Ivan Fernandez Paris
Javier Fraile Iglesias
Pareja 12

Memoria Práctica 1

I-C. Cuestiones

1. **¿A qué función f se deberían ajustar los tiempos de multiplicación de la función de multiplicación de matrices? y a continuación dibujar tanto los tiempos como el ajuste calculado, y comentar los resultados.**

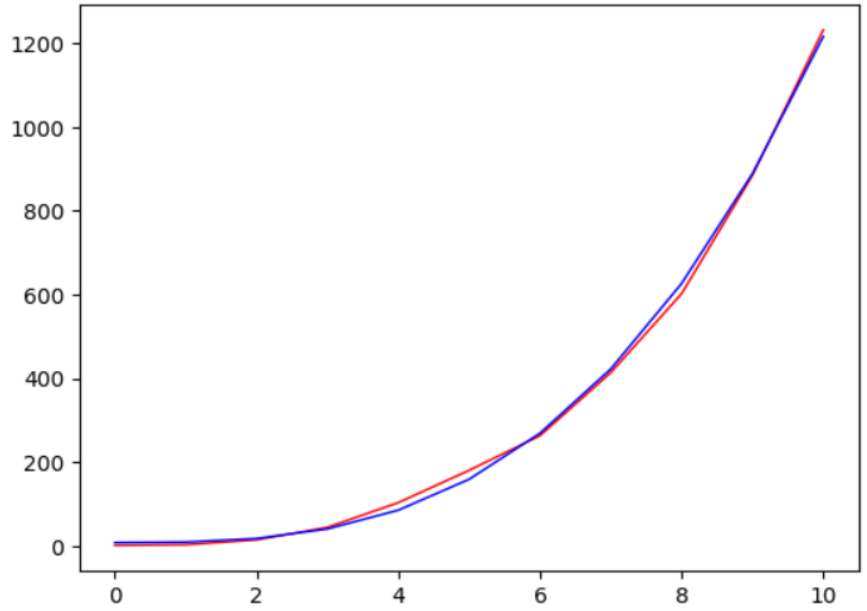
   Se ajusta a la función n^3, ya que el método de multiplicar matrices, se realiza con 3 bucles, con lo cual tiene un coste O(n^3).
   Una vez asignamos la función de ajuste y ejecutamos la función, normalizamos de vuelta el eje Y sobre el primer elemento, y ya está listo para graficar:

```
predict = fit_func_2_times(timings, func_2_fit)
eje_x = timings[:, 0]
eje_y = timings[:, 1] / timings[0, 1]


plt.plot(eje_x, eje_y, color="red", linewidth=1)
plt.plot(predict, color="blue", linewidth=1)

plt.show()
```

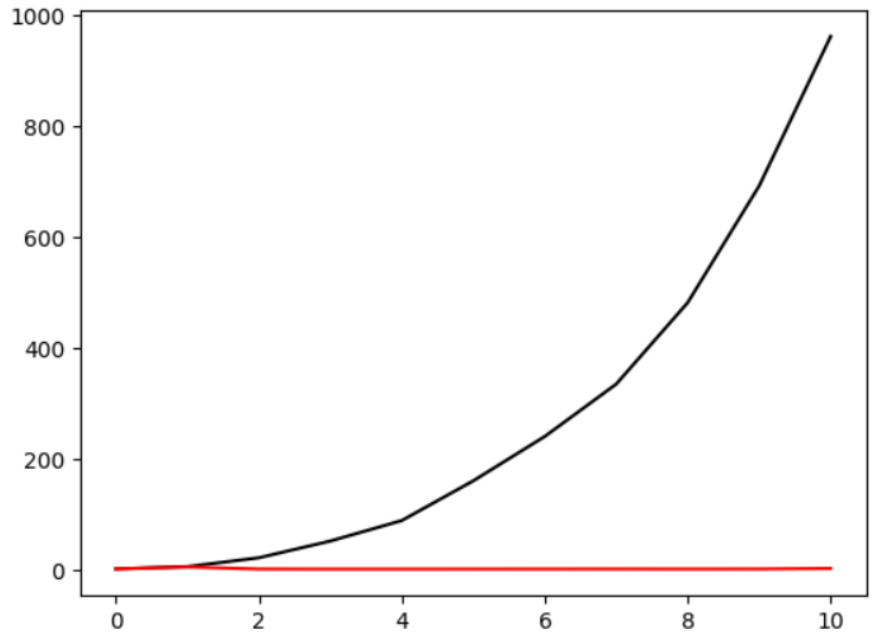
   En la siguiente gráfica se puede ver representado el ajuste sobre la función de multiplicar matrices:



   Siendo la línea azul la función de ajuste n^3 y la roja, la de multiplicar matrices.

2. **Calcular los tiempos de ejecución que se obtendrían usando la multiplicación de matrices a.dot(b) de Numpy y compararlos con los anteriores.**

   Si realizamos la comparación de nuestra función de multiplicar matrices (la cual tiene un comportamiento claramente exponencial, en función que aumenta el tamaño de la matriz), con la función “dot” de numpy, observamos que el comportamiento de esta segunda, es lineal.

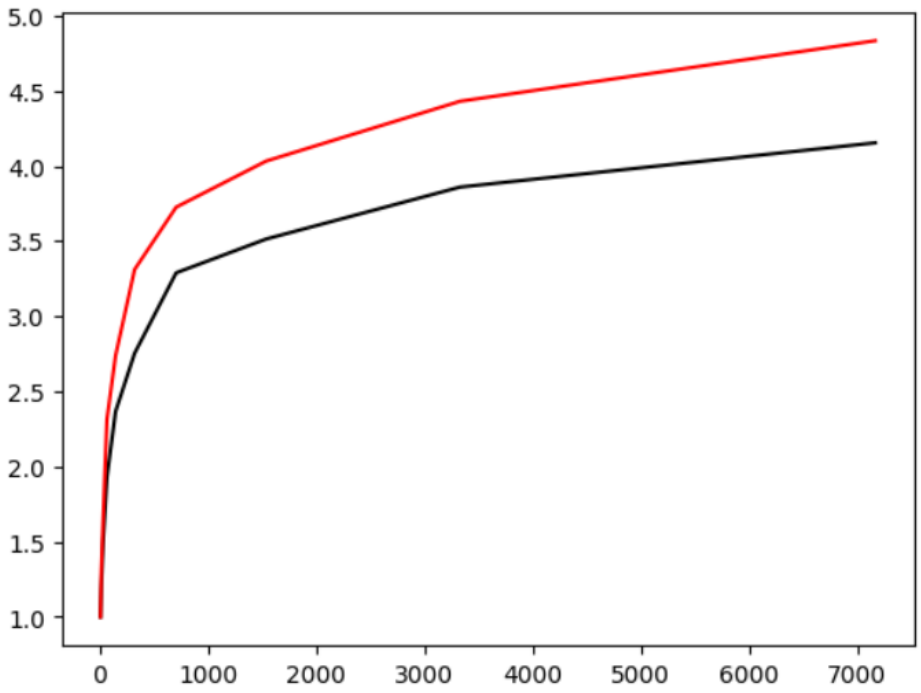


3. **Comparar los tiempos de ejecución de las versiones recursiva e iterativa de la búsqueda binaria en su caso más costoso y dibujarlos para unos tamaños de tabla adecuados. ¿Se puede encontrar alguna relación entre ellos?**

   Ahora se comparan los comportamientos de las funciones de búsqueda binaria tanto recursiva como iterativa. Hemos aumentado el número de iteraciones, y de llamadas que realizan a las funciones:

```
timings = %timeit -n 1000 -r 100
```

   A continuación, las graficamos, de manera que la roja es la versión recursiva, y la negra es la iterativa:



Aunque las dos variantes tienen un coste O(*log n*), se puede apreciar que la versión recursiva tarda un poco más, pero realmente es muy poco, porque la escala a la que está el eje X es del orden de diez elevado a menos seis.
Hemos llegado a la conclusión, que esa mínima diferencia, se debe a que el compilador de python, primero convierte la recursividad, a una variante iterativa.

II-D. Cuestiones

1. **Analizar visualmente los tiempos de ejecución de nuestra función de creación de min heaps. ¿A qué función se deberían ajustar dichos tiempos?**
   En primer lugar creamos las matrices de tamaño creciente y las desordenamos. Además, obtenemos el correspondiente tiempo para el tamaño en concreto de la matriz.

```
l_times = []

for i, size in enumerate(range(5, 15)):
    t = list(range(2**i * size))
    random.shuffle(t)
    key = t[-1]
    timings = %timeit -n 500 -r 10 -o -q create_min_heap(t)
    l_times.append([len(t), timings.best])
    min_heap = np.array(l_times)
```

   La función de crear un min heap se ajusta a *n*, ya que a pesar del crecimiento de tamaños de matriz exponencialmente, los tiempos se siguen creciendo de manera lineal, con coste O(*n*):

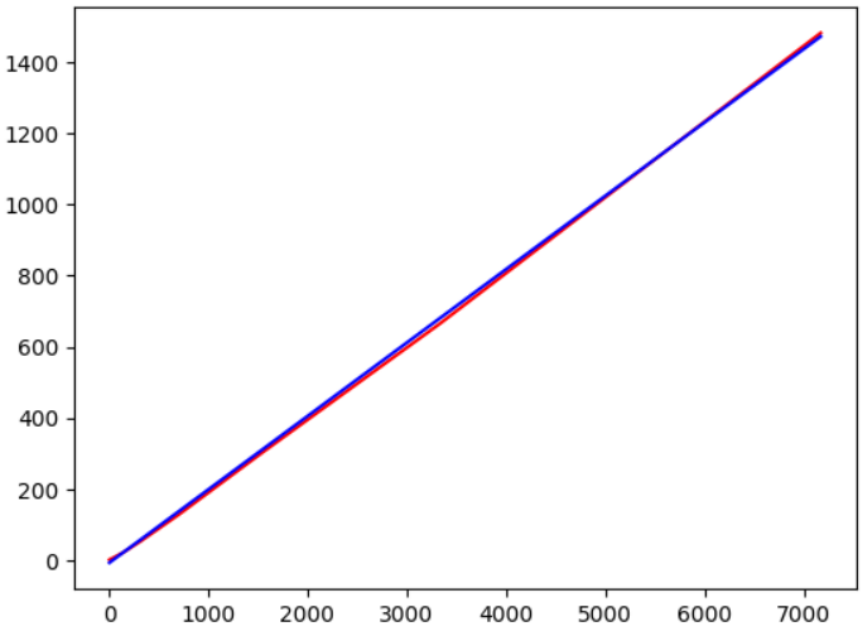
```
min_predict = fit_func_2_times(min_heap, func_2_fit)

eje_x = min_heap[:, 0]
eje_y = min_heap[:, 1] / min_heap[0, 1]


plt.plot(eje_x, eje_y, color="red")
plt.plot(eje_x, min_predict, color="blue")

plt.show()
```

Finalmente, mostramos la representación gráfica:



Donde la línea azul es la de la función de ajuste O(*n*), y la línea roja, son nuestros tiempos obtenidos. El eje X es el del tamaño de matriz, y el eje Y es el de los tiempos.

2. **Expresar en función de k y del tamaño del array cual debería ser el coste de nuestra función para el problema de selección.**

   El coste expresado en función del k y del tamaño del array (n) es:
   O(n * log k)

3. **Una ventaja de nuestra solución al problema de selección es que también nos da los primeros k elementos de una ordenación del array. Explicar por qué esto es así y como se obtendrían estos k elementos.**

   En nuestra solución lo que realizamos es crear un min heap de valores numéricos negados (el tamaño de dicho min heap es de k elementos), donde los elementos de menor tamaño (cuando su valor es positivo) son ahora los elementos de mayor tamaño en el heap (cuando su valor es negativo), lo que se asemeja a implementar un max heap.

En nuestra solución solo devolvemos el elemento que en un array ordenado estaría en la posición k, que en nuestro min heap de elementos creados es el elemento que se encontraría en la raíz. Si en vez de devolver el elemento situado en la posición k queremos los k elementos deberíamos devolver el nuevo min heap en vez de su raíz pero con los elementos no negados.

Deberíamos hacer esto cambios:

```
return h_aux[0] * (-1)
```
=>
```
return np.negative(h_aux)
```

Siendo h_aux el min heap de valores negativos.

4. **La forma habitual de obtener los dos menores elementos de un array es mediante un doble for donde primero se encuentra el menor elemento y luego el menor de la tabla restante. ¿Se podrían obtener esos dos elementos con un único for sobre el array? ¿Como?**

   Si se podría, más abajo dejaremos el código implementado con un ejemplo pero a continuación vamos a explicar la lógica. Puesto que queremos obtener los dos valores de menor valor, declaramos dos variables que serán las que almacenen los valores e inicialmente serán los números situados en las dos primeras posiciones, después recorremos el resto de la lista de modo que si es menor que el valor que hay en la primera variables aux (m1) se hará un swap de tal forma que en m2 estará el valor que había en m1 y ahora en m1 estará el elemento de la lista que acabamos de comparar con m1, puesto que en m1 deberá de estar el número de menor valor, si al compararlo con m1 no es menor, entonces pasará a compararse con m2, si es menor se cambia el valor a m2 y si no se pasa a iterar el siguiente elemento de la lista.

   Código implementado:

```
lista = [10 ,11, 1, 9, 0, 2, 3]

m1 = lista[0]
m2 = lista[1]

for l in lista[2:]:
    if l < m1:
        m2, m1 = m1, l
    elif l < m2:
        m2 = l


print(m1, m2)
```

```
0 1
```