G1962 - Programación

Grado en Ingeniería Civil
Práctica 4

Javier González Villa
(19 de diciembre de 2025)

Licencia: Creative Commons BY-NC-SA Internacional

Recursión

Una vez trabajados los conceptos previos de descomposición y abstracción, así como practicado con la implementación de funciones en Python, el siguiente paso es entender cómo estas funciones pueden llamarse a sí mismas con el propósito de optimizar algoritmos que, a priori, tienen una alta complejidad. Para ello, en este apartado se trabajará con el concepto de recursión y su aplicación mediante código Python.

La función exponencial:

Una vez entendido como plantear la estimación del valor de la función exponencial a través de la serie planteada en la primera parte de la práctica, el siguiente paso es volver a implementar la misma función, pero en este caso de manera recursiva. Para ello, se implementará primeramente de forma recursiva una función denominada **factorial_recursivo** la cual recibirá como argumento un número n y retornará el valor factorial de dicho número $n!$. Posteriormente, se implementará, pero esta vez de manera recursiva, la función **exponencial_recursivo** la cual recibe como argumentos el número x y la cantidad de sumandos, que por defecto son 25. Esta función hará uso de la función **factorial_recursivo** para el cálculo de los divisores en vez del bucle iterativo de la primera parte de la práctica.
Para comprobar el correcto funcionamiento de los códigos creados, se puede hacer uso de las funciones *math.exp(x)* y *math.factorial(n)* de la librería MATH.

Búsqueda binaria:

Otro problema muy común en diversos ámbitos, es también la búsqueda en listas ordenadas. Para ello, normalmente se suele iterar la lista completa hasta encontrar la posición del objeto buscado o utilizar algoritmos basados en técnicas de *Divide y Vencerás*, como puede ser la búsqueda binaria, presentando una implementación iterativa como la del siguiente código:

```
def busqueda_binaria_iterativa(a,L):
    menor = 0
    mayor = len(L)-1
    while(menor <= mayor):
        mitad = (menor+mayor)//2
        if (L[mitad] > a):
            mayor = mitad - 1
        elif (L[mitad] < a):
            menor = mitad + 1
```

```
        else:
            return mitad + 1
    return -1
```

Se pide, tras entender el funcionamiento del algoritmo en su versión iterativa, implementar la función **busqueda_binaria_recursiva(a,L)**, donde se le pasan como argumentos el objeto a buscar y la lista de objetos, y el algoritmo recursivo va subdividiendo la lista ordenada en sublistas que vuelve a pasar como argumento a la función implementada, realizando una estructura de pila de llamadas recursivas.

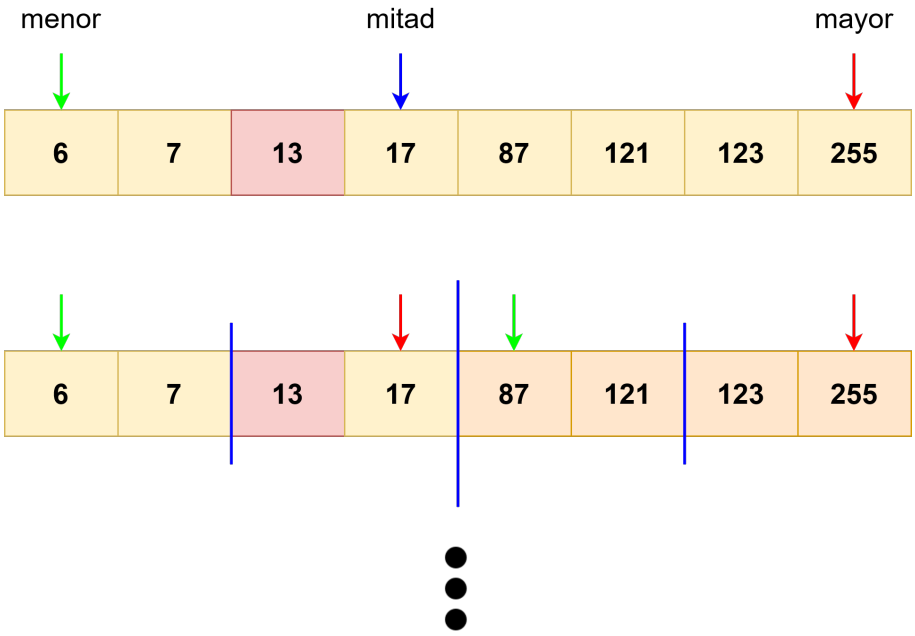

Figura 1: Ejemplo de búsqueda binaria con $a = 13$.

Tras implementar la función, genera una lista ordenada de menor a mayor que contenga 20 números cualesquiera y prueba ambos métodos para comprobar que funcionan de manera adecuada.