Basic Python. Min Heaps

Algoritmos y Estructuras de Datos Avanzadas 2022-2023

Practica 1

Grupo 02:
Guillermo Martil-Coello Juárez
Daniel Varela Sánchez
10/10/22

I-C. Questions

1. ¿A qué función f se deberían ajustar los tiempos de multiplicación de la función de multiplicación de matrices? ´ Usar el código inferior para ajustar valores de la forma a· f(t)+b a los tiempos de ejecución de la función que hayamos guardado en el array Numpy timings, para a continuación dibujar tanto los tiempos como el ajuste calculado por la función, y comentar los resultados.

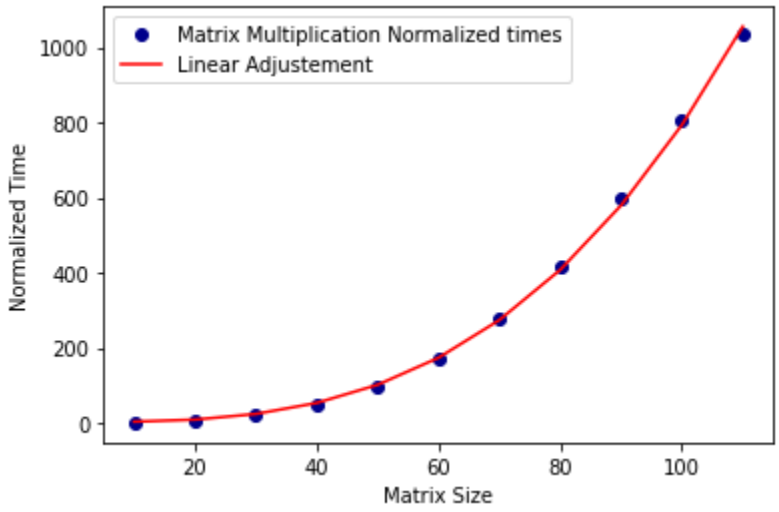
Since as we see in the image, in the code there are three nested loops, we can make the prediction that the times will fit the function n^3.

```
        for i in range(m1.shape[0]):
            for j in range(m2.shape[1]):
                for k in range(m2.shape[0]):
                    result[i][j] += m1[i][k] * m2[k][j]
```

We then introduce n^3 as the function to be fitted:

```
def func_2_fit(n):
   return (n**3)
```

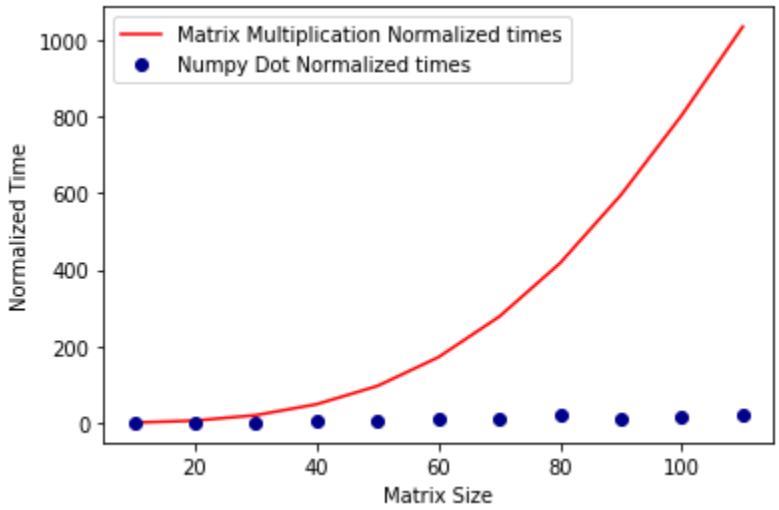
Then we draw a plot of the times and the adjustment calculated by the function:



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

We introduce in the script a new line that will calculate the execution times of the matrix multiplication using the a.dot(b) function and we compare with the times taken with our own matrix multiplication function.

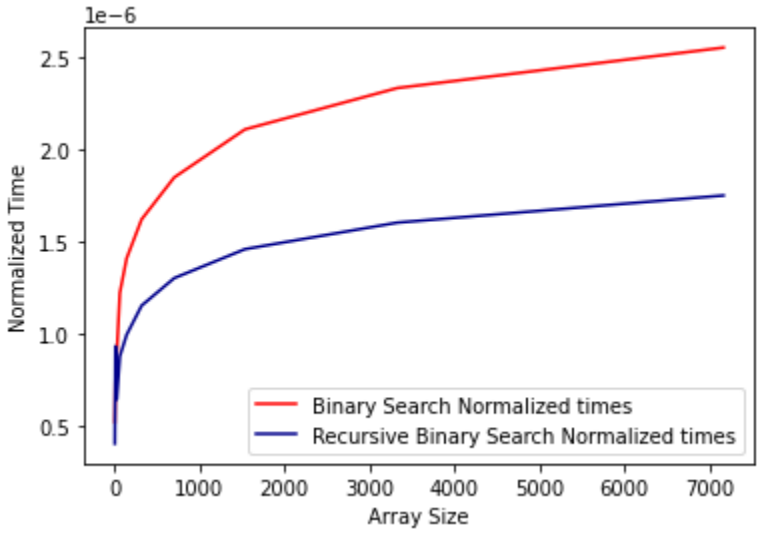
We obtain the following plot after normalizing:



We can observe that the dot function is a lot faster than our matrix_multiplication function. That is because the dot function is vectorized and its complexity is O(n).

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?

We make a recursive and an iterative version of the binary search. After that we plot the cost of each for the maximum cost case (which would be -1 since it is not on the table hence it will have to travel the full table). The plot would be the following:



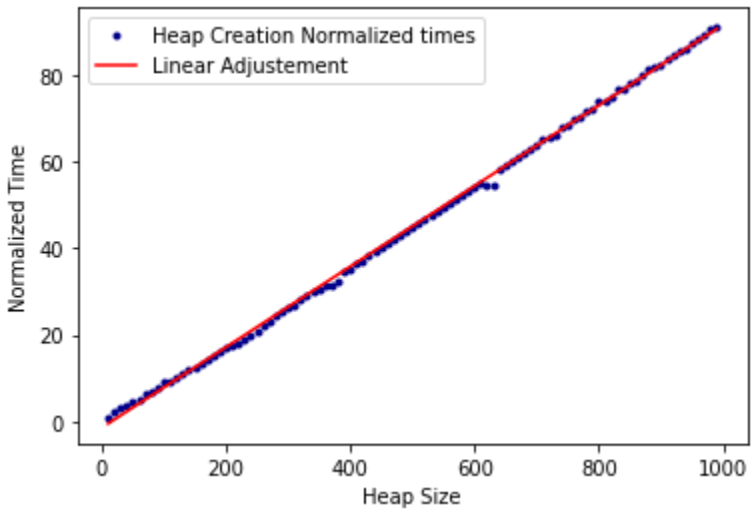
In this plot we can observe that the iterative version of the code is far more efficient than the recursive one. This is because each call to a function implies the execution of the CPU instructions necessary to

introduce the parameter and the return address in the stack, along with the instructions inside the function to access the parameter in the stack and return, hence being much slower than the iterative version of the code.

II-D. Questions

1. Analizar visualmente los tiempos de ejecución de nuestra función de creación de min heaps. ¿A qué función se deberían ajustar dichos tiempos?

A O(n).



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

The function used for the selection problem would iterate through all the elements of the array to find the solution. This will make it have a cost of O(n).

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 cómo se obtendrían estos k elementos.

For the implementation of our solution we use a min-heap of size k and multiply by -1 all the numbers in the array, then we go through all those numbers and if a number is greater (lower after being inverted) than the root (which should always be the greatest number on the tree), that number is discarded and it goes on to the next one. However if the number being analyzed is lower than the root it is incorporated in the tree and the old greatest number gets discarded. With this algorithm then, we end up with a min heap of the k lowest numbers of the array.

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?

Yes, by adapting the selection problem algorithm we could make a function that obtained the two lowest numbers with only one for. To do that we would create a min heap with the numbers inverted and do the same procedure as in the selection problem only this time sith k=2. After that we obtain a min heap with the two lowest values with only one for.

After changing the algorithm as said we would obtain the following code

```
    # Change the array to its negative
    h = -h
    # Create a min heap with the first 2 elements
    create_min_heap(h[:2])
    # For each element in the array, if it is larger than the root of the
    # heap, replace the root with the element and heapify
    for i in range(2, len(h)):
        if h[i] > h[0]:
            h[0] = h[i]
            min_heapify(h[:2], 0)
    # Return the two elements of the heap
    return h
```