Memoria
Algoritmia y Estructuras de Datos Avanzadas

Sergio Hidalgo y Miguel Ibáñez
Grupo:

I-C. Cuestiones 1. ¿A qué función f se deberían ajustar los tiempos de ejecución de la función de multiplicación de matrices? Suponiendo que el tiempo abstracto de ejecución de un algoritmo es f(t), usar el código inferior para ajustar valores de la forma a · f(t) + b a los tiempos reales de ejecución de dicho algoritmo, tiempos que habremos guardado en el array Numpy a_timings , para a continuación dibujar tanto los tiempos como el ajuste calculado por la función, y comentar los resultados.

```
from scipy.optimize import curve_fit

def tofit(x, a, b):
        #cambiar f por el código Python de la función de
        return a * f(x) + b
x = a_timings[: , 0]
#o (más "pythonico"?)
x = a_timings.T[0].T
y = a_timings[: , 1]
y = y / y[0] #normalizar timings
pars, _ = curve_fit(tofit, x, y)
```

El tiempo de ejecución de la multiplicación de matrices generalmente se ajusta a una función cuadrática, ya que la complejidad computacional de la multiplicación de matrices es O(n^3) en el peor de los casos. Entonces, la función f(t) podría ser del tipo f(t) = a * t^2 + b, donde 't' representa el tamaño de las matrices y 'a' y 'b' son coeficientes que deben ser ajustados.

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

Para medir los tiempos de ejecución de la multiplicación de matrices en NumPy, usamos la misma técnica que utilizamos en el primer ejercicio, utilizando la función %timeit. Aquí está el código para calcular y comparar los tiempos:

```
def matrix_multiplication(m_1: np.ndarray, m_2: np.ndarray) ->
np.ndarray:
return np.dot(m_1, m_2)

l_timings_np = []
for i in range(10, 21):
dim = 10 + i**2
m1 = np.random.rand(dim, dim)
```

```
m2 = np.random.rand(dim, dim)
timings = %timeit -o -n 10 -r 5 -q matrix_multiplication(m1, m2)
l_timings_np.append([dim, timings.best])
```

3. ¿Qué clave resultaría en el caso más costoso de la búsqueda binaria? 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. ¿Qué relación encuentras entre ambos tiempos? Argumentar gráficamente dicha relación.

En una búsqueda binaria, el peor caso ocurre cuando la clave que estamos buscando no está en la lista o array. En ese caso, el algoritmo de búsqueda binaria iterará hasta que los índices de inicio y finalización se crucen, sin encontrar la clave. Por lo tanto, la clave que resultaría en el caso más costoso es una que no esté presente en la lista.

Para comparar los tiempos de ejecución de las versiones recursiva e iterativa de la búsqueda binaria en el peor caso se utiliza la misma técnica que utilizamos para medir los tiempos de ejecución de la multiplicación de matrices. Ejemplo:

```
for dim in range(10, 1001, 50): # Se puede poner cualquier valor en
range
lista = list(range(dim))
key = -1 # Una clave que no está en la lista
timings_rec = %timeit -o -n 10 -r 5 -q rec_bb(lista, 0, dim - 1, key)
timings_iter = %timeit -o -n 10 -r 5 -q bb(lista, 0, dim - 1, key)
l_timings_rec.append([dim, timings_rec.best])
l_timings_iter.append([dim, timings_iter.best])

# Estas funciones son para dibujar la grafica
dim_rec, tiempo_rec = zip(*l_timings_rec)
dim_iter, tiempo_iter = zip(*l_timings_iter)

plt.figure(figsize=(10, 6))
plt.plot(dim_rec, tiempo_rec, label='Recursiva')
plt.plot(dim_iter, tiempo_iter, label='Iterativa')
plt.xlabel('Tamaño de la lista')
plt.ylabel('Tiempo de ejecución (segundos)')
plt.legend()
plt.grid(True)
plt.show()
```

II-D. Cuestiones 1. Analizar visualmente los tiempos de ejecución de nuestra función de creación in place de min heaps. Establecer razonadamente a qué función se deberıan ajustar dichos tiempos.

2. Dar razonadamente cuál debería ser el coste de nuestra función de ordenación mediante Min Heaps en función del tamaño del array.

3. Analizar visualmente los tiempos de ejecución de nuestra función de ordenación mediante Min Heaps y comprobar que los mismos se ajustan a la función de coste del punto anterior.