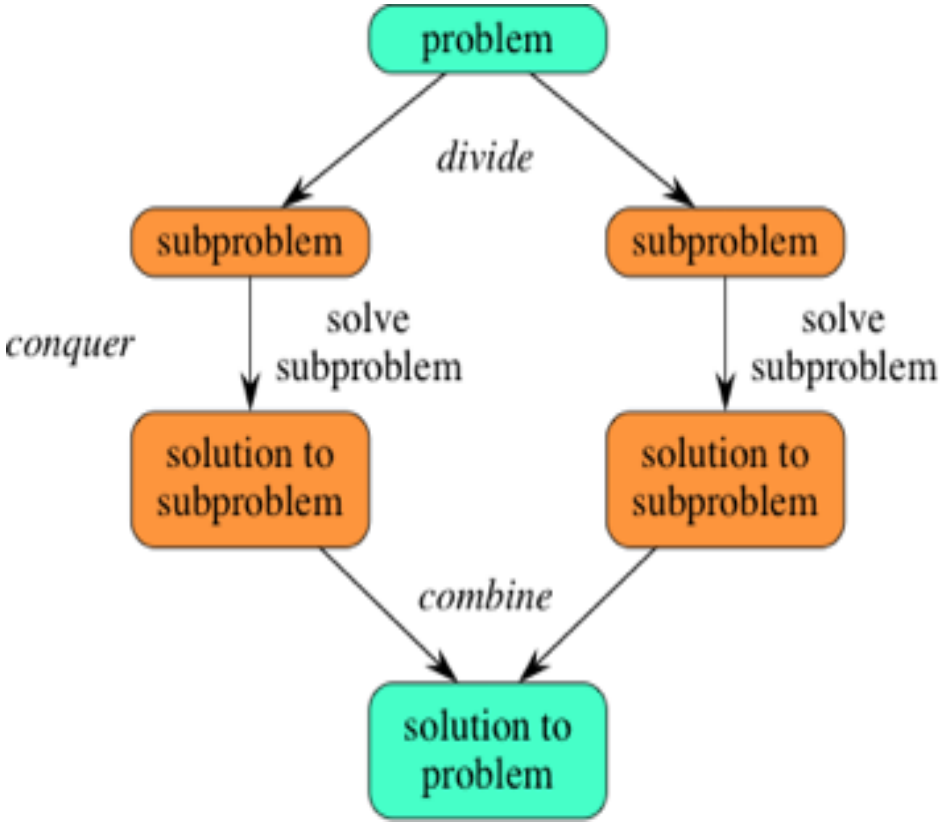
Dina El-Manakhly, Ph. D.

dina_almnakhly@science.suez.edu.eg

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.

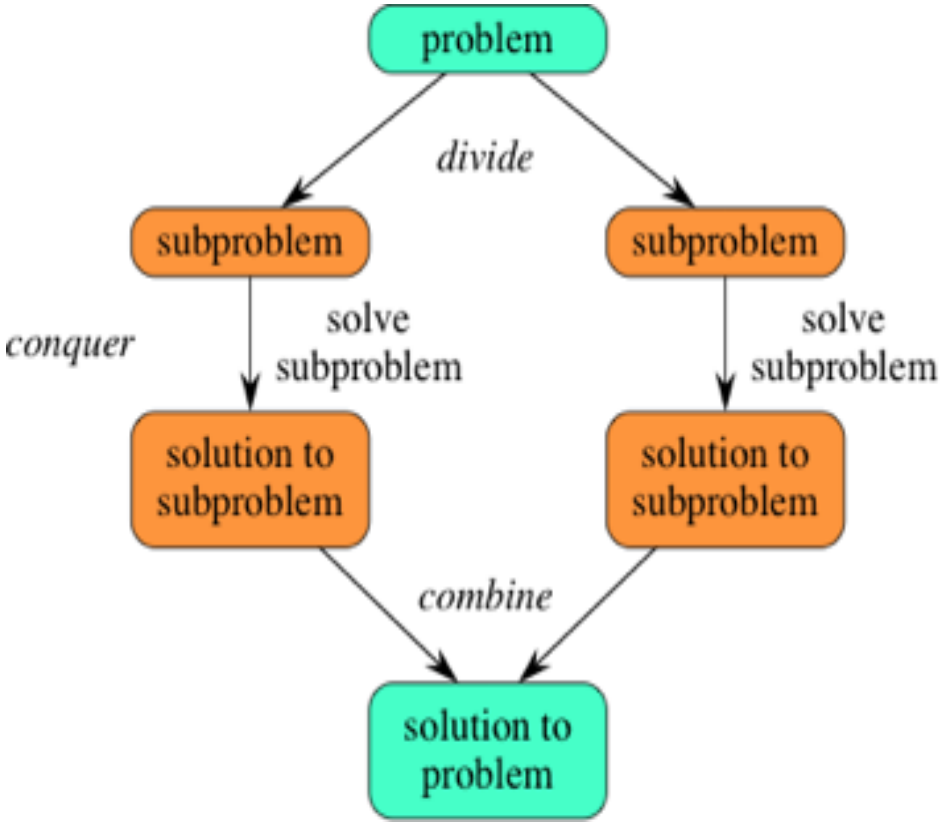


Divide and conquer

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.

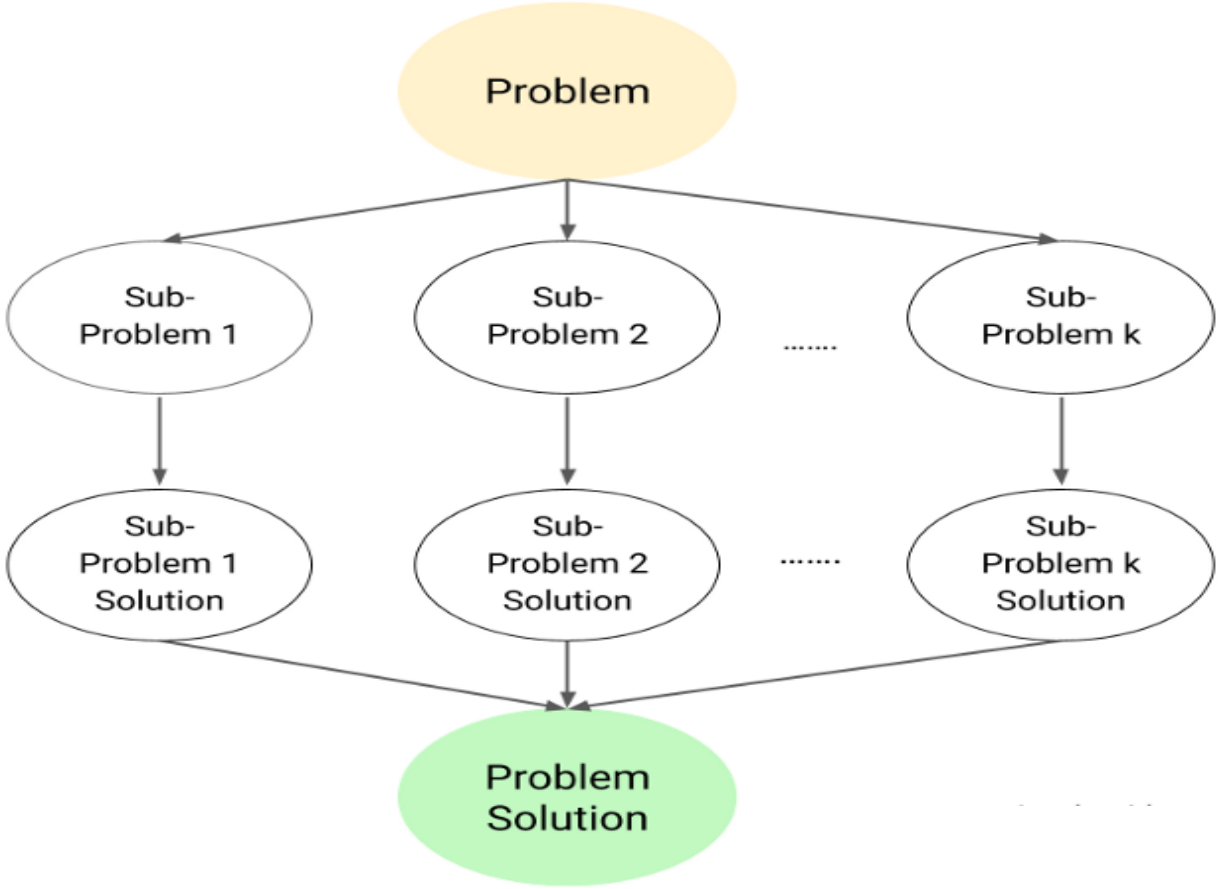


Divide and conquer

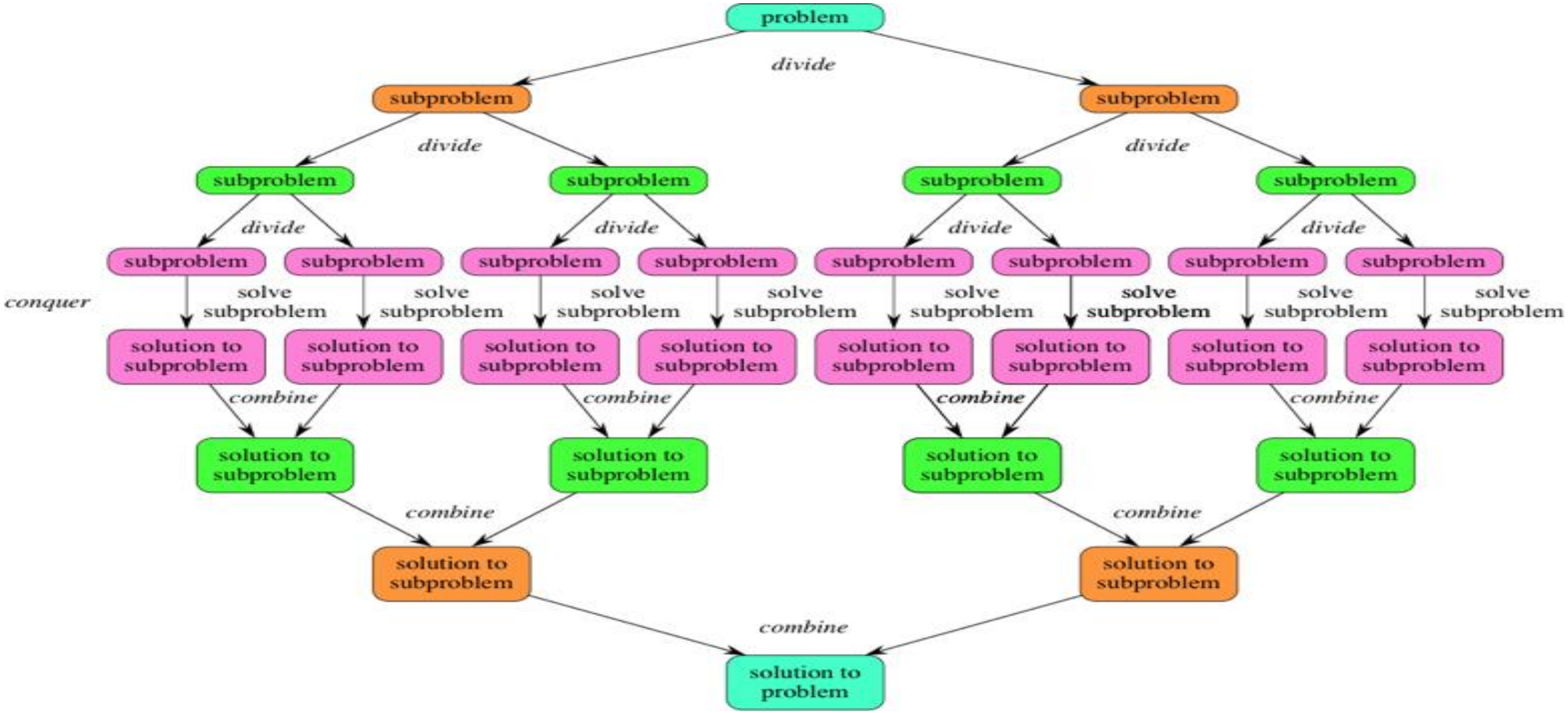
Divide
Dividing the problem into smaller sub-problems

Conquer
Solving each sub-problems recursively

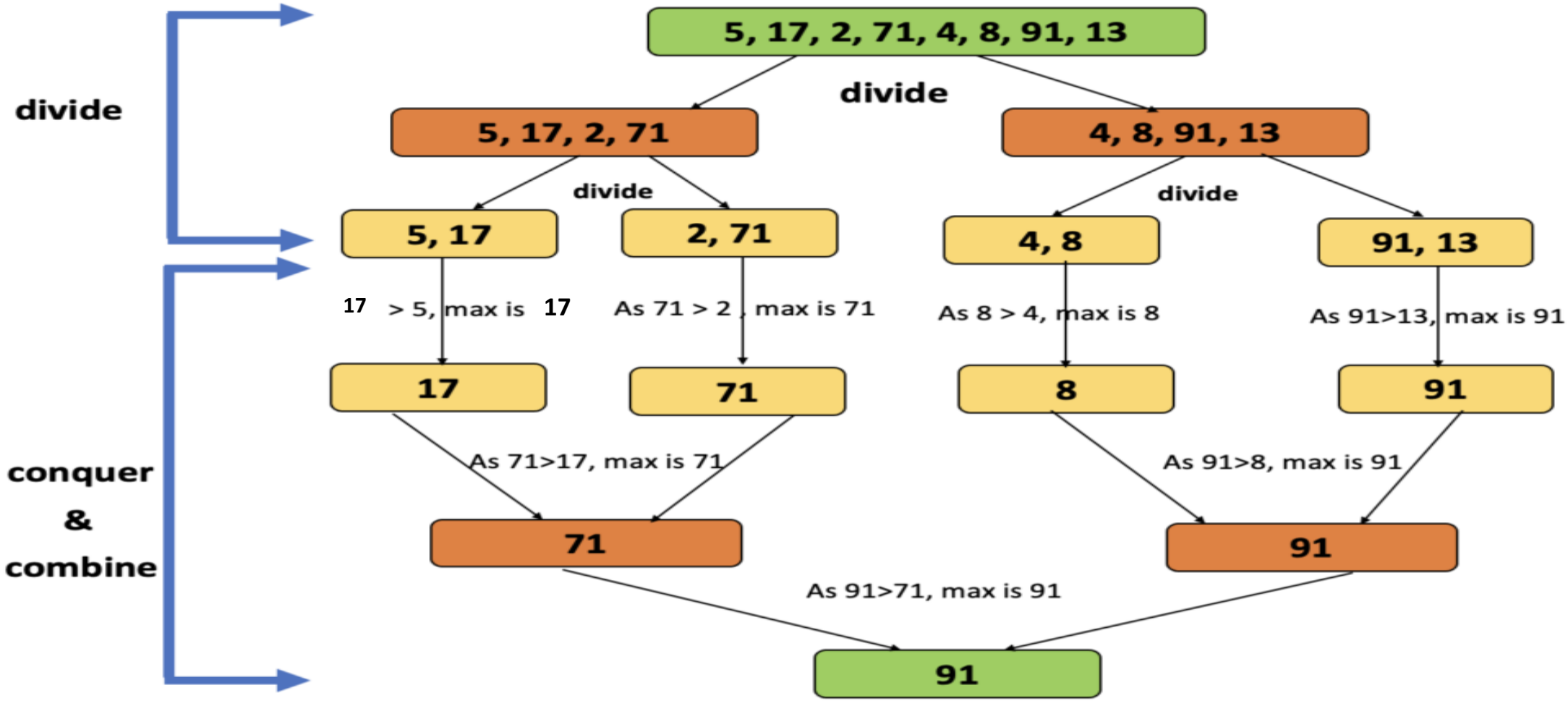
Combine
Combining sub-problem solutions to build the original problem solution



Divide and conquer

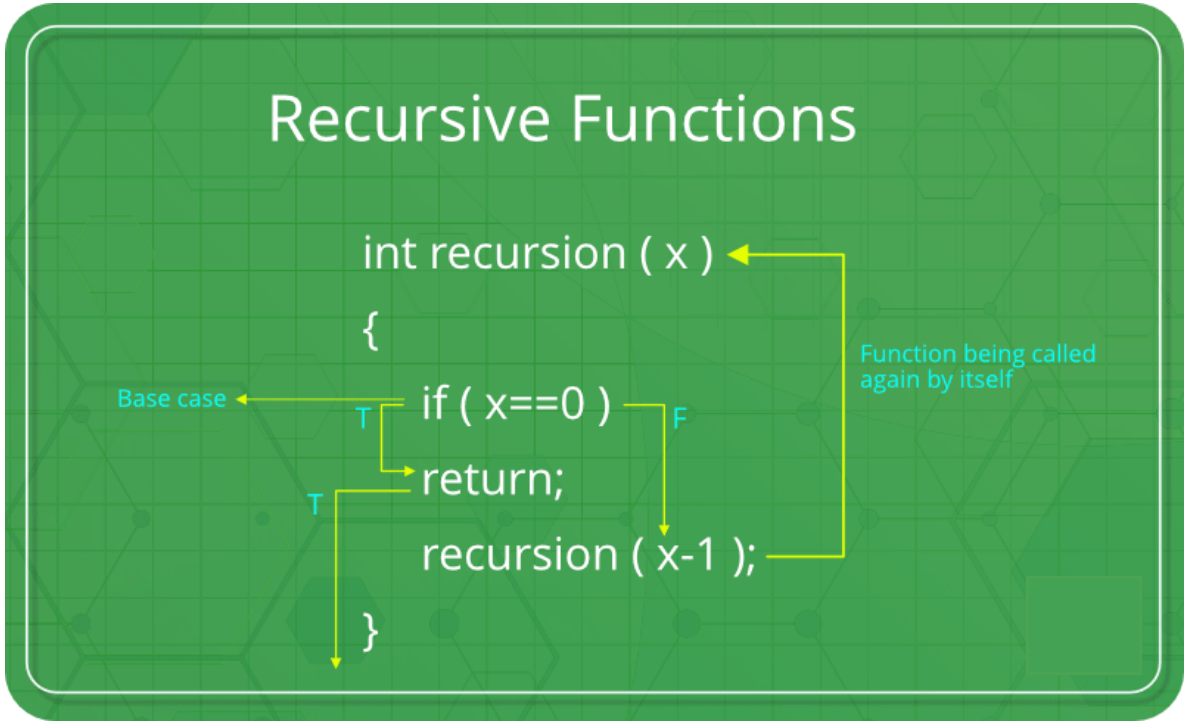


Divide and conquer (Find The Maximum)



Recursive Functions

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



Divide and conquer Algorithm

```
Divide_Conquer(problem P){
   if Small(P) return SimpleSolution(P);
   else {

     divide P into smaller instances P1, P2, ..., Pk, k≥2;

     Apply Divide_Conquer to each of these subproblems;

      return Combine(Divide_Conque(P1), Divide_Conque(P2),...,
      Divide_Conque(Pk));
   }
}
```

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 parallelly 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, 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.

Assignment 2 (two weeks)

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

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.

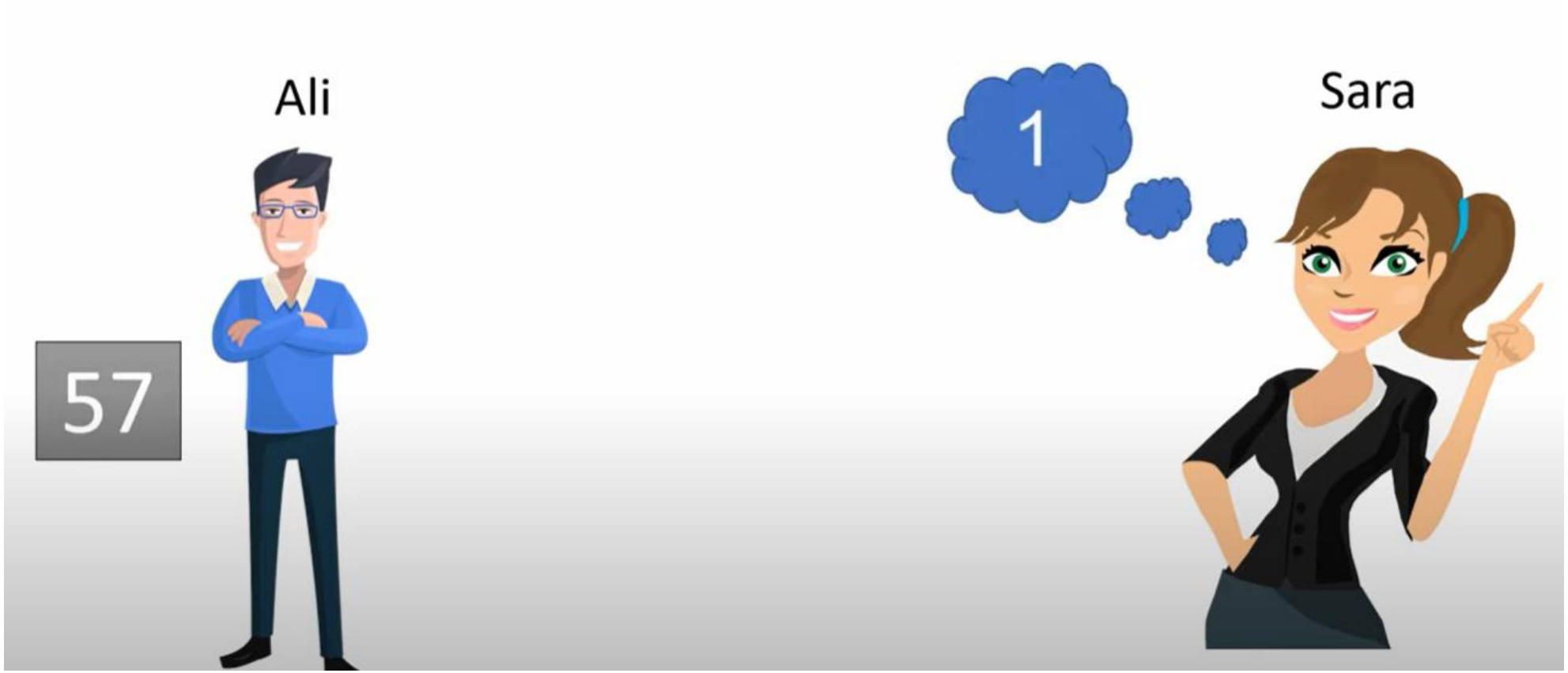
Some standard algorithms that follow Divide and Conquer algorithm

- Binary Search
- Merge Sort
- Quick Sort
- Closest Pair of Points
- Strassen's Algorithm (matrix multiplication)
- Finding maximum and minimum

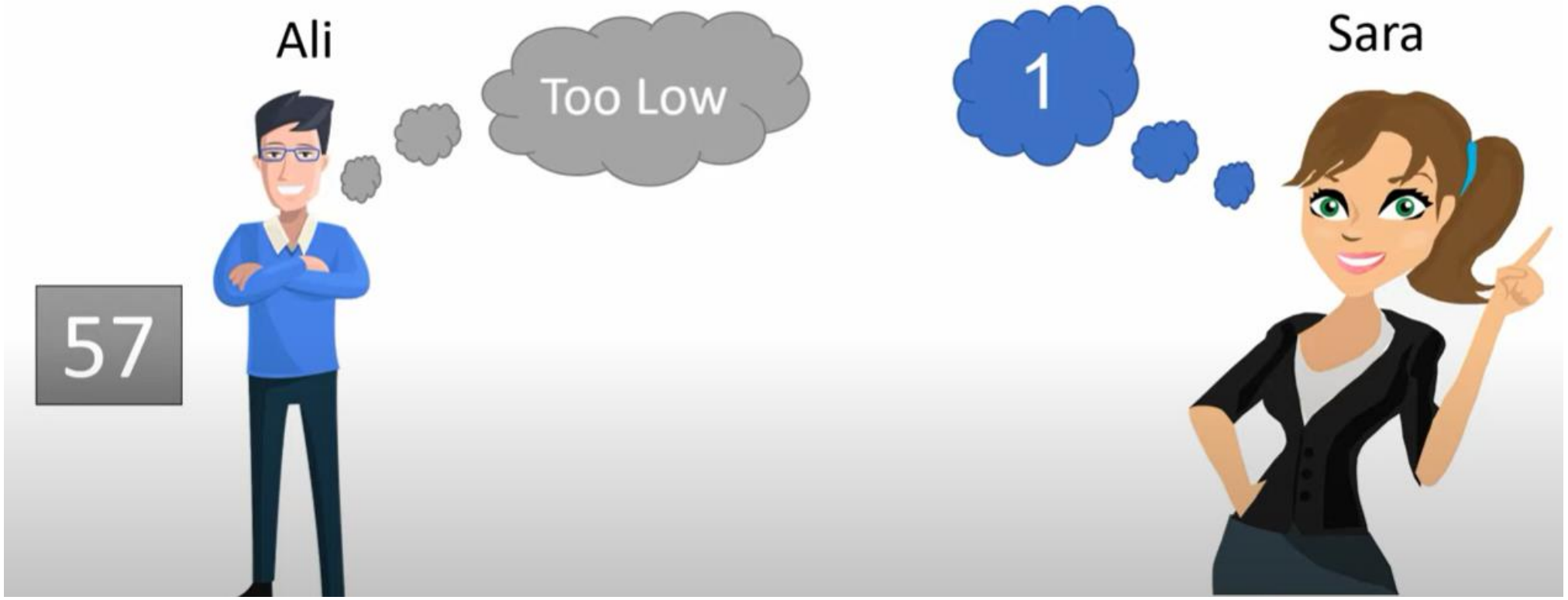
Guess the number from 0 to 100
[Traditional Search]

Guess the number [Traditional Search]

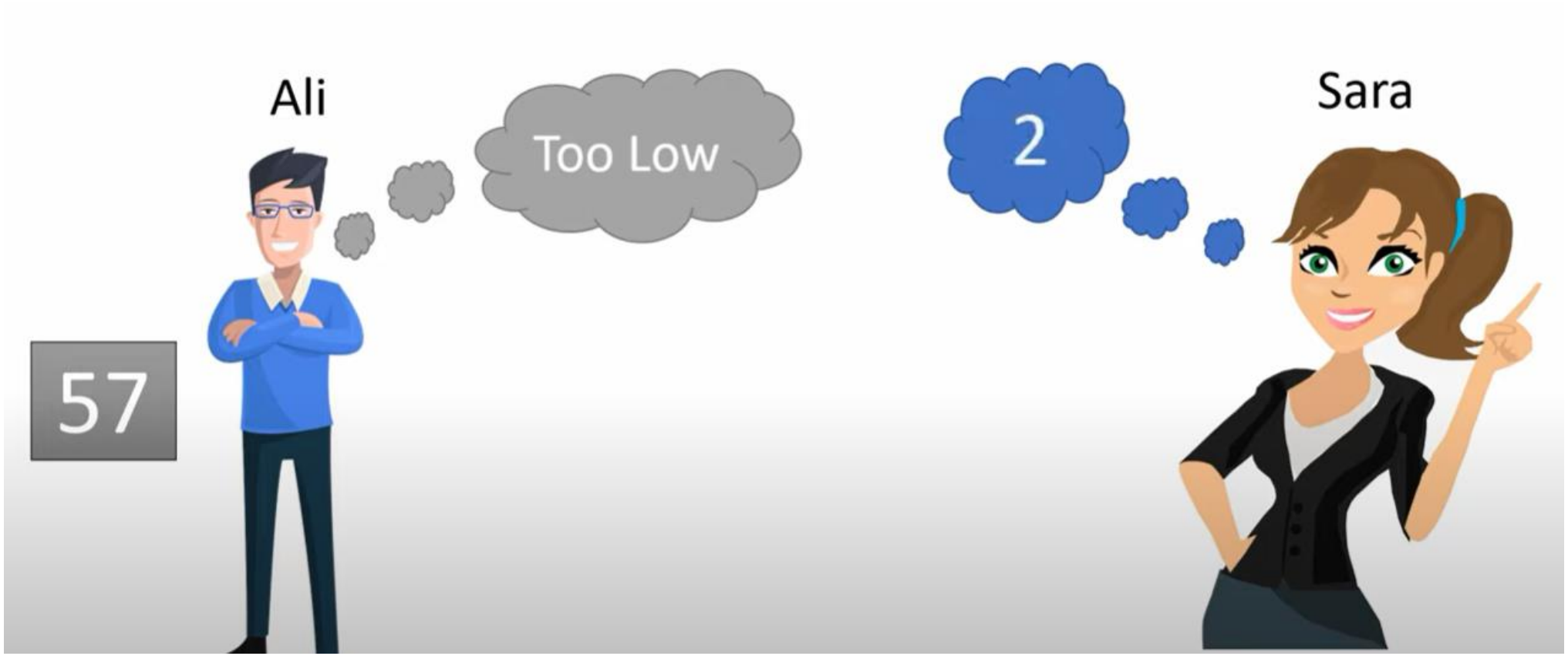
Guess the number [Traditional Search]



Guess the number [Traditional Search]

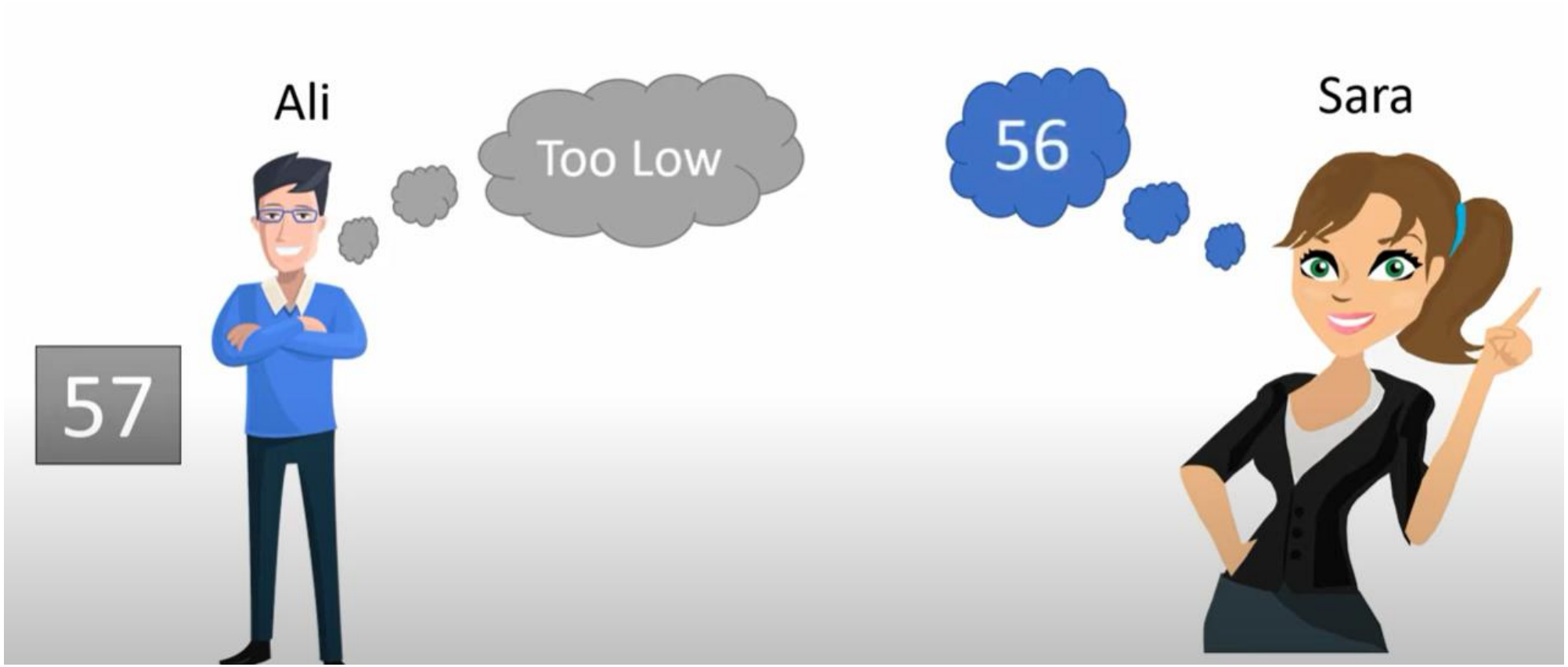


Guess the number [Traditional Search]



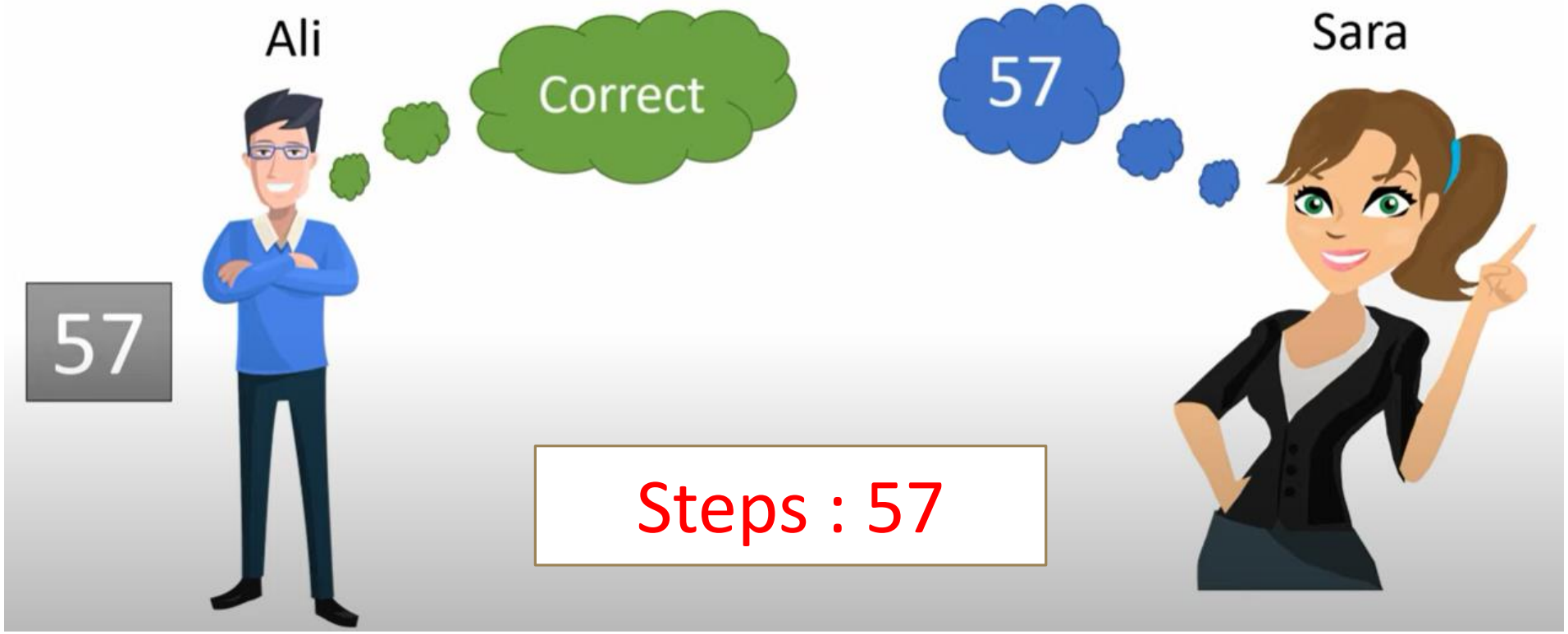
Guess the number [Traditional Search]

Guess the number [Traditional Search]



Guess the number [Traditional Search]

Sara follows the linear Search method

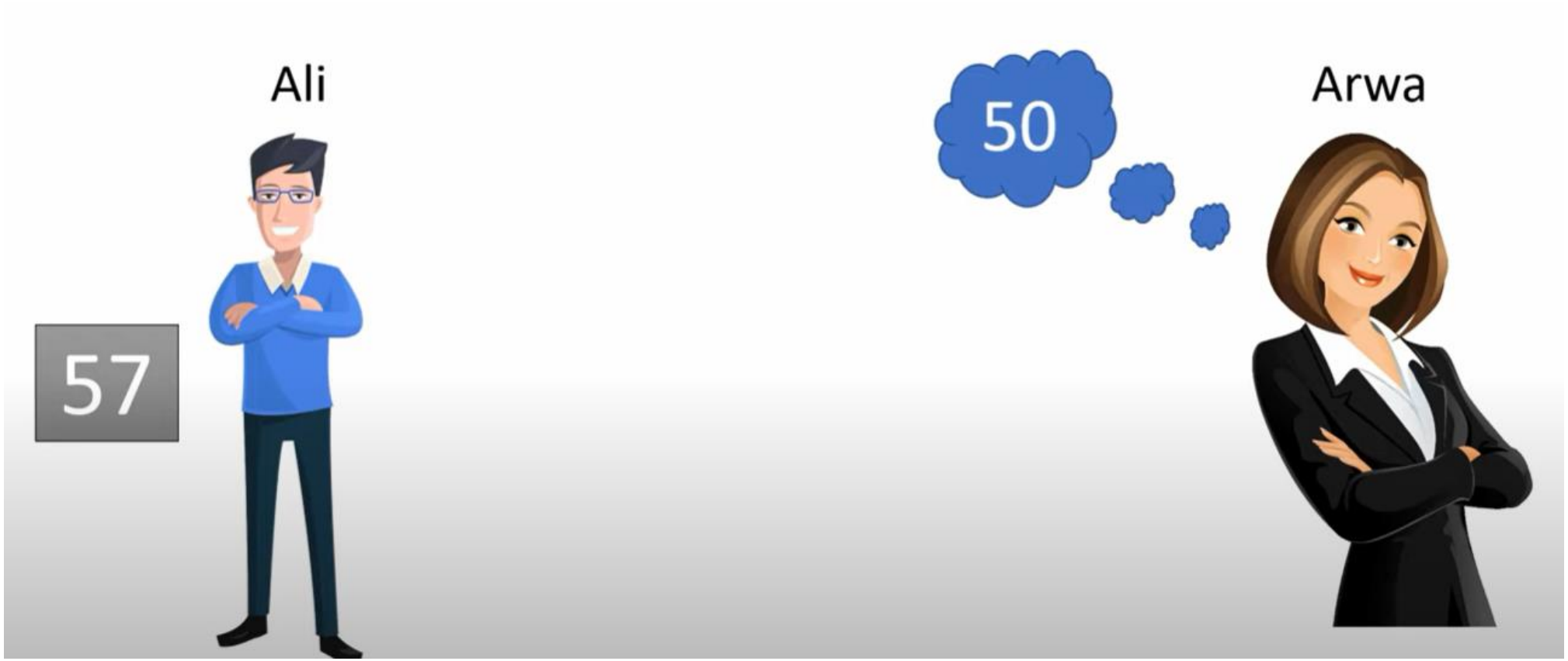


Guess the number from 0 to 100

[Binary Search]

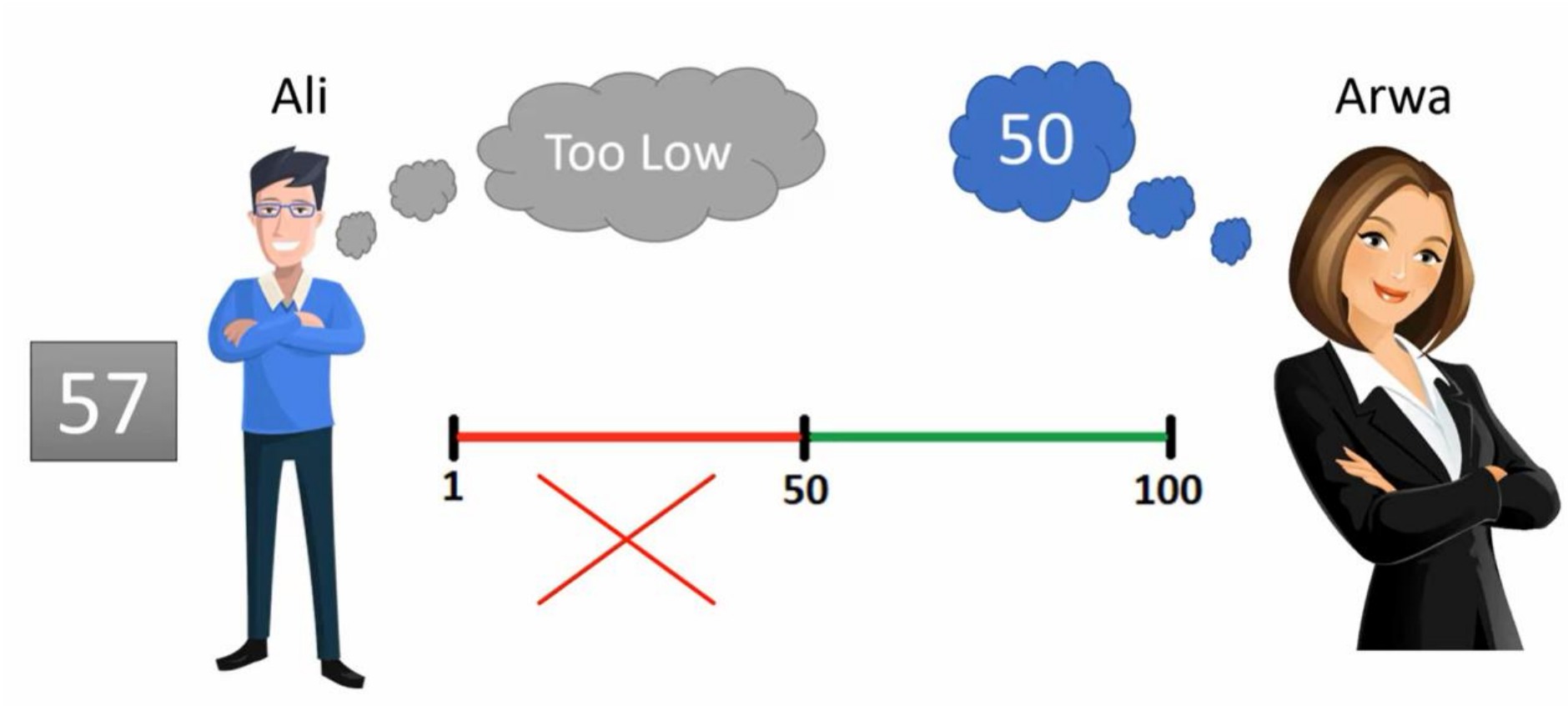
Guess the number [Binary Search]

Guess the number [Binary Search]

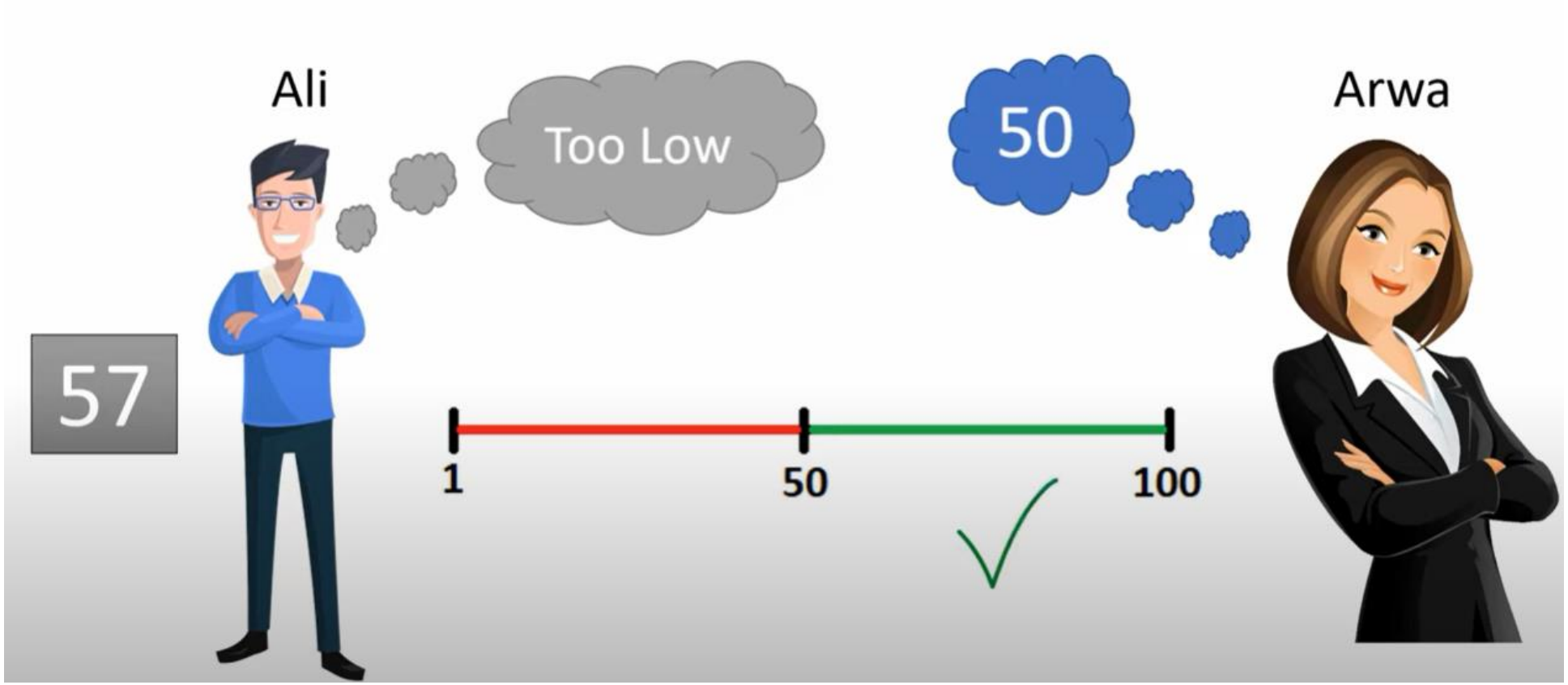


Guess the number [Binary Search]

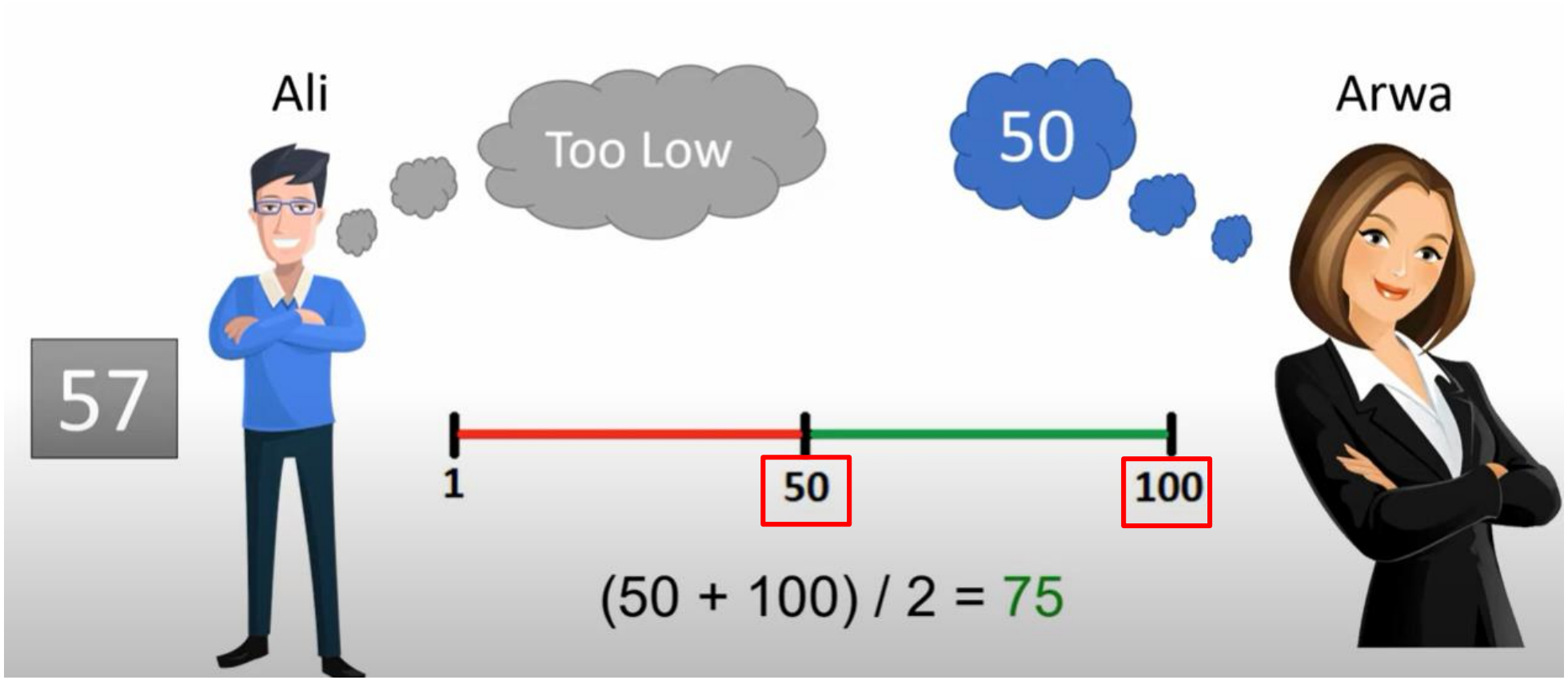
Guess the number [Binary Search]



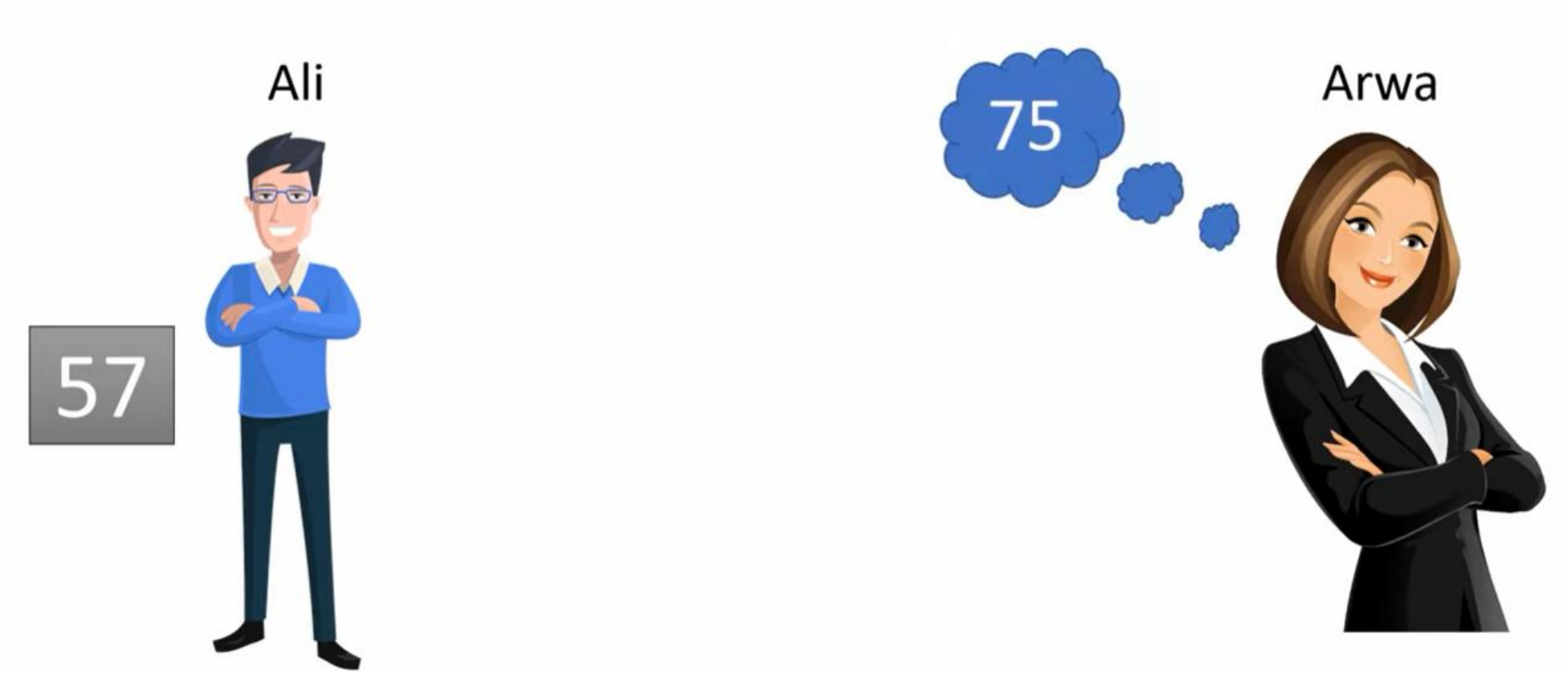
Guess the number [Binary Search]



Guess the number [Binary Search]

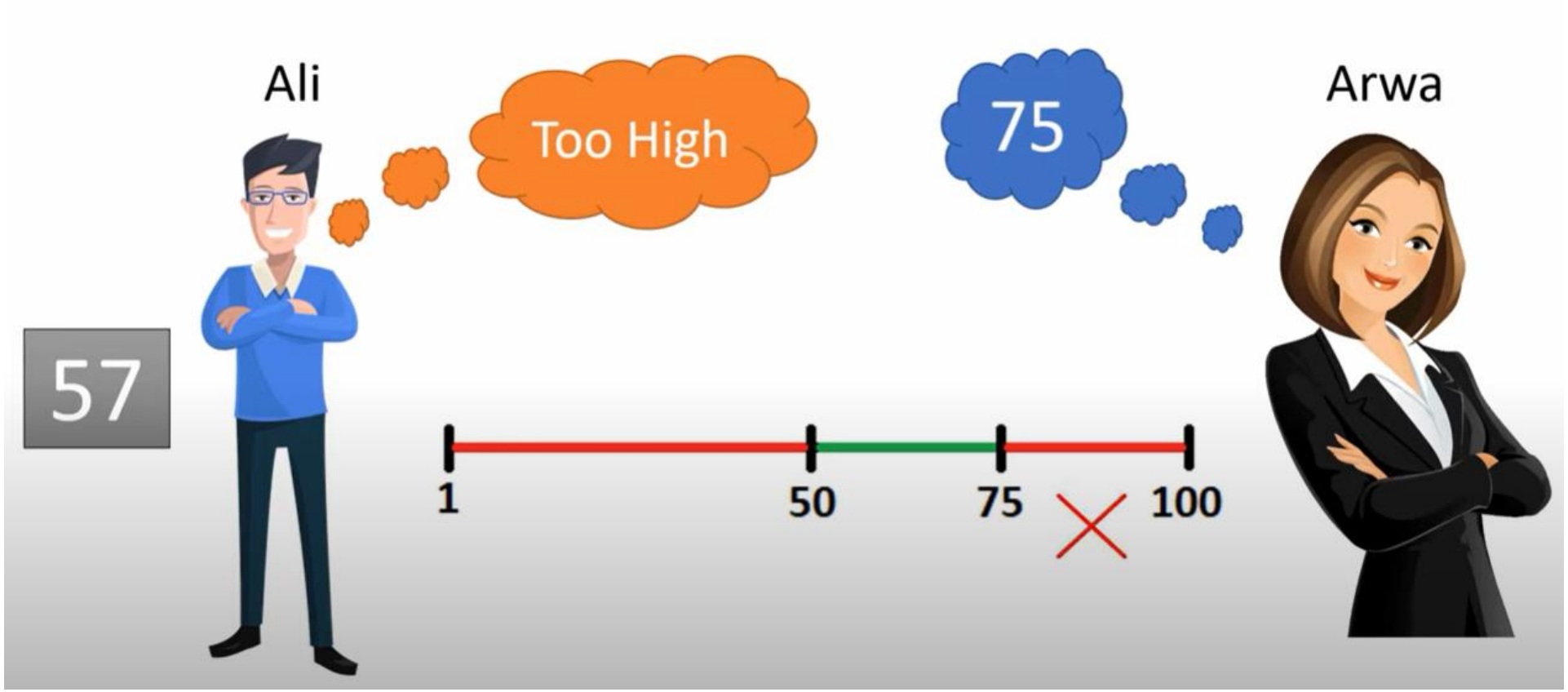


Guess the number [Binary Search]

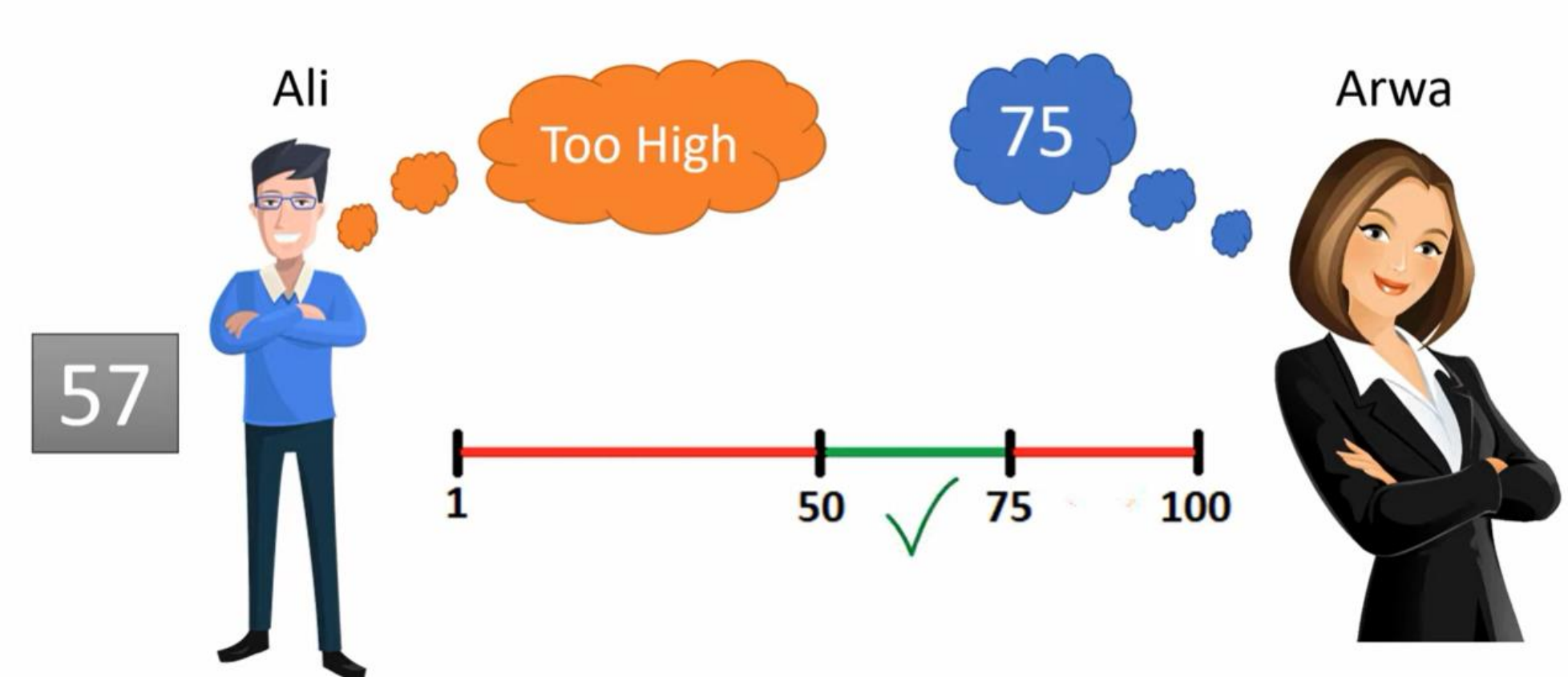


Guess the number [Binary Search]

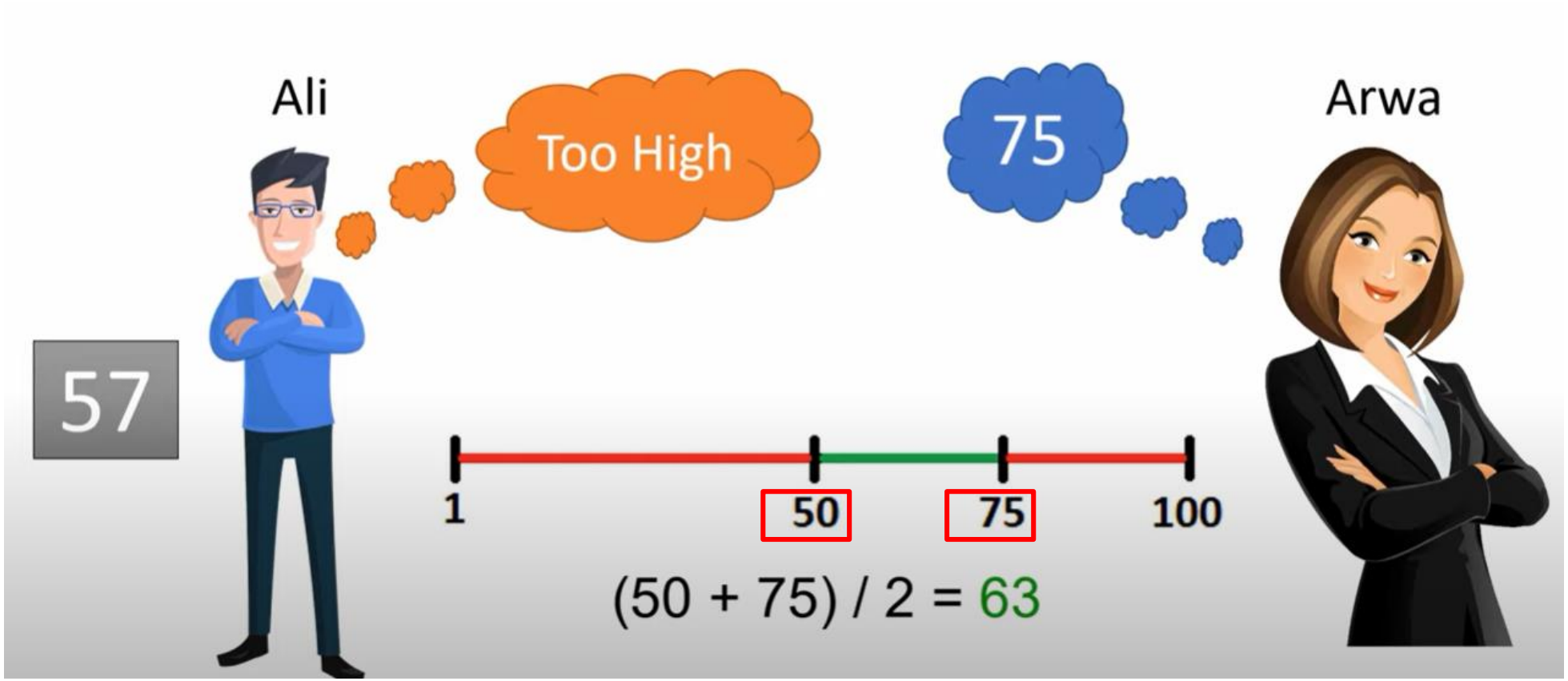
Guess the number [Binary Search]



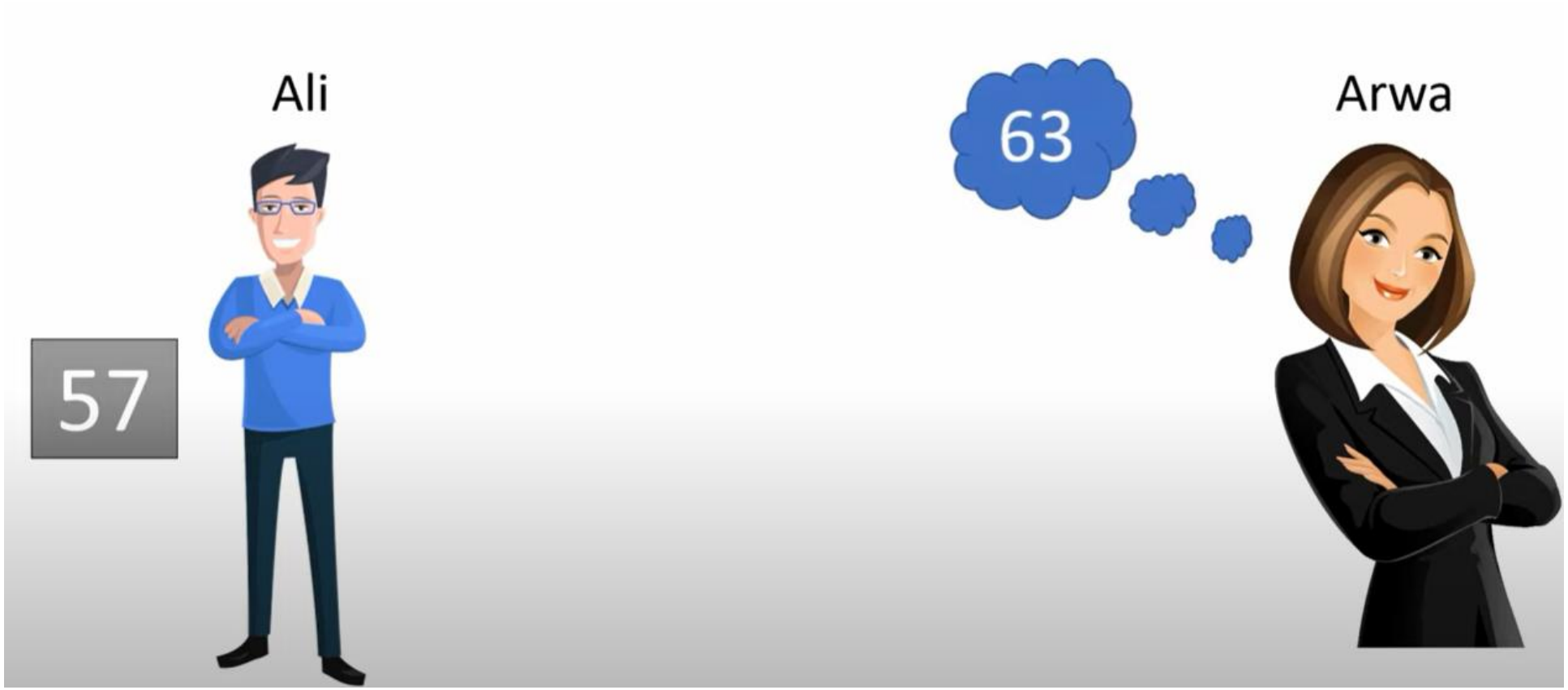
Guess the number [Binary Search]



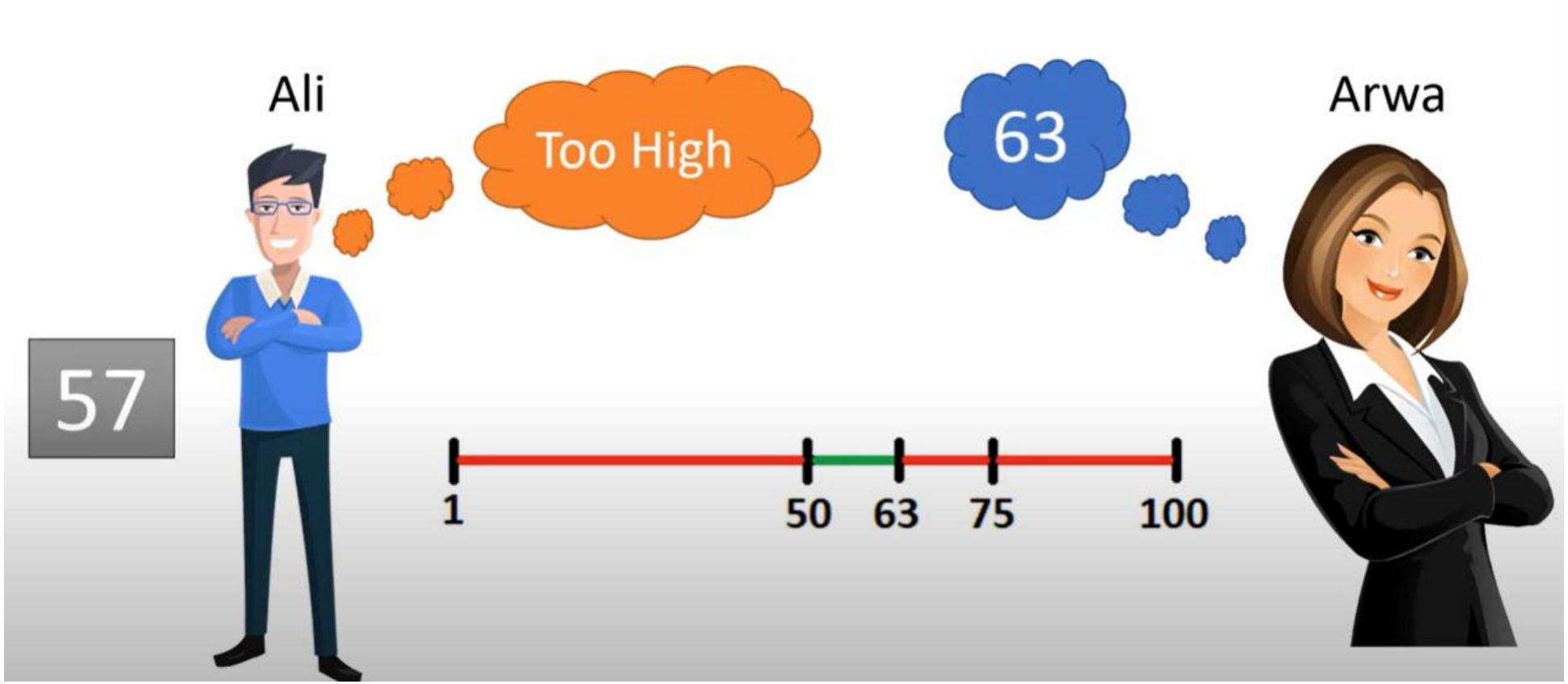
Guess the number [Binary Search]



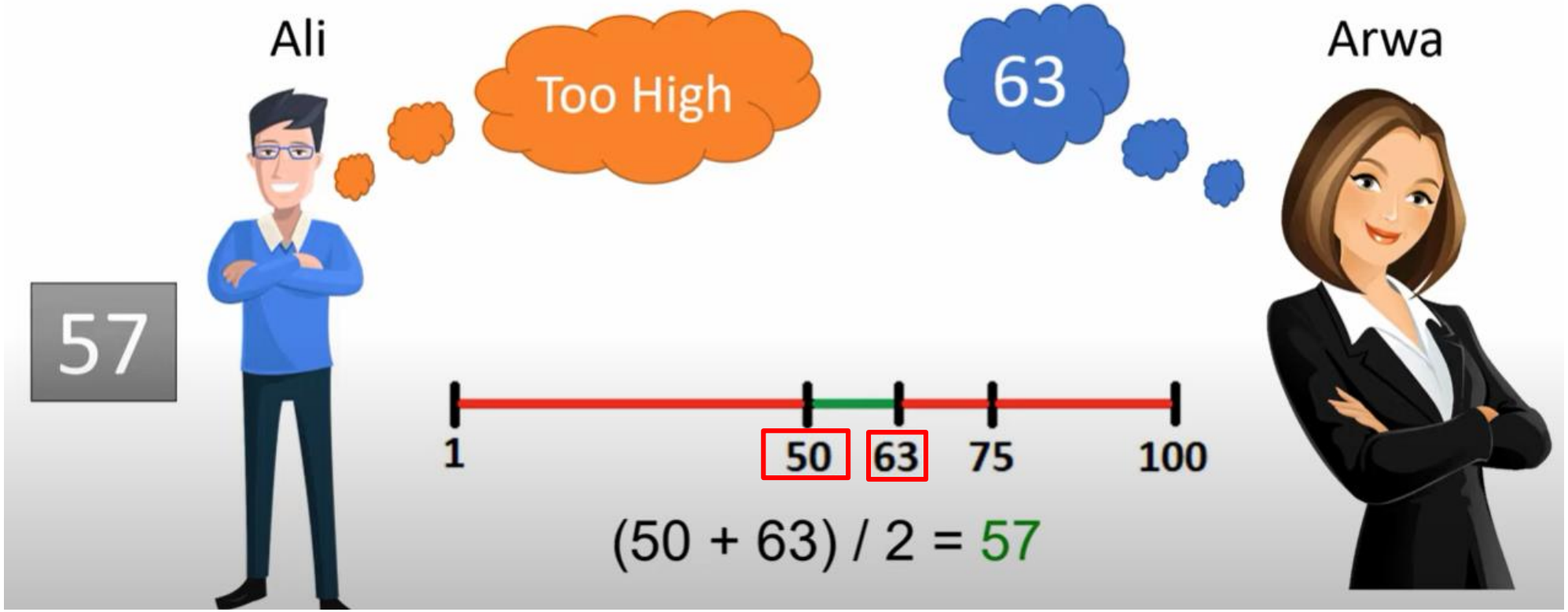
Guess the number [Binary Search]



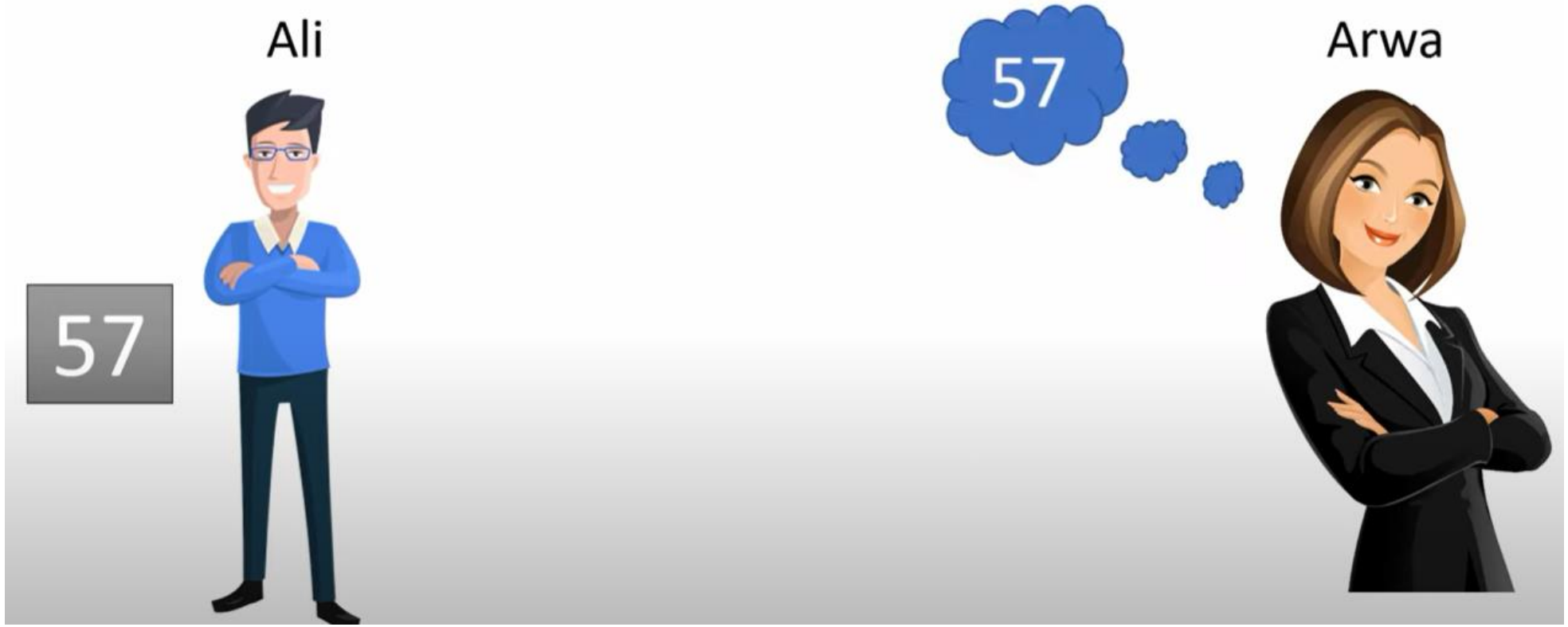
Guess the number [Binary Search]



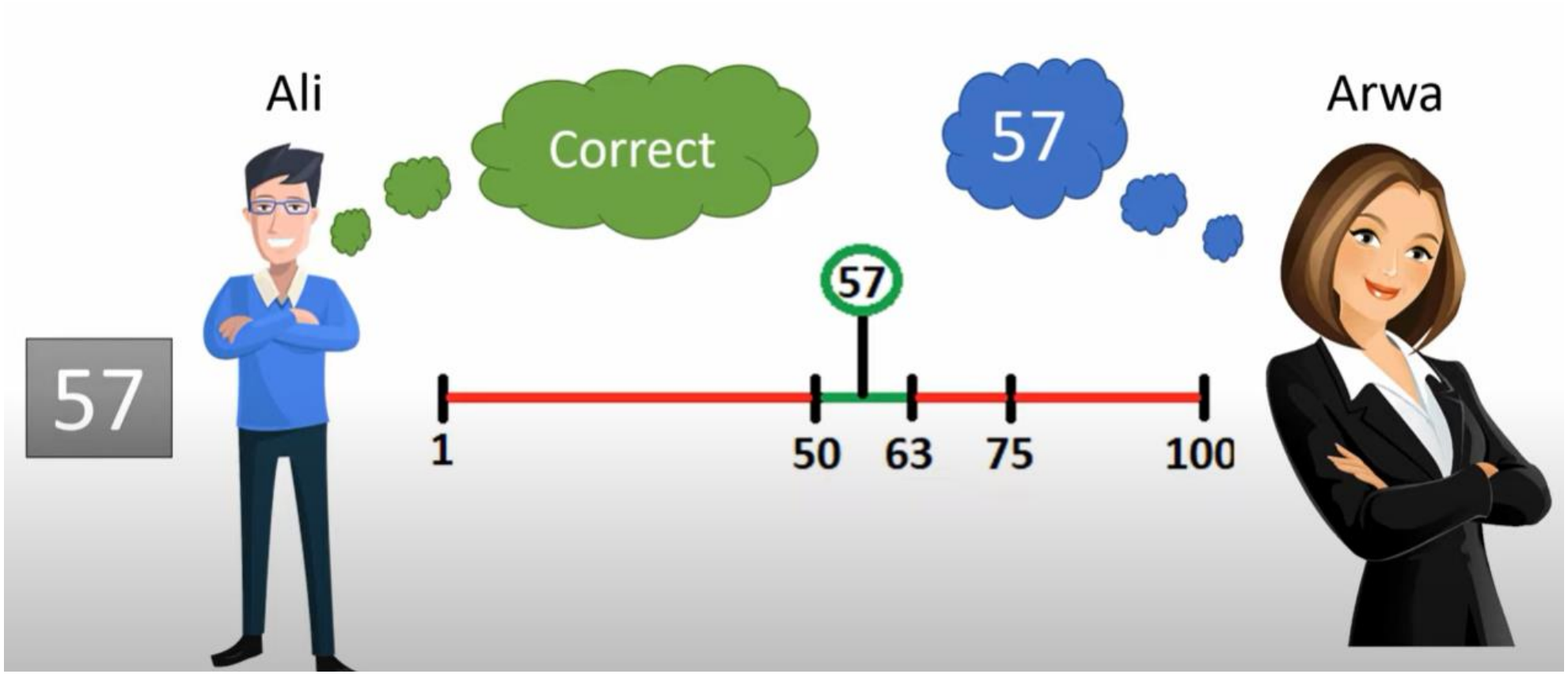
Guess the number [Binary Search]



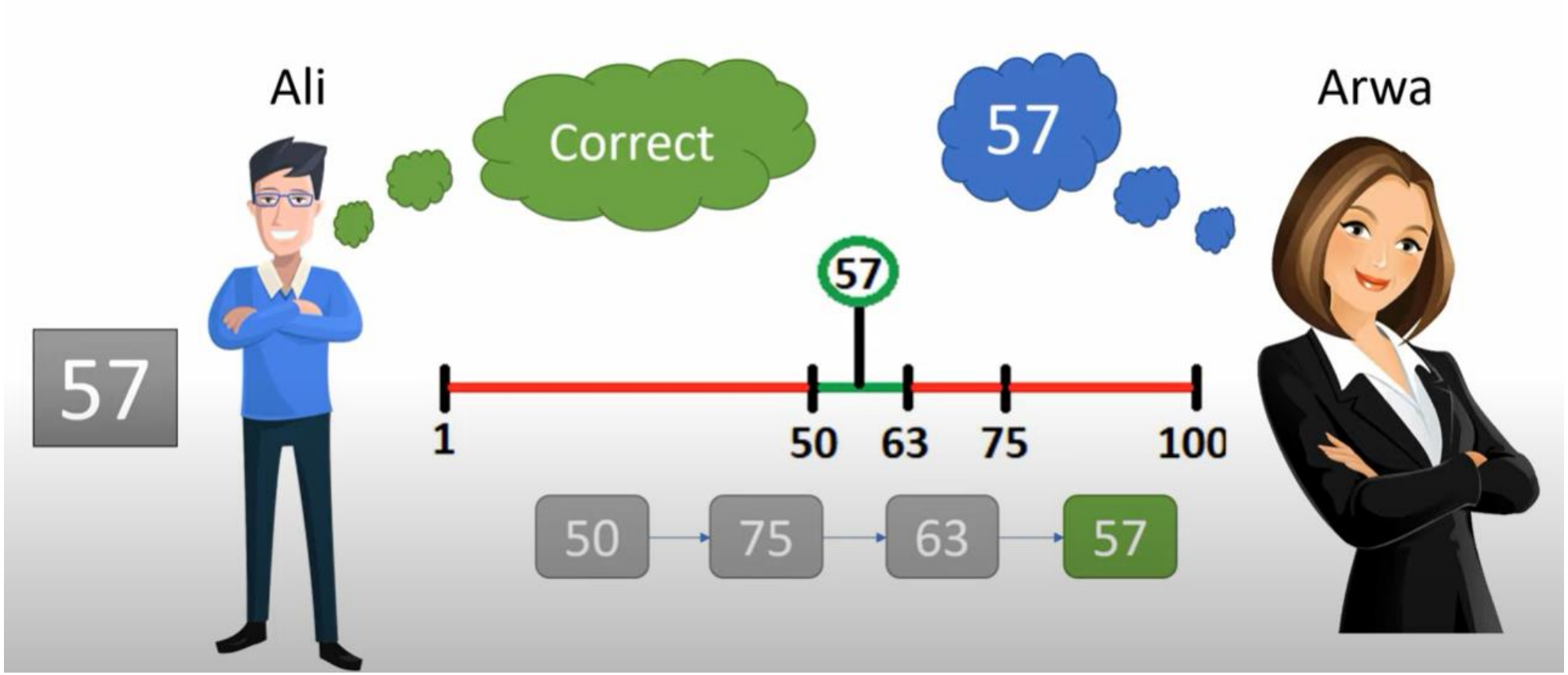
Guess the number [Binary Search]



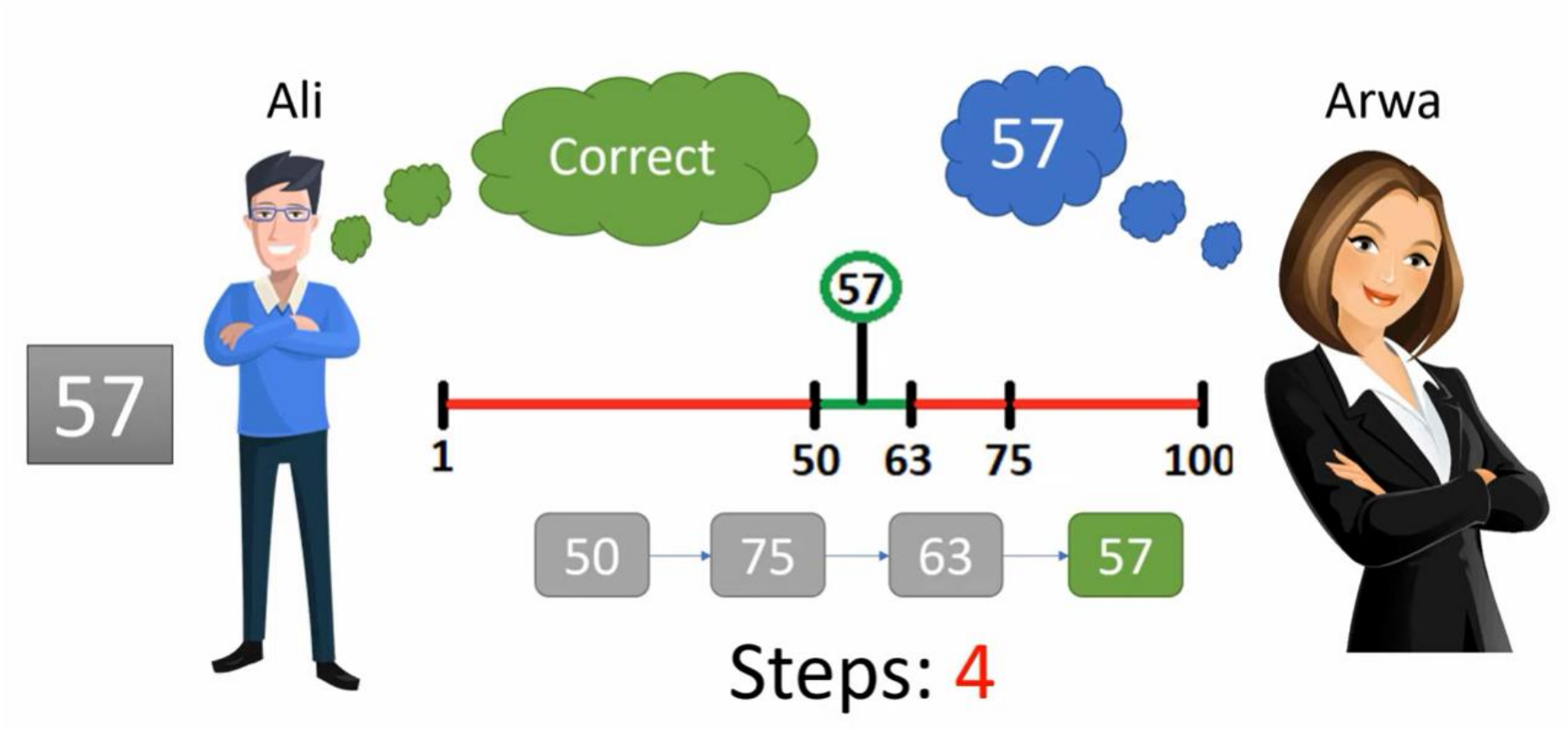
Guess the number [Binary Search]



Guess the number [Binary Search]



Guess the number [Binary Search]



Binary Search

- **Problem Definition:** Given a sorted array A=(a_1, a_2,…, 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($A(l..m\text{-}1),k$)

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

End.

Recursion

Some standard algorithms that follow Divide and Conquer algorithm

- Binary Search
- Merge Sort
- Quick Sort
- Closest Pair of Points
- Strassen's Algorithm (matrix multiplication)
- Finding maximum and minimum

Merge Sort Algorithm

- **Merge Sort** is one of the most popular sorting algorithms that is based on the principle of Divide and Conquer Algorithm.
- Here, a problem is divided into multiple sub-problems. Each sub-problem is solved individually. Finally, sub-problems are combined to form the final solution.
- **Definition**

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 $a_i \le a_{i+1}$, $1 \le i \le n-1$.

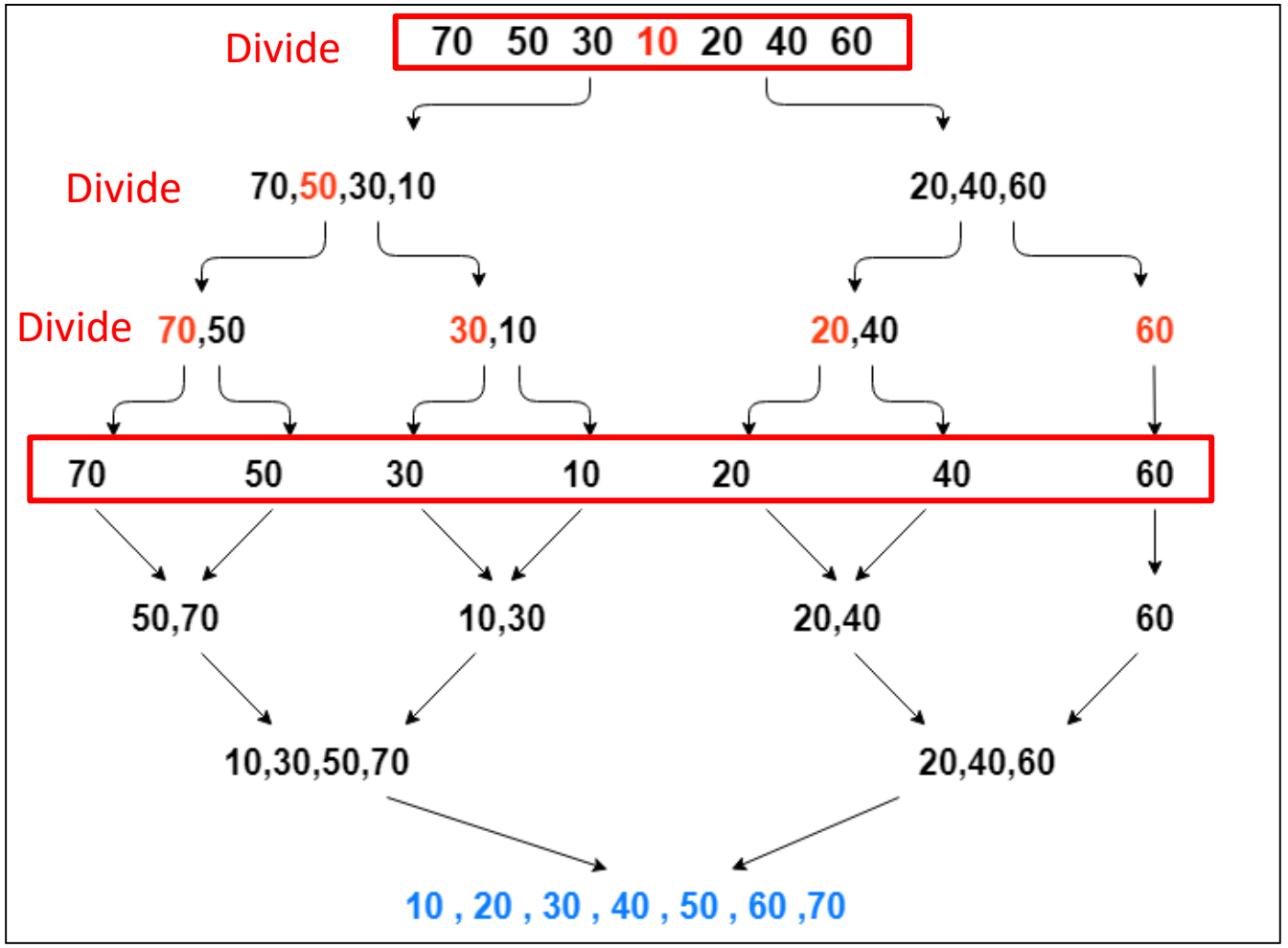
Example 1: Given A=(20,4,10,17,6,7,2)

Goal A=(2,4,6,7,10,17,20)

How Merge Sort Works

- **Merge sort** works in the following way:

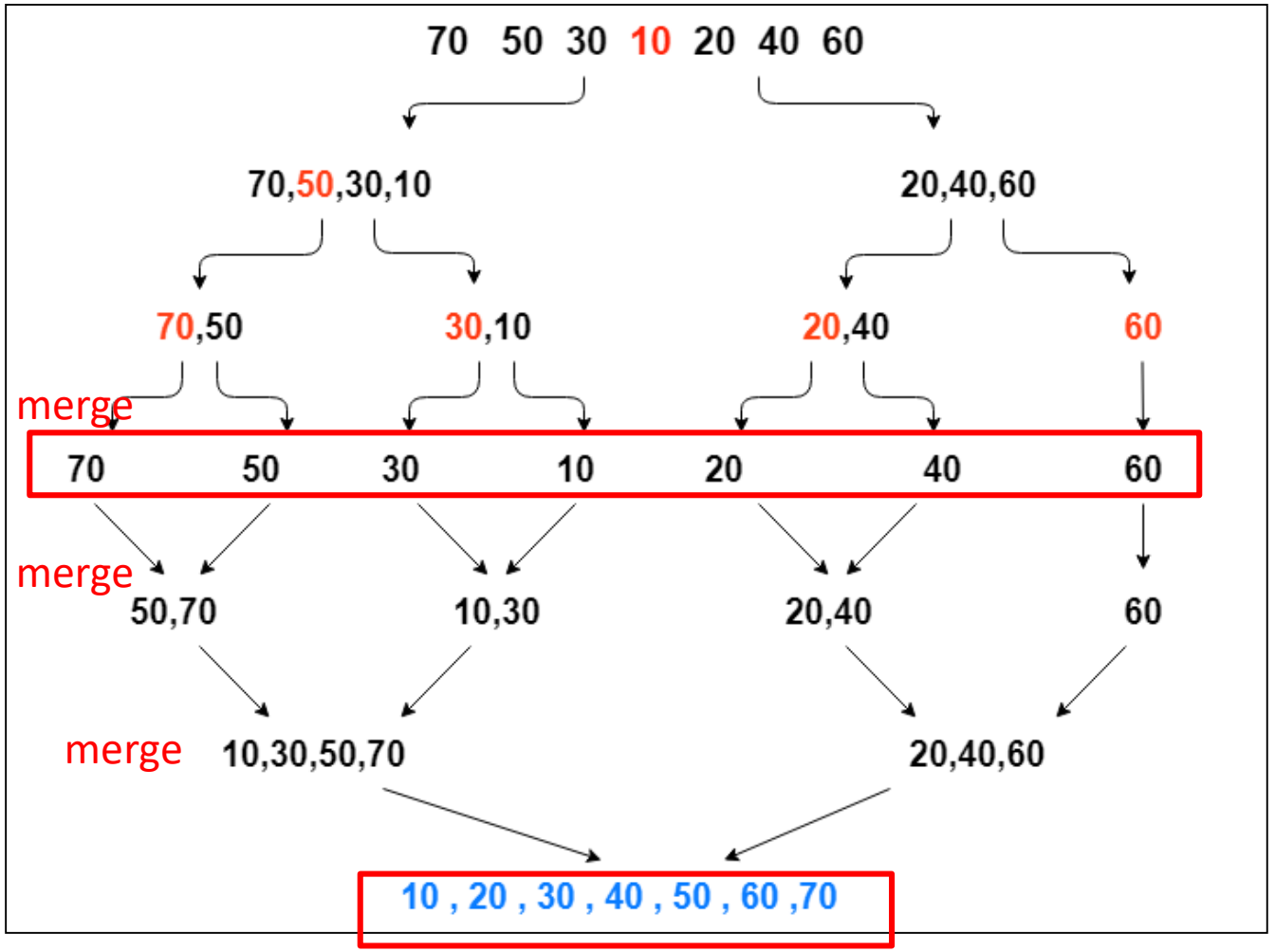
  - **Merge sort**, take middle index in data set and split into two collections : one collection for items left of middle index and second for right values of middle index.
  - Repeat step 1 as long as these collections size reach to one item only.



How Merge Sort Works

- **Merge sort** works in the following way:

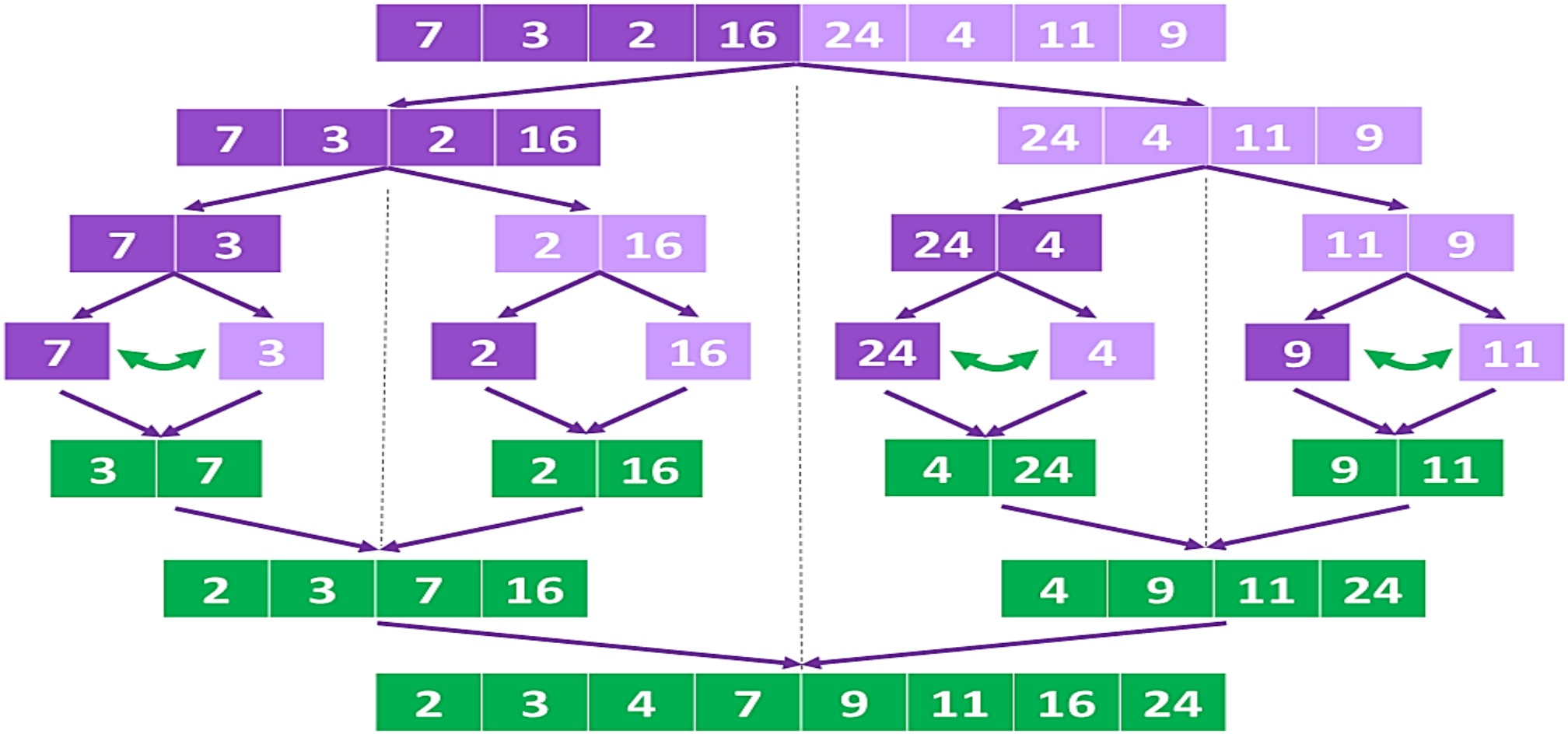
  - Now merge sort picks the items from smaller collections, sort them and merge to create new collection.
  - Repeat this sort and merge process until all small collection sort and merge not completed.
  - Finally will get single collection with sorted result.



How Merge Sort Works *(Main Idea)*

- Divide the array A(1 .. n) into two subarrays A(1 .. m) and A(m + 1 .. n), where m = n/2.
- Recursively mergesort the subarrays A(1 .. m) and A(m + 1.. n).
- Merge the newly sorted subarrays A(1 .. m) and A(m + 1 .. n) into a single sorted array A.

How Merge Sort Works



Split sub-lists in two until you reach pair of values.

Sort/swap pair of values if needed.

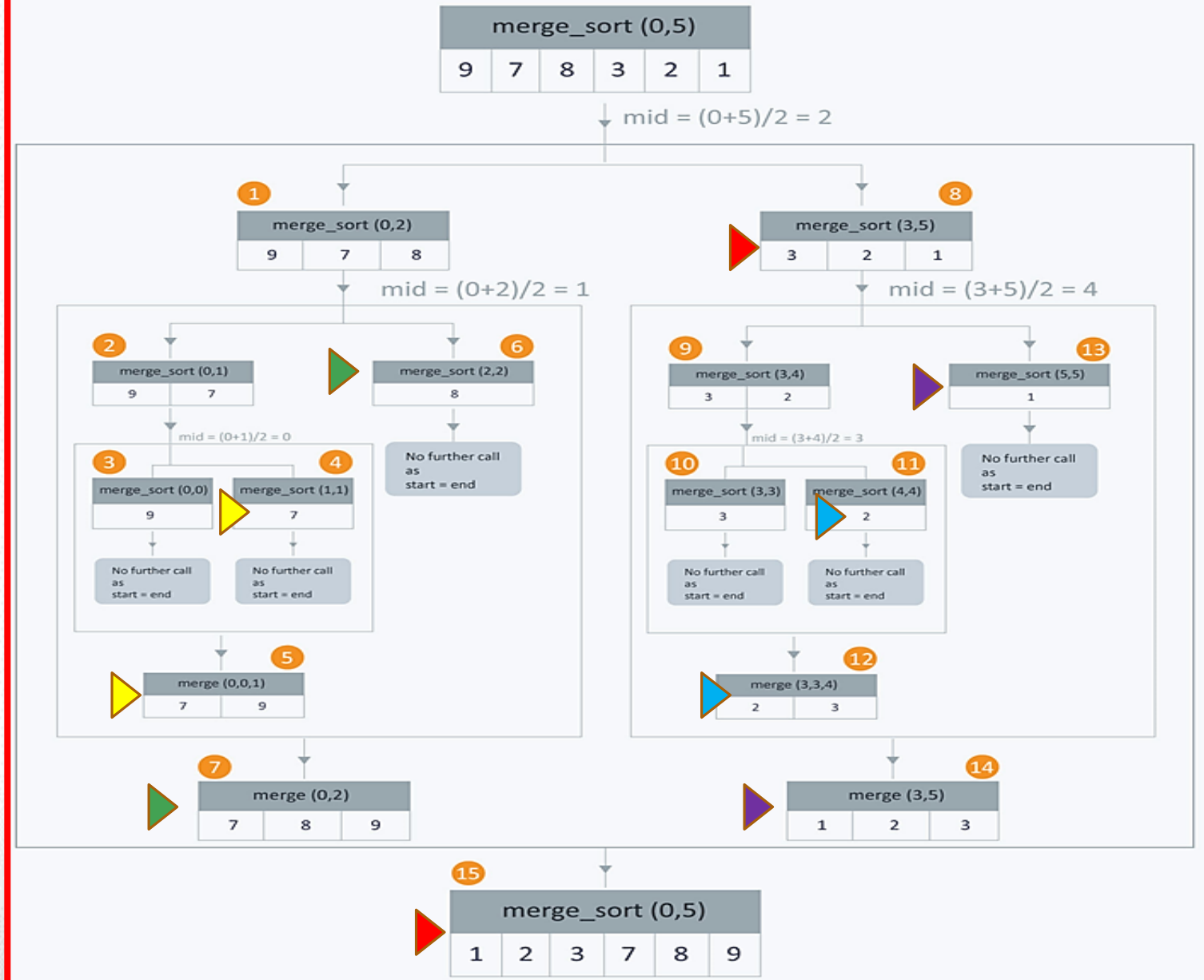
Merge and sort sub-lists and repeat process till you merge to the full list.

Pseudo Code

```
MERGE-SORT(A, p, r)
1   if p < r
2       q = ⌊(p + r)/2⌋
3       MERGE-SORT(A, p, q)
4       MERGE-SORT(A, q + 1, r)
5       MERGE(A, p, q, r)
```

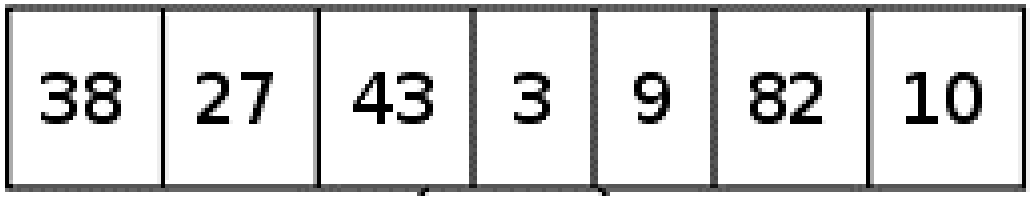
```
// merge sort sorting algorithm

// merge function
void merge(int arr[], int l, int m, int r)
{

}
```

```
// merge sort function
void mergeSort(int arr[],int l, int r)
{
    if(l<r)
    {
        int m = (l+r)/2;
        mergeSort(arr,l,m);
        mergeSort(arr,m+1,r);
        merge(arr,l,m,r);
    }
}
```

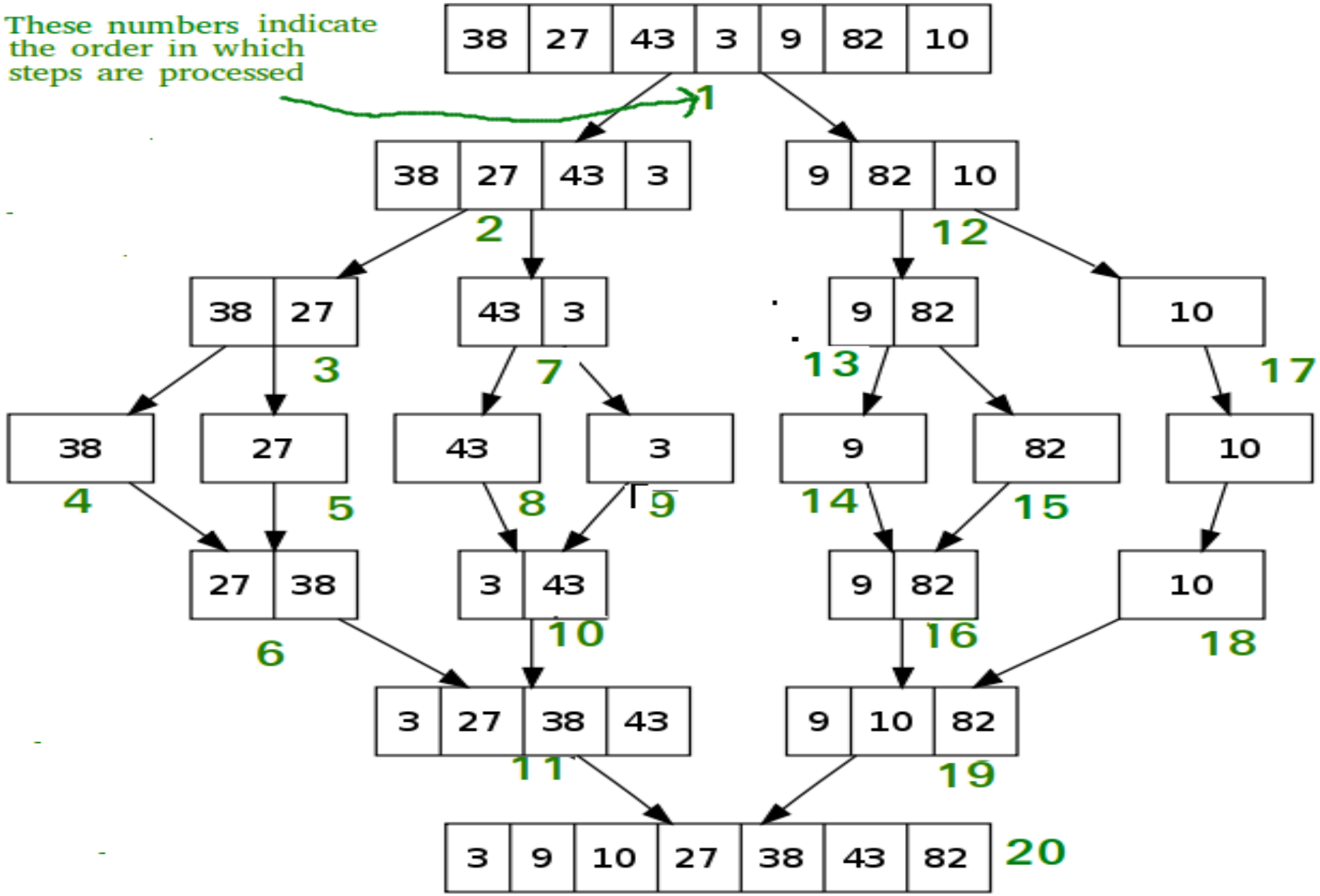


Tracing Merge Sort Algorithm

Indicate the order in which steps are processed in merge sort algorithm



Answer


These numbers indicate the order in which steps are processed
38 27 43 3 9 82 10
1
38 27 43 3
9 82 10
2
12
38 27
43 3
9 82
10
3
7
13
17
38
27
43
3
9
82
10
4
5
8
9
14
15
27 38
3 43
9 82
10
6
10
16
18
3 27 38 43
9 10 82
11
19
3 9 10 27 38 43 82
20