LECTURE 09

RECURSION

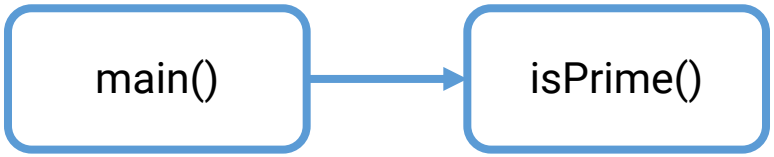
Big-O Coding

Website: www.bigocoding.com

Giới thiệu

Hàm f() gọi hàm g()

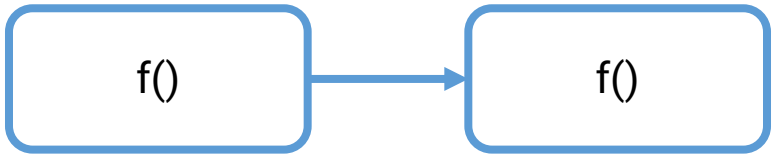
- Chương trình kiểm tra số nguyên tố.



- Chương trình đếm các số nguyên tố trong mảng.

Hàm đệ qui

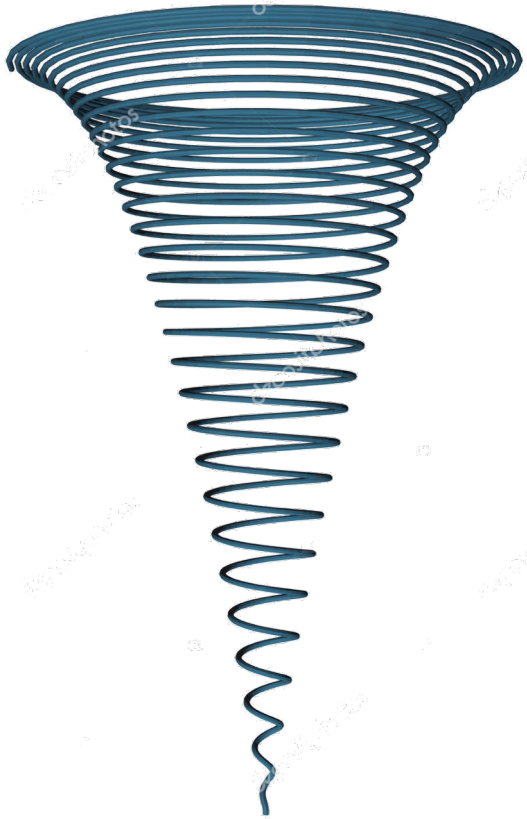
- Hàm đệ qui là hàm mà trong code **gọi lại chính nó**.



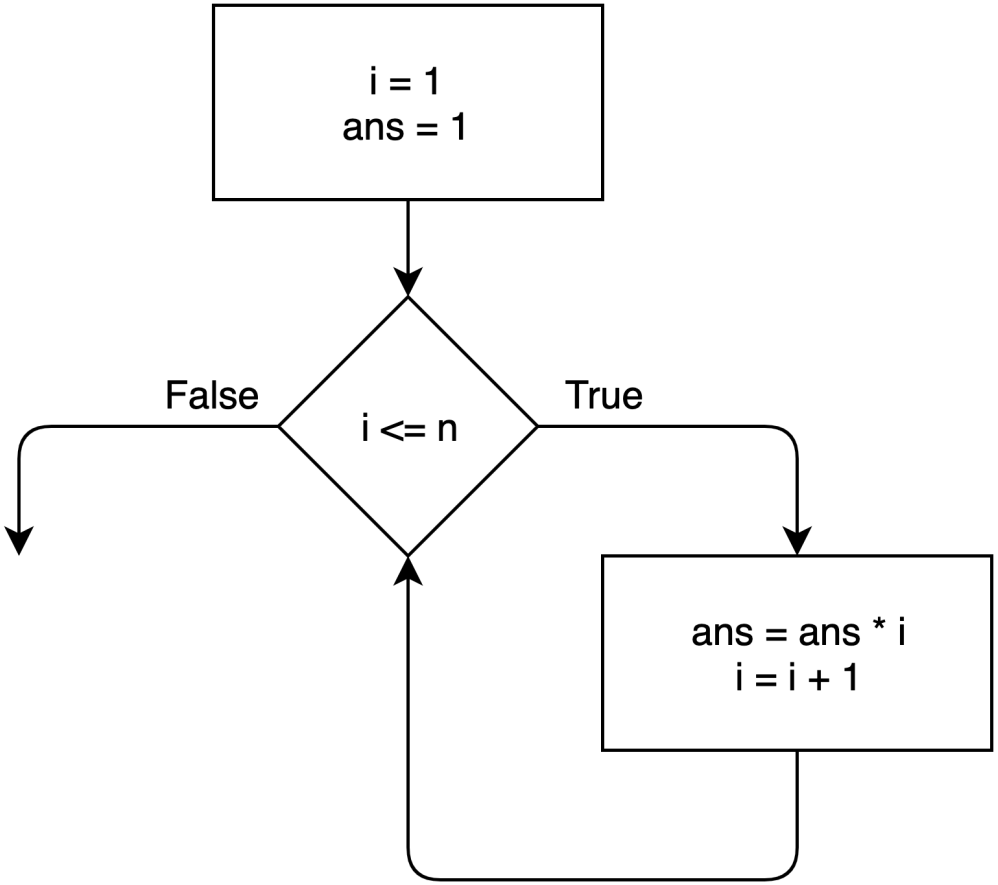
Credit to Marvel and Minh Tran ☺

Hàm đệ qui

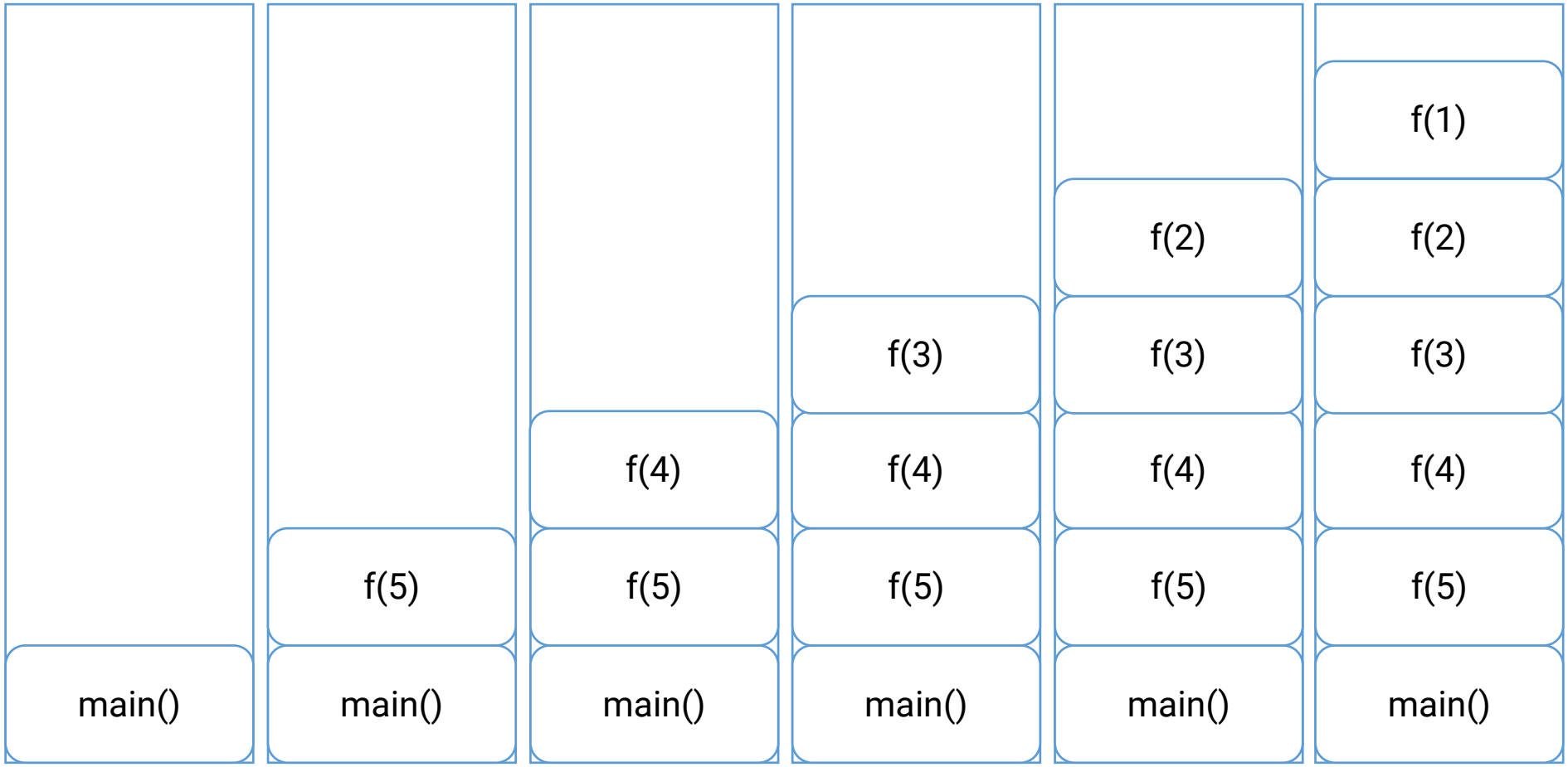
- Gọi lại thực hiện chính hàm đó, **nhưng với dữ liệu xử lí nhỏ hơn**.



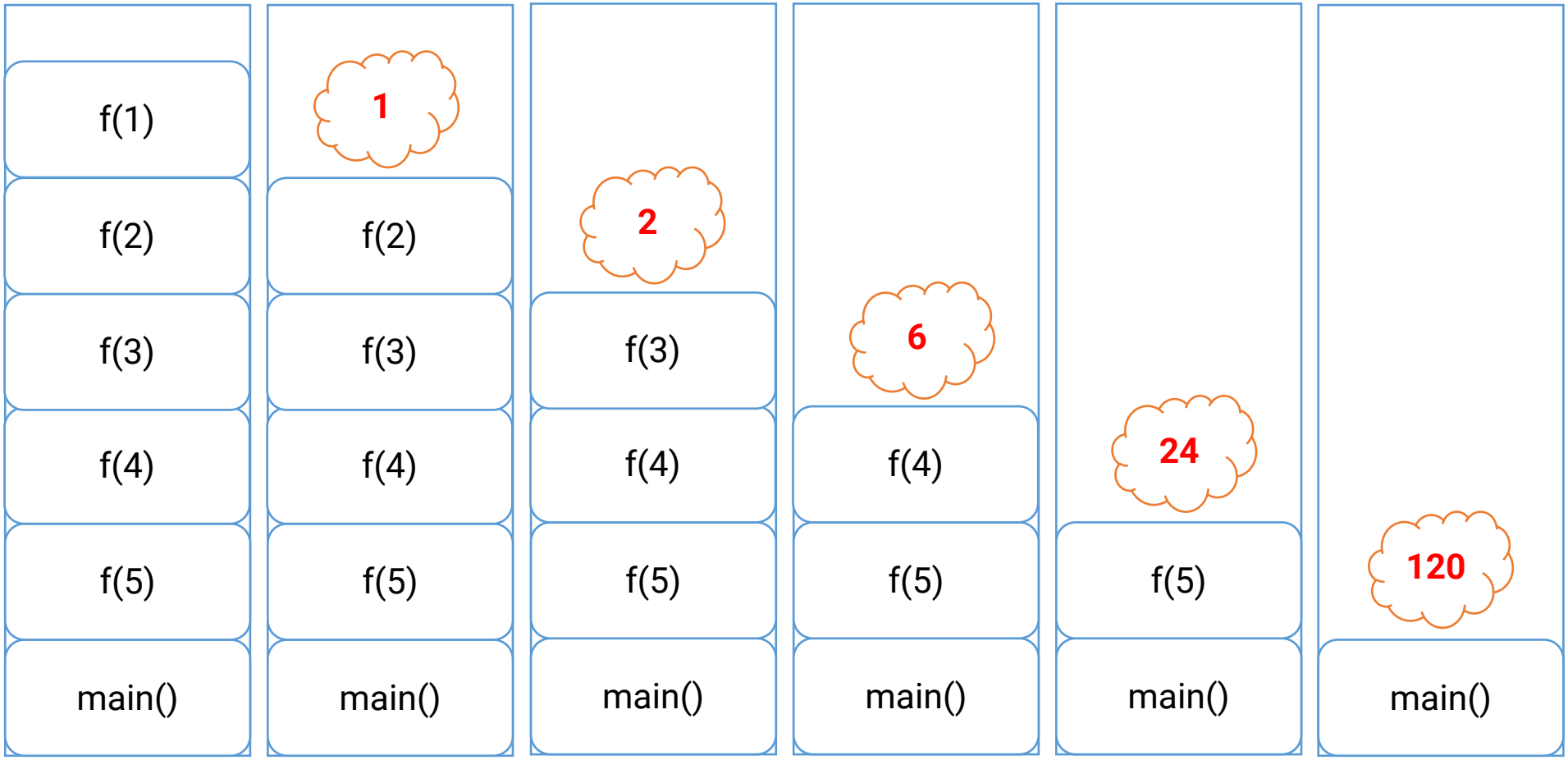
VD: Tính n! – Không đệ qui (n = 5)



VD: Tính n! – Đệ qui (n = 5)



VD: Tính n! – Đệ qui (n = 5)



Recursive case

- f(n) = n * f(n-1)
- f(n-1) = (n-1) * f(n-2)
- f(n-2) = (n-2) * f(n-3)
- f(n-3) = (n-3) * f(n-4)
- …

- f(3) = 3 * f(2)
- f(2) = 2 * f(1)
- **f(1) = 1 * f(0)**
- **f(0) = 0 * f(-1)**
- ➔ Có gì đó sai sai????

Base case

- Nếu không có base case, hàm đệ qui sẽ gọi lại chính nó liên tục và không có thời điểm dừng.
- **f(1) = 1**
- **f(0) = 1**

Tính n! – Mã nguồn

```java
int factorial(int n){
    if(n <= 0)
        return 1;
    return n * factorial(n-1);
}
```

```java
static int factorial(int n){
    if(n <= 0)
        return 1;
    return n * factorial(n - 1);
}
```

```python
import sys
sys.setrecursionlimit(10000)

def factorial(n):
    if(n <= 0):
        return 1
    return n * factorial(n - 1)
```

Divide-and-conquer approach

- Nhiều giải thuật thông dụng được diễn đạt theo kiểu đệ qui.
- Các giải thuật đệ qui theo hướng tiếp cận divide-and-conquer.
  - **Divide**: chia bài toán thành các bài toán con.
  - **Conquer**: giải trên từng bài toán con.
  - **Combine**: gom kết quả trên từng bài toán con để ra kết quả cuối cùng của bài toán.

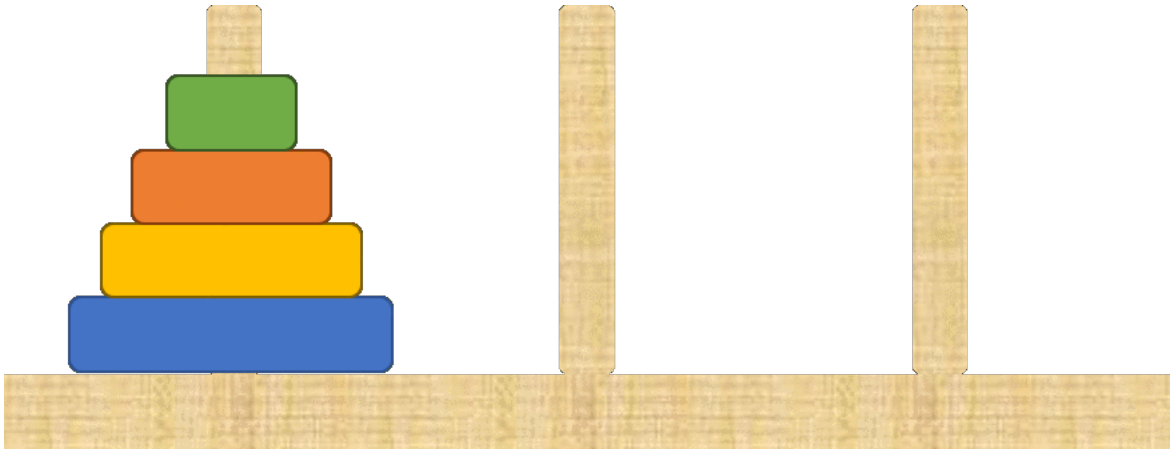
Một số ví dụ

1. Tính dung lượng của thư mục. Biết mỗi thư mục bao gồm tập tin và thư mục con.

2. Giải thuật sắp xếp merge sort.

3. Các bài toán kinh điển: tháp Hà Nội, 8 hậu, mã đi tuần.

Tháp Hà Nội



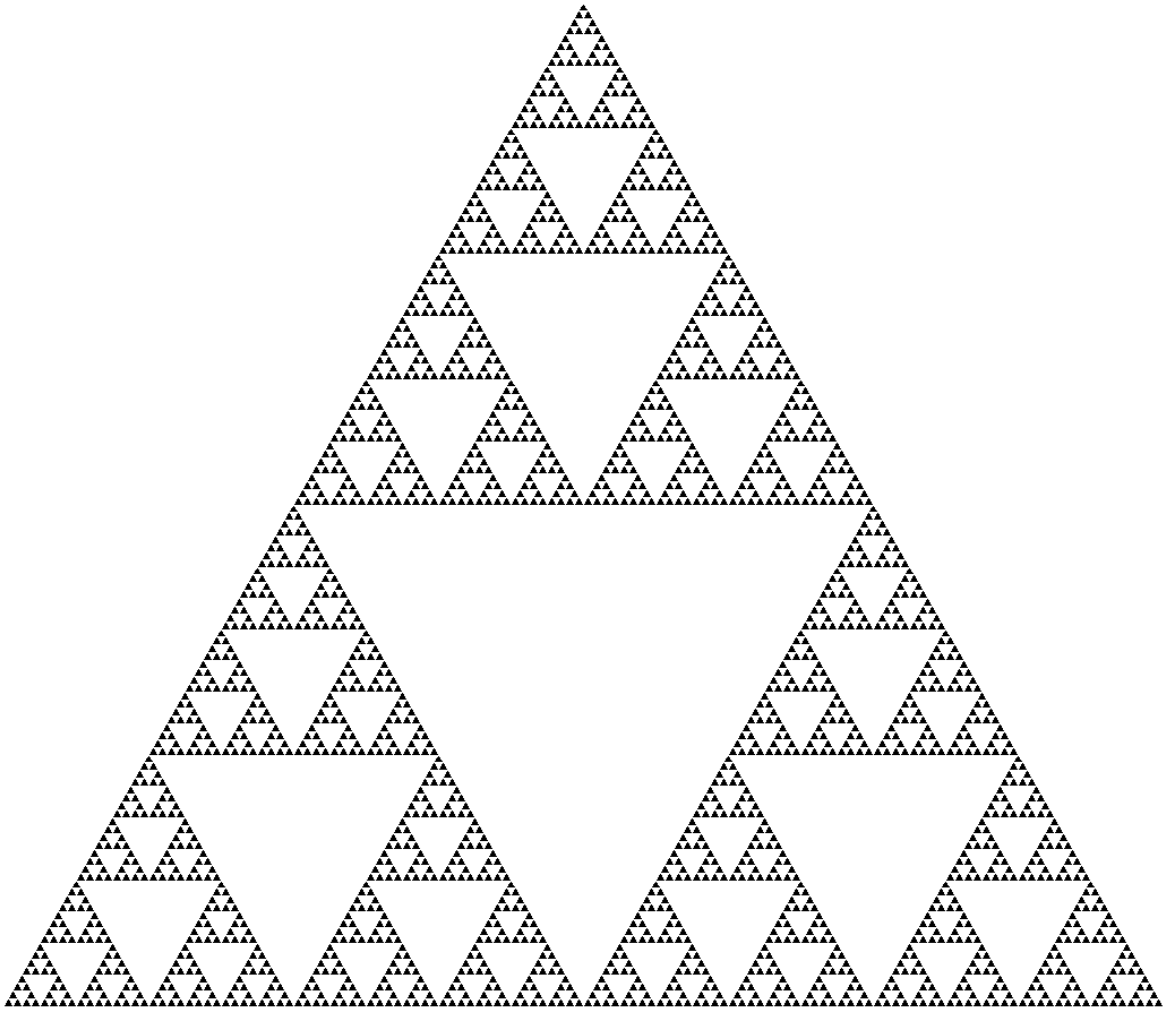
Tháp Hà Nội – Đệ qui vs Không đệ qui

- Đệ qui:
  - 11 dòng code.
  - https://www.hackerearth.com/blog/developers/tower-hanoi-recursion-game-algorithm-explained/

```
1  void tower(int a,char from,char aux,char to){
2      if(a==1){
3         cout<<"\t\tMove disc 1 from "<<from<<" to "<<to<<"\n";
4         return;
5      }
6      else{
7         tower(a-1,from,to,aux);
8         cout<<"\t\tMove disc "<<a<<" from "<<from<<" to "<<to<<"\n";
9         tower(a-1,aux,from,to);
10     }
11 }
```

- Không đệ qui:
  - 136 dòng code.
  - https://www.geeksforgeeks.org/iterative-tower-of-hanoi/

Fractal

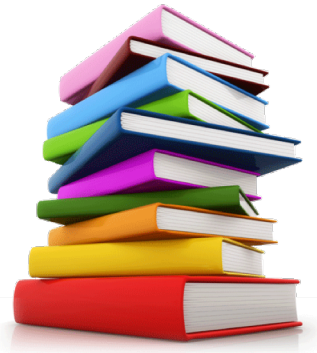


Sierpinski Triangle

Call stack

Stack

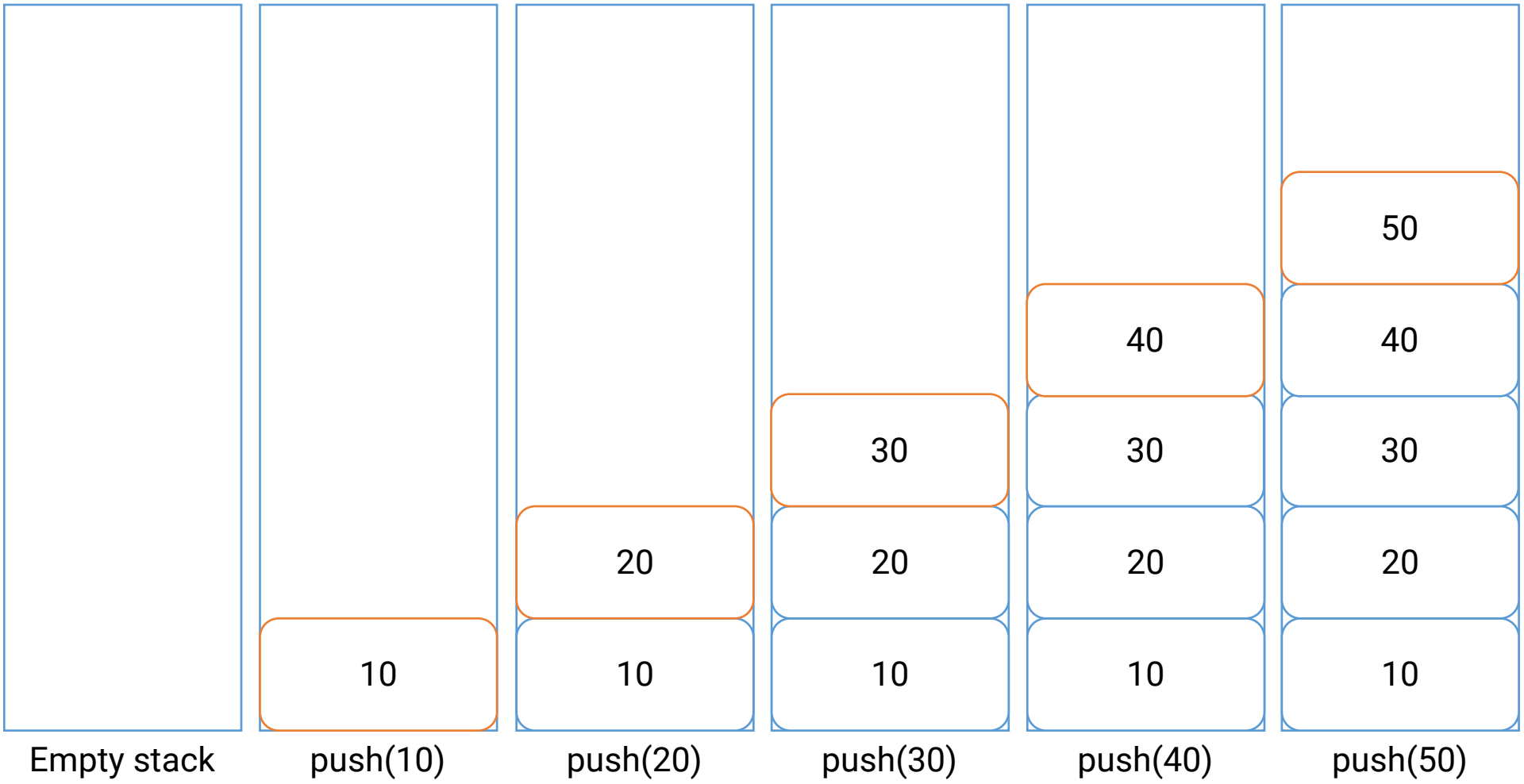
- Stack là cấu trúc dữ liệu, chứa danh sách phần tử, hoạt động theo cơ chế LIFO (**Last In First Out**).
  - Phần tử đưa vào sau cùng sẽ lấy ra đầu tiên.



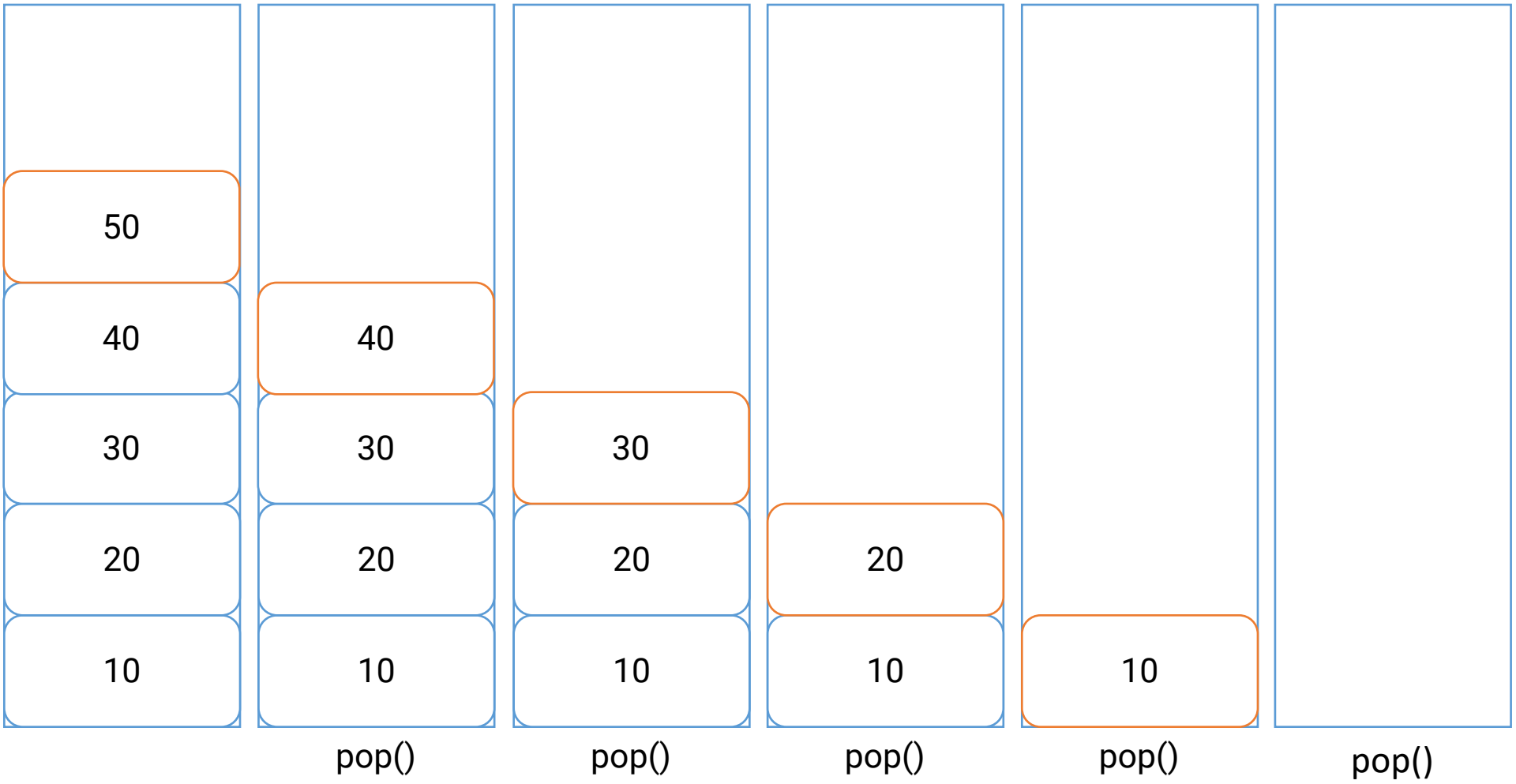
Stack vs Array

- Stack và array đều có thể chứa danh sách phần tử.
- Array: có thể truy cập phần tử ở vị trí **bất kì**, sử dụng index. VD: a[0], a[n-1], a[2]…
- Stack: chỉ có thể lấy **phần tử ở đầu**.

push()


10
20
10
30
20
10
40
30
20
10
50
40
30
20
10
Empty stack
push(10)
push(20)
push(30)
push(40)
push(50)

pop()

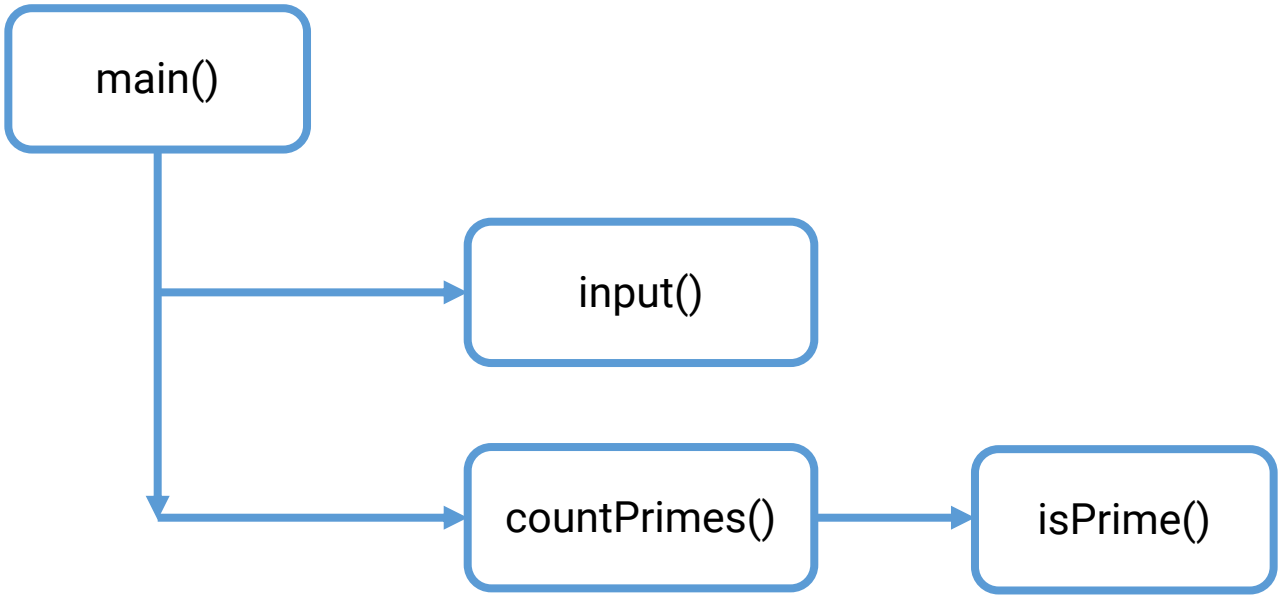


Call stack

- Khi ta chạy 1 chương trình, máy tính tạo 1 stack cho chương trình đó.
- Mỗi khi chương trình gọi 1 hàm, hàm đó được push vào stack.
- Khi 1 hàm kết thúc (return/exit), hàm đó được pop ra khỏi stack.
- Stack này được gọi là **call stack**.

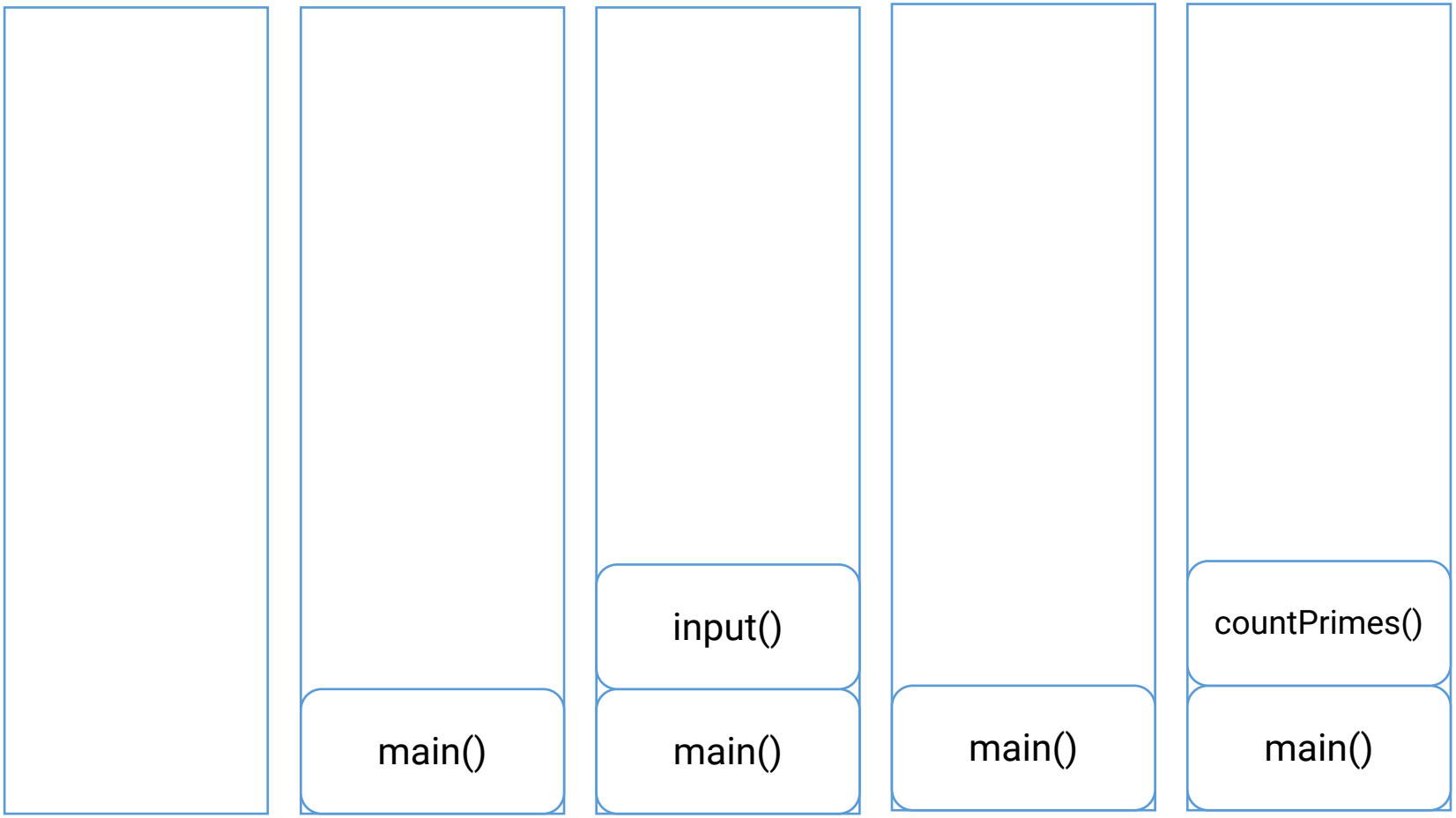
Ví dụ

- Viết chương trình nhập vào mảng số nguyên. Đếm số lượng số nguyên tố có trong mảng.



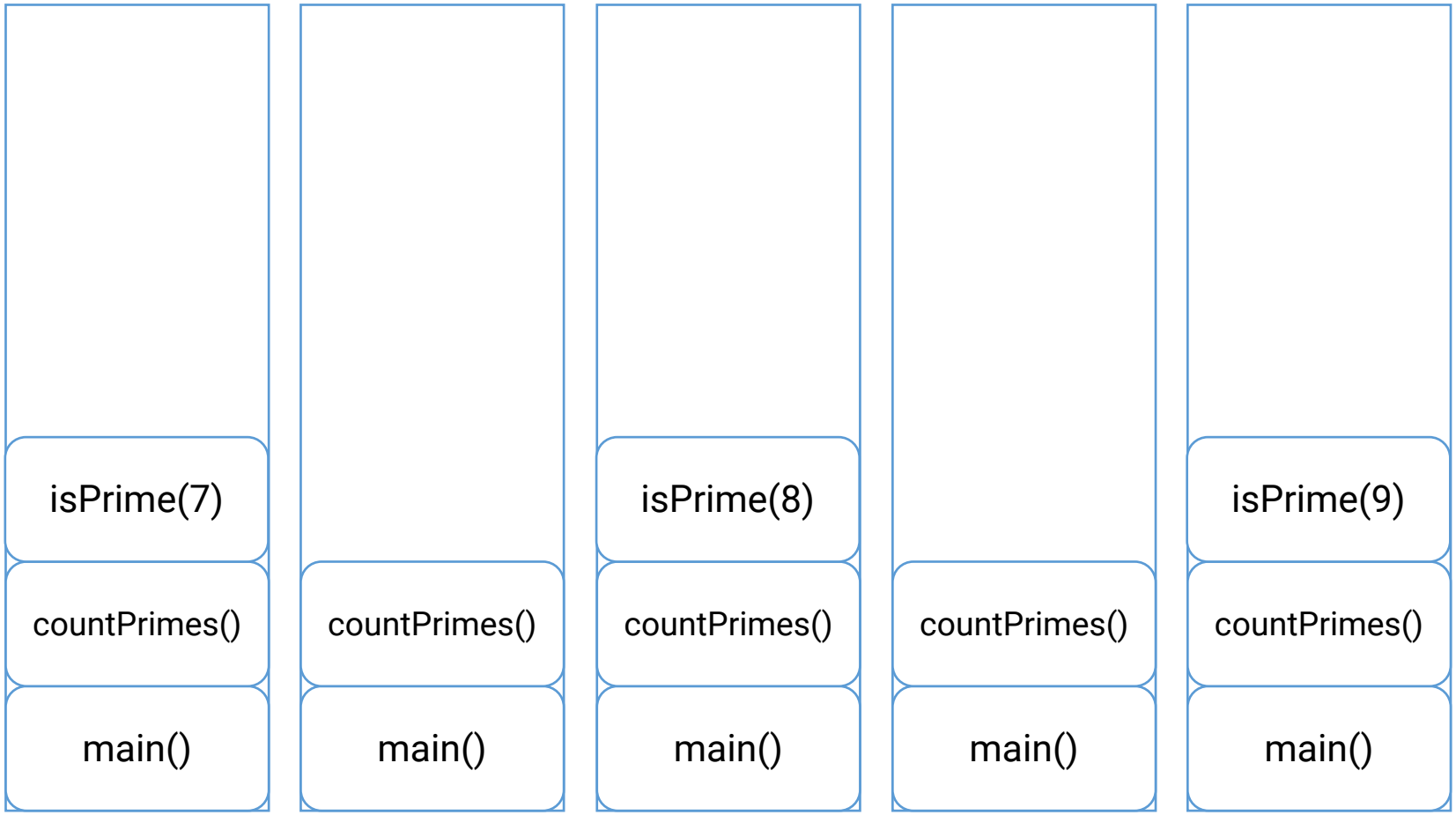
Ví dụ

7	8	9	2



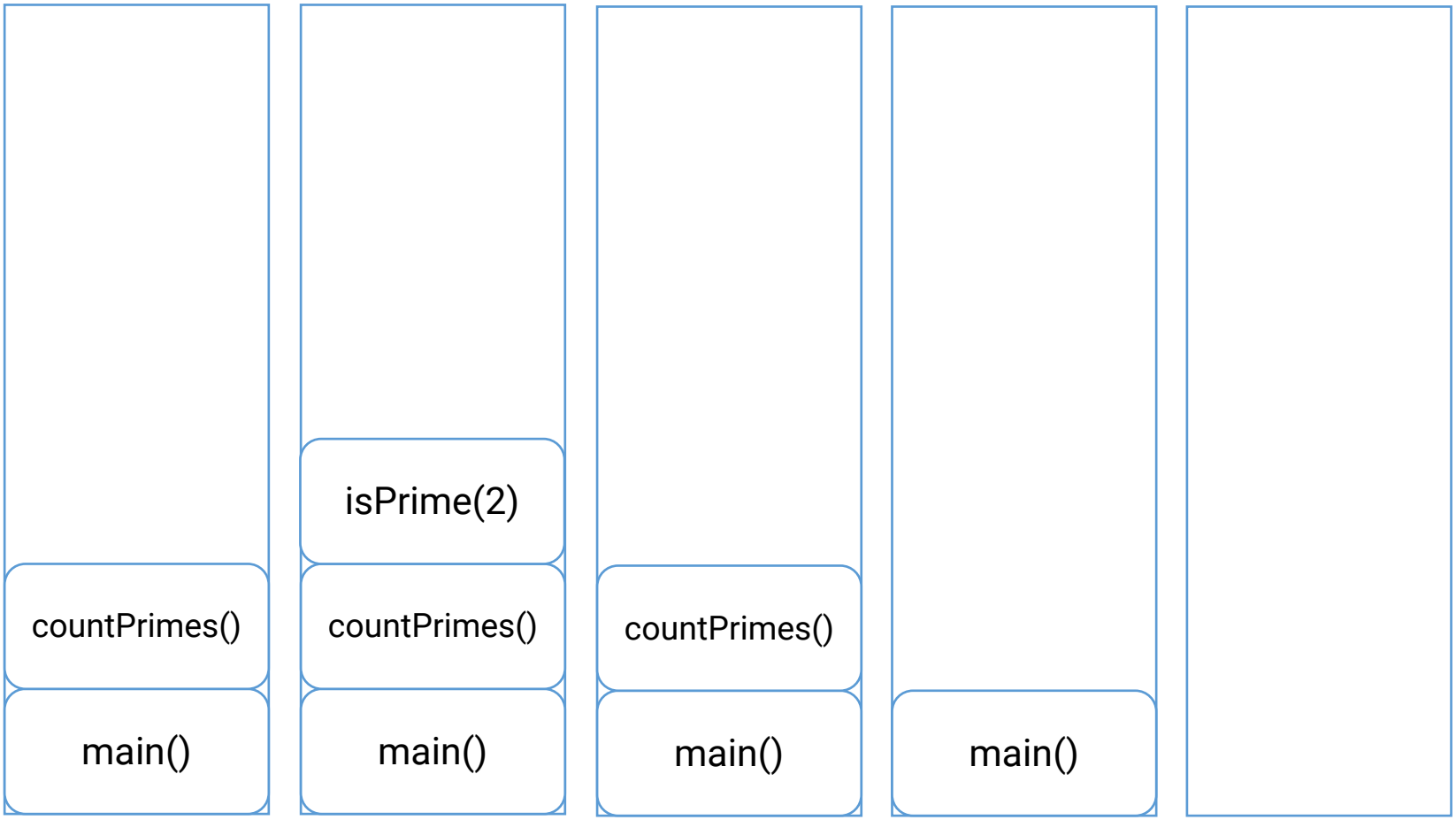
Ví dụ

7	8	9	2

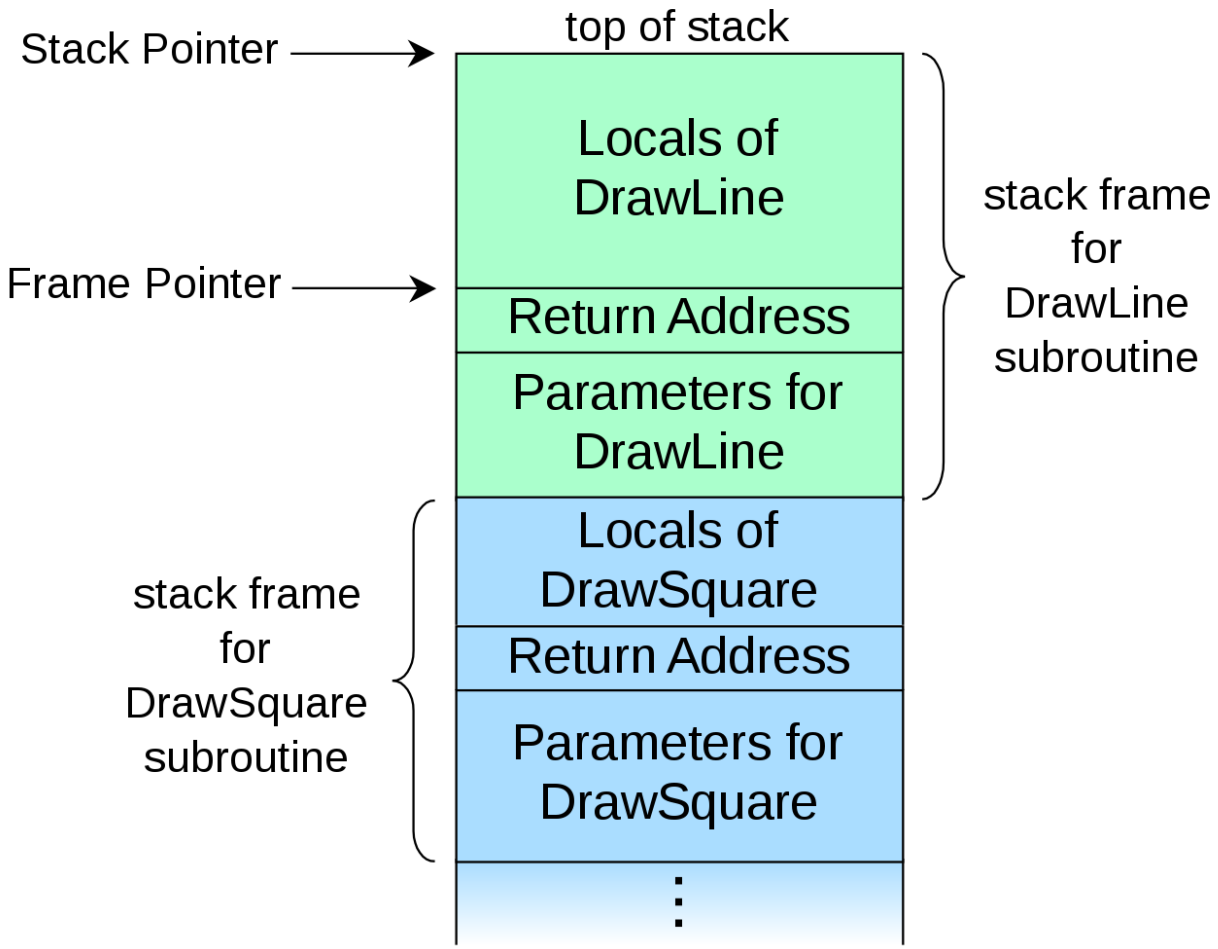

isPrime(7)
countPrimes()
main()
countPrimes()
main()
isPrime(8)
countPrimes()
main()
countPrimes()
main()
isPrime(9)
countPrimes()
main()

Ví dụ

7	8	9	2


countPrimes()
main()
isPrime(2)
countPrimes()
main()
countPrimes()
main()
main()

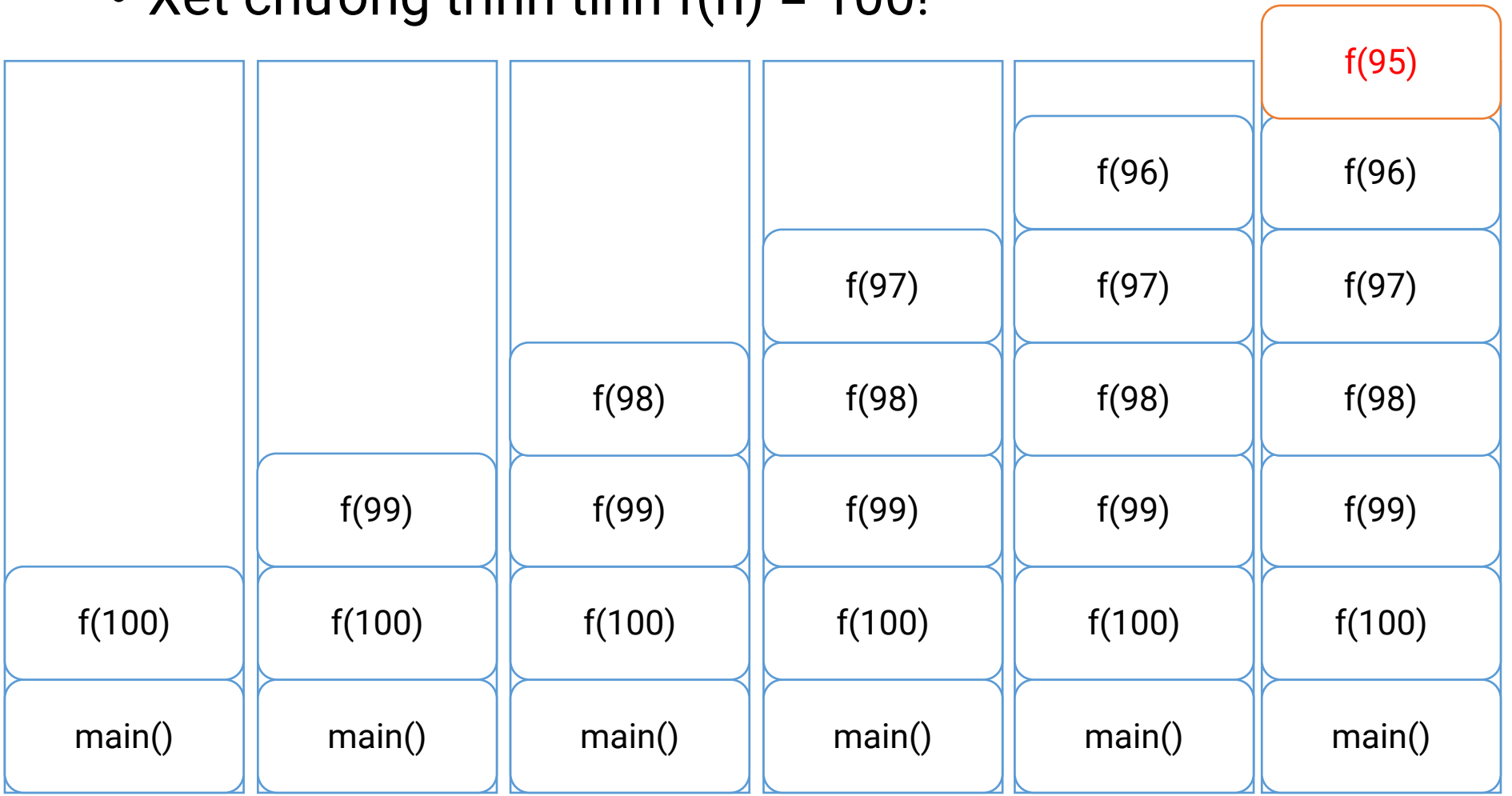
Call stack in detail



https://en.wikipedia.org/wiki/Call_stack

Call stack & recursion

- Xét chương trình tính f(n) = 100!



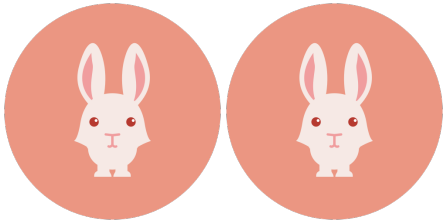
Khử đệ qui

- Sử dụng vòng lặp.
- Sử dụng stack.

Pair of rabbits

- Mỗi cặp thỏ bao gồm 1 con đực và 1 con cái.
- Khi 1 cặp thỏ được 2 tháng tuổi, chúng bắt đầu sinh 1 cặp thỏ mới.
- Mỗi tháng sau đó, cặp thỏ này, lại sinh ra 1 cặp thỏ mới.
- Các con thỏ không chết.
- Giả sử, ở tháng 1 ta có 1 cặp thỏ mới sinh.
- Hỏi tháng thứ n ta có bao nhiêu cặp thỏ?

Pair of rabbits



Số cặp thỏ tháng này	=	Số cặp thỏ tháng trước	+	Số cặp thỏ mới sinh ra
Số cặp thỏ tháng này	=	Số cặp thỏ tháng trước	+	Số cặp thỏ 2 tháng trước

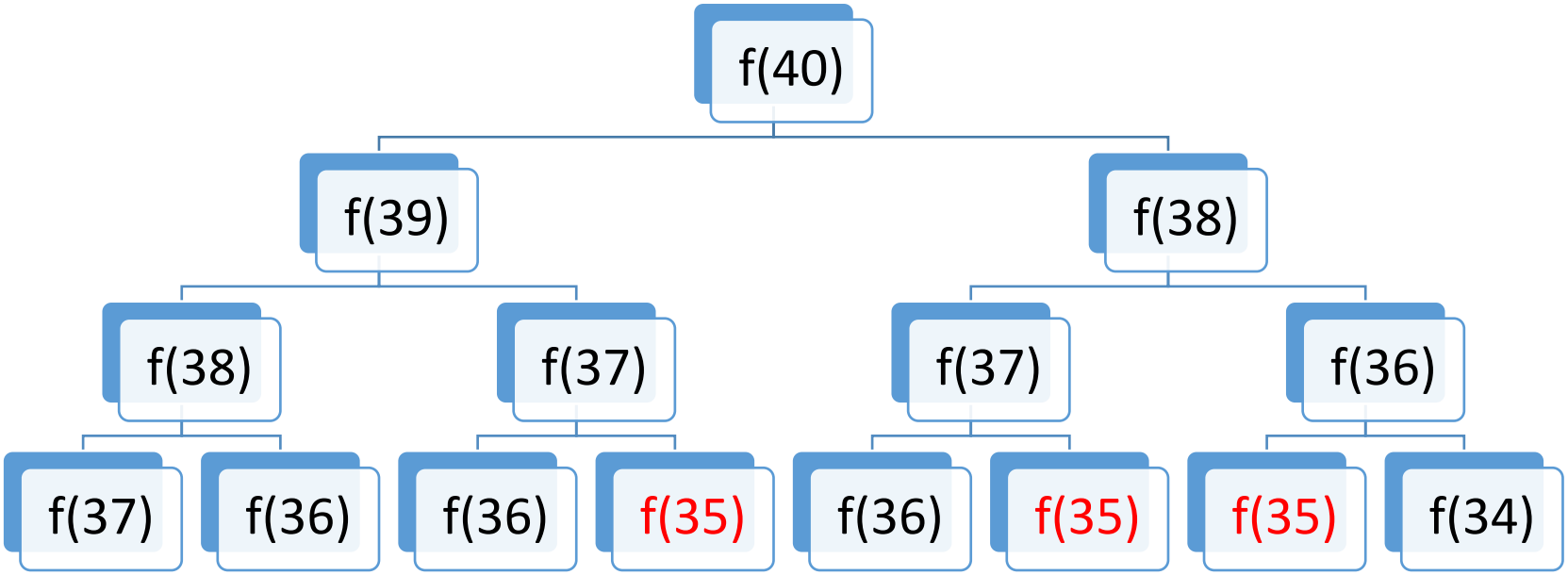
Fibonacci numbers

Inefficient case in recursion

- Tính giá trị F(n) trong dãy Fibonacci.

$$F(n) = \begin{cases} 1, nếu\ n = 0 \\ 1, nếu\ n = 1 \\ F(n-1) + F(n-2), nếu\ n > 1 \end{cases}$$

Inefficient case in recursion



Inefficient case in recursion

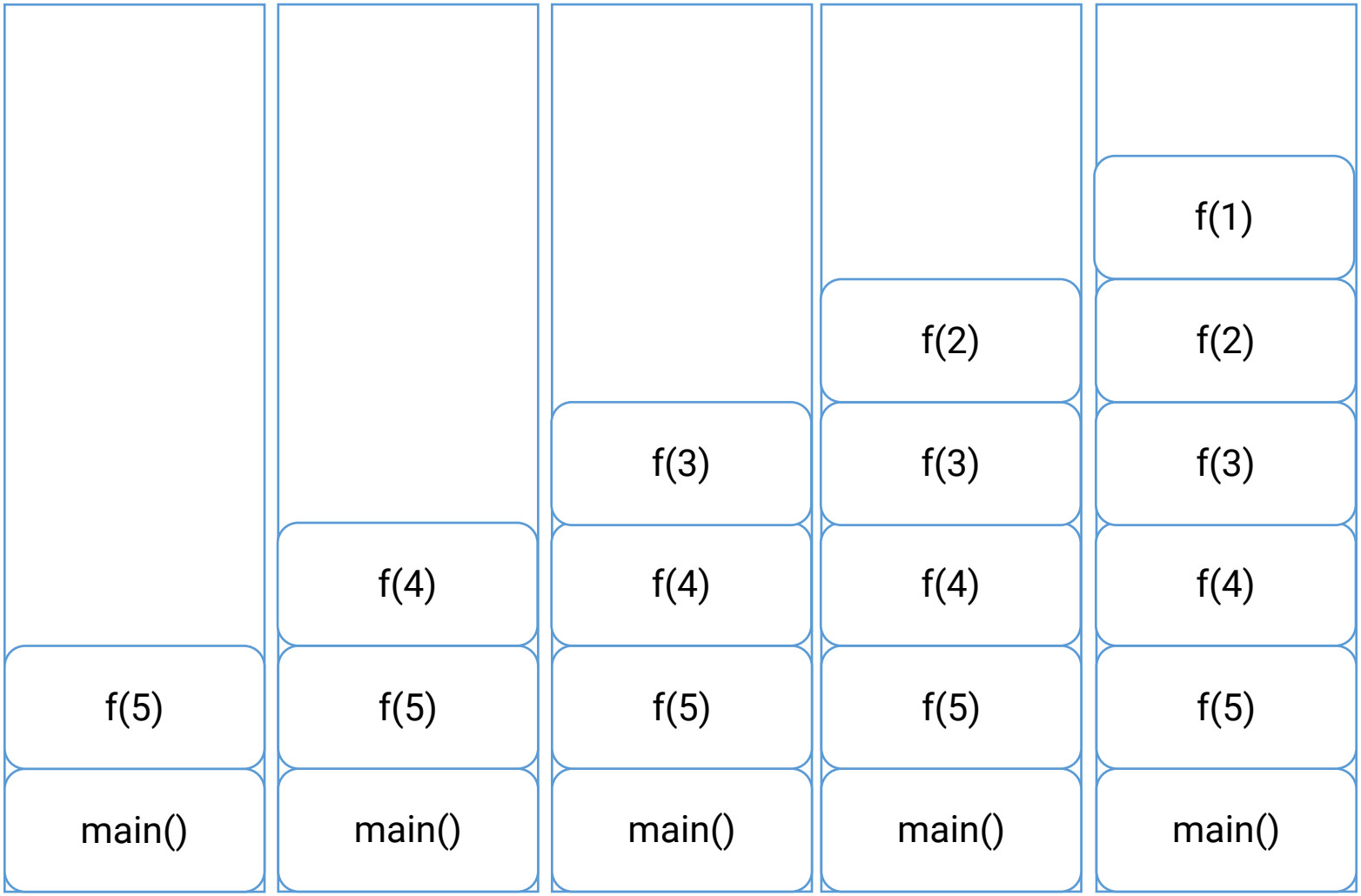
- f(39): 1 lần
- f(38): 2 lần
- f(37): 3 lần
- f(36): 5 lần
- f(35): 8 lần
- …
- f(0): 165.580.141 lần

Bài tập và gợi ý

BT1 – TÍNH ĐỆ QUY N!

- Viết chương trình tính đệ quy:
  - f(n) = n! = 1 * 2 * 3 * … * n

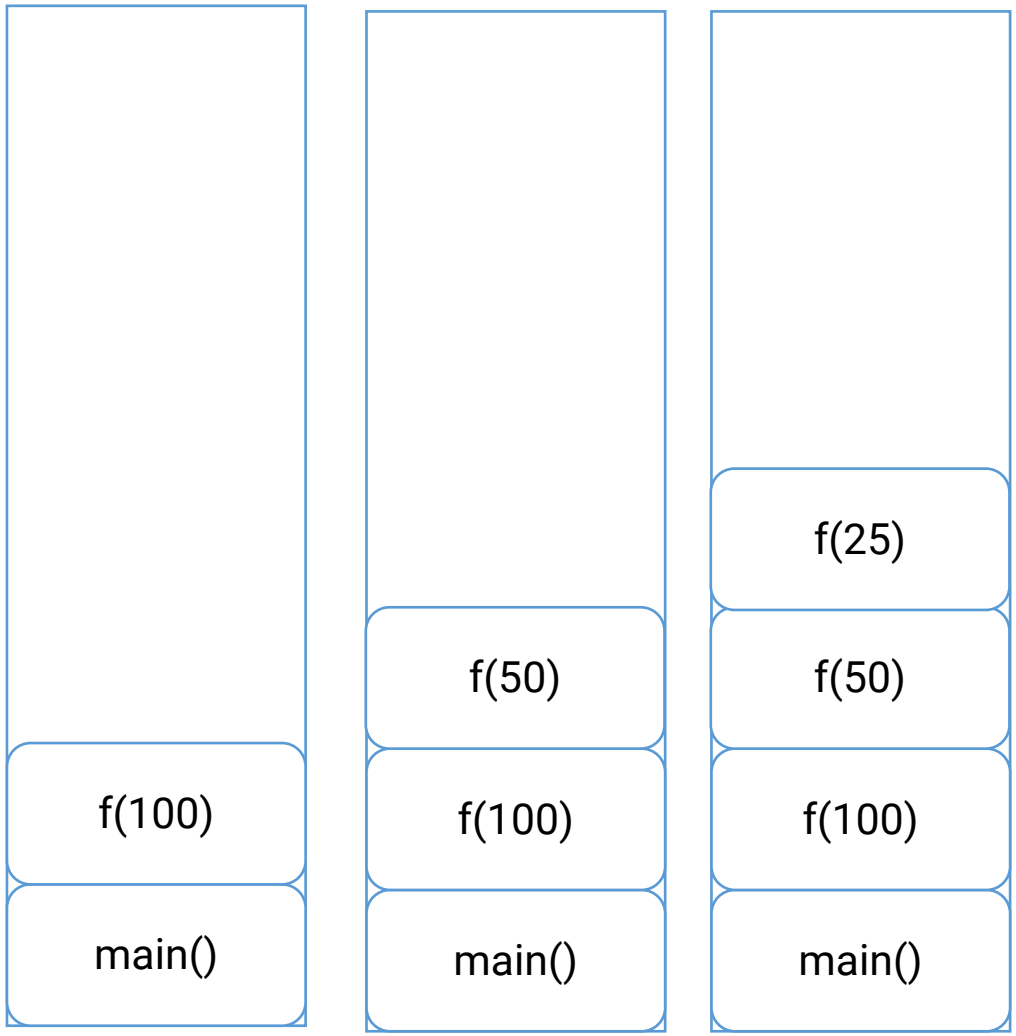
BT1 – Call stack



BT2 – ƯỚC LẺ LỚN NHẤT

- Viết đệ quy tìm ước số lẻ lớn nhất của số nguyên dương n.

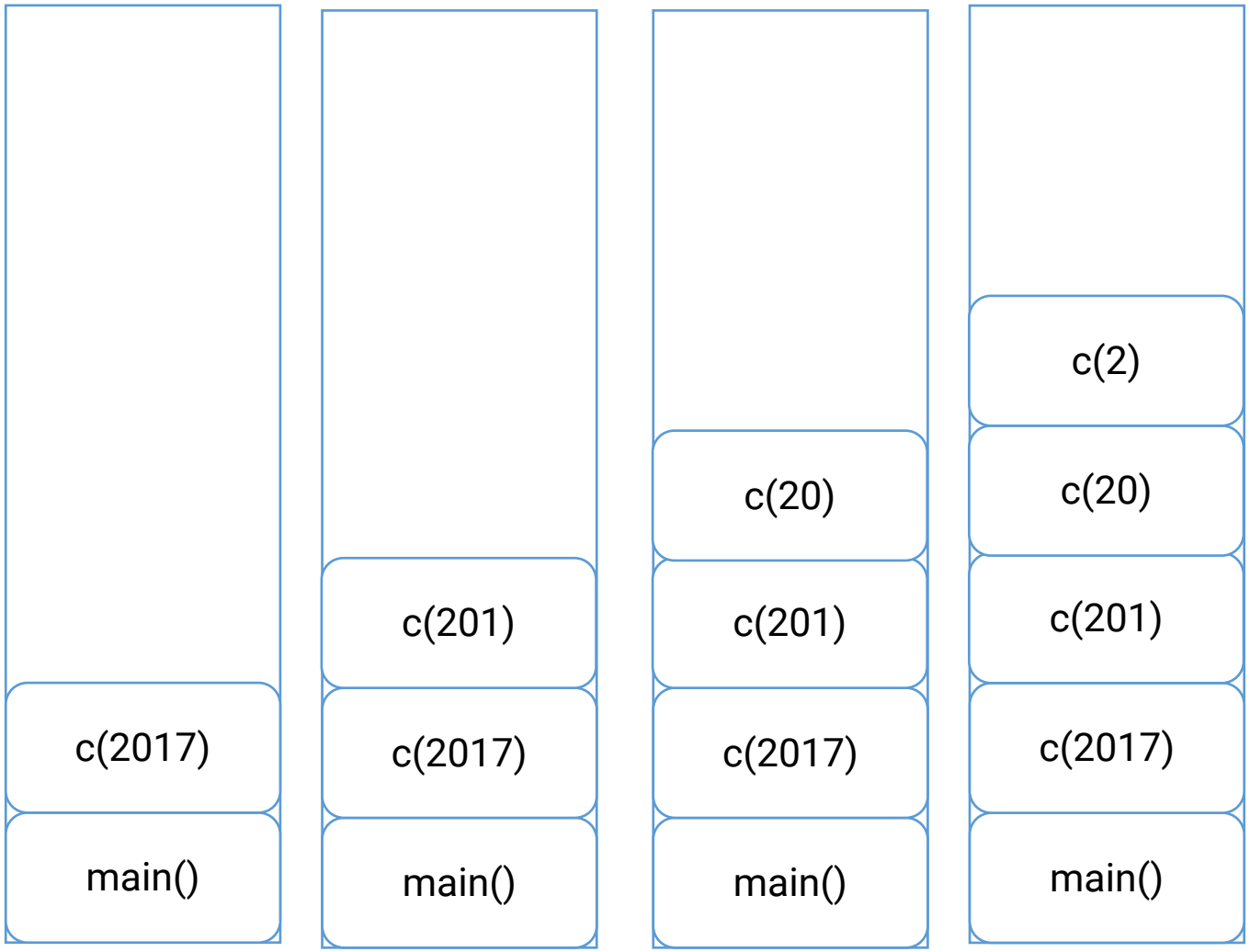
BT2 – Call stack



BT3 – ĐẾM SỐ LƯỢNG CHỮ SỐ

- Nhập số nguyên N. Viết chương trình đệ quy đếm số lượng chữ số của số nguyên N.

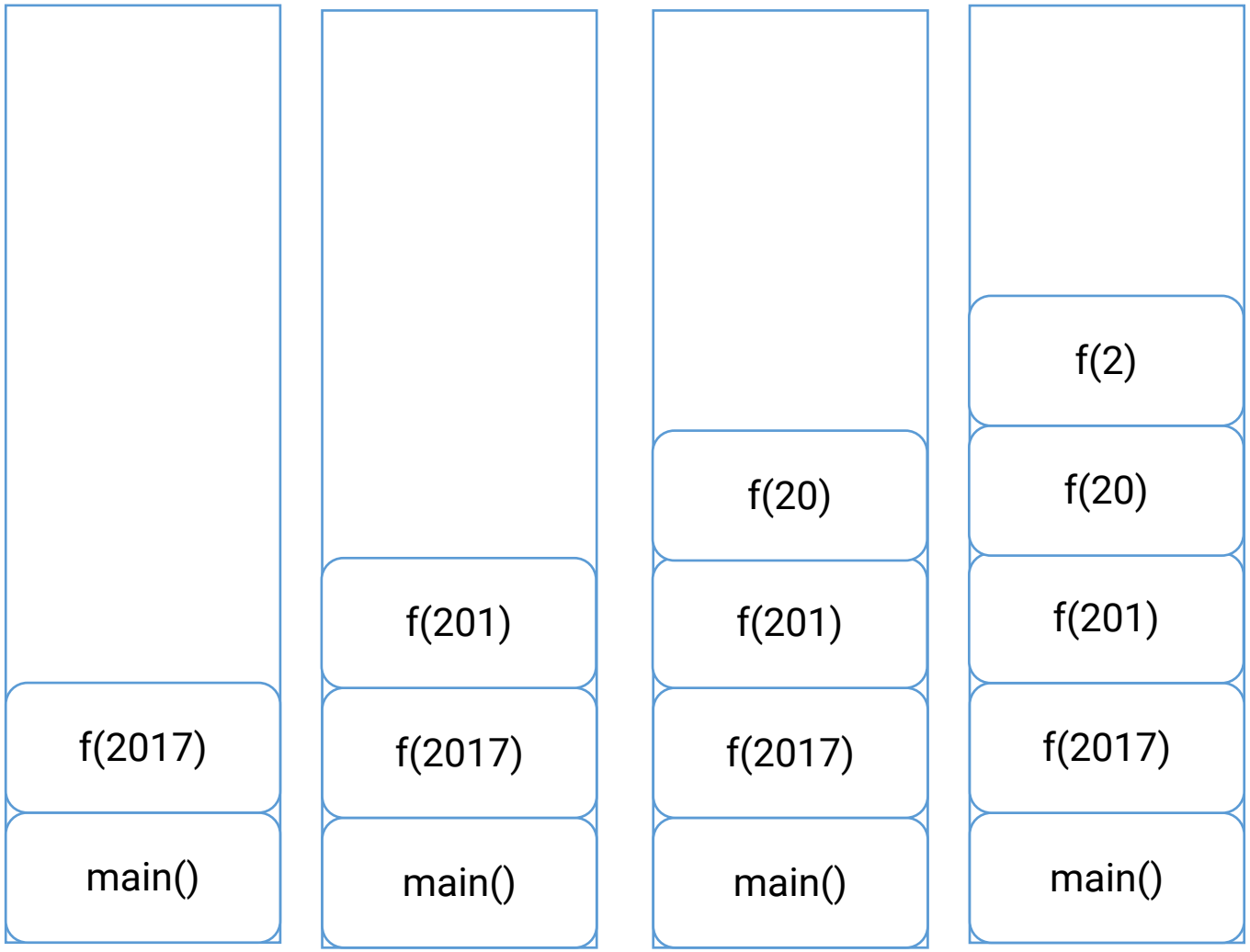
BT3 – Call stack



BT4 – TÌM CHỮ SỐ ĐẦU TIÊN

- Cho số nguyên n. Hãy viết chương trình đệ quy tìm chữ số đầu tiên của n.

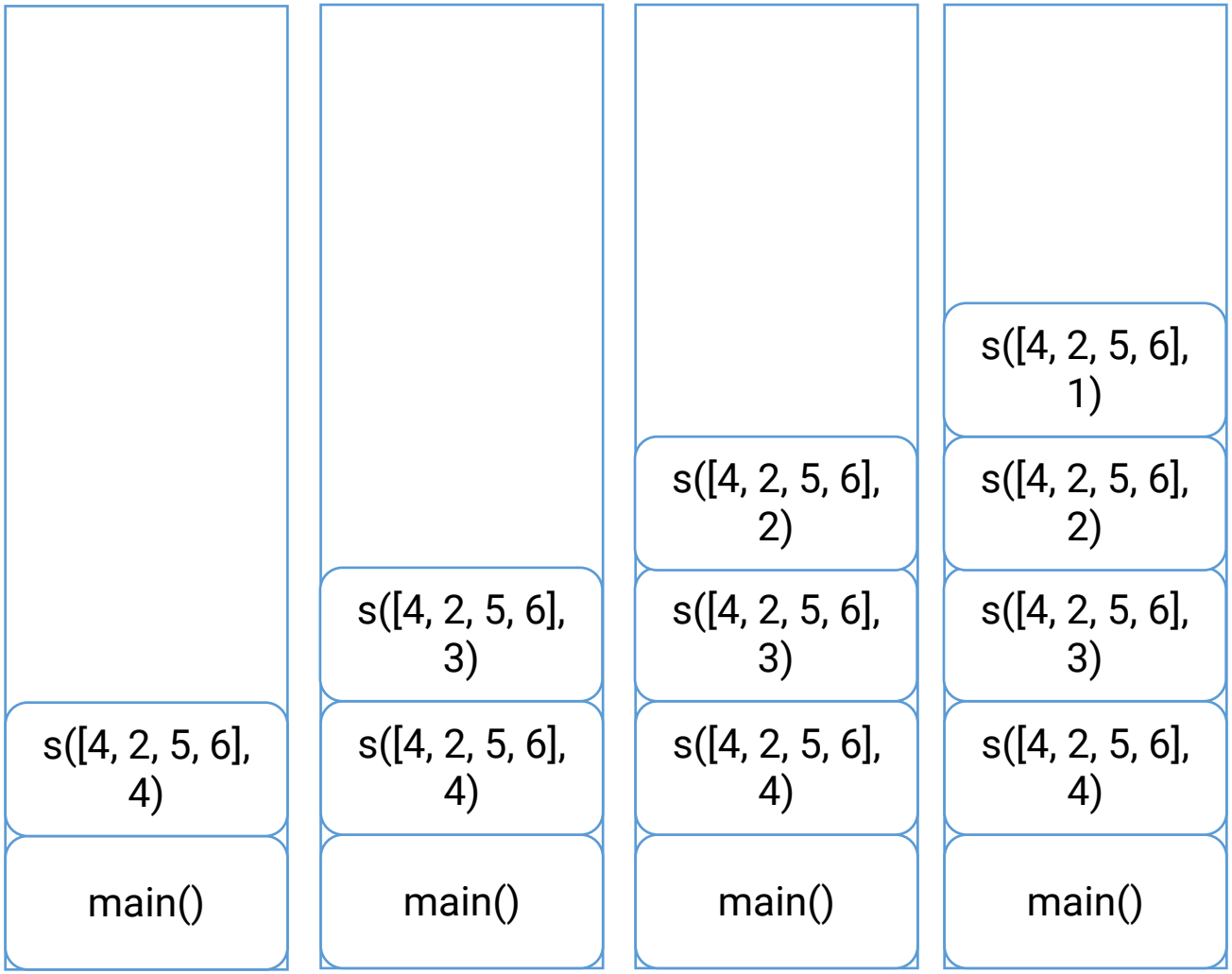
BT4 – Call stack



BT6 – TỔNG CHẴN

- Viết chương trình tính tổng các số chẵn trong mảng 1 chiều các số nguyên.

BT6 – Call stack



Hỏi đáp