**Diseño y Análisis de Algoritmos – Memoria Práctica 3**

Pareja 05

Borja González Farías

Darío Adrián Hernández Barroso

**Cuestiones sobre la DFT**

**1) Con nuestras funciones podríamos ser al parecer capaces de multiplicar enteros con cualquier cantidad de dígitos. Esto nos lleva a la siguiente cuestion: ¿cuál es el tamaño máximo de un entero en Python 3.X? Para responderlo, investigar la implementación en Python 3.X de los enteros y describir brevemente los aspectos principales de la misma.**

En Python 2.X, los enteros normalmente tenían un tamaño máximo de 64 bits (9,223,372,036,854,775,807), y en el caso de excederse ese umbral, Python 2.X automáticamente usaba los *long ints*, los cuales estaban limitados únicamente por la memoria disponible. En Python 3.X, la limitación de 64 bits se ha eliminado y ahora todos los enteros de tipo *int* tienen todo el tamaño que tenga la memoria disponible.

**2) No nos hemos esforzado en optimizar el coste computacional de nuestra FFT. Un defecto más o menos obvio es que muy probablemente calculemos repetidamente senos y cosenos en todas las llamadas recursivas mientras que en realidad bastaría hacerlo una vez en la primera ejecucion recursiva. ¿Cómo implementarías esta posibilidad?**

**Cuestiones sobre QuickSelect**

**1) Argumentar que MergeSort ordena una tabla de 5 elementos en a lo sumo 8 comparaciones de clave. En realidad, en qselect\_5 solo queremos encontrar la mediana de una tabla de 5 elementos, pero no ordenarla. ¿Podríamos reducir así el numero de comparaciones de clave necesarias? En función del correspondiente número mínimo de cdcs, ¿cual sería el caso peor WQSelect(N) en comparaciones de clave?**

* 8 comparaciones de clave para MergeSort con 5 elementos: Supongamos que tenemos una lista desordenada L = [a1, a2, a3, a4, a5]. Primero, se subdivide ésta en sublistas atómicas.

L1 =[a1] L2 =[a2] L3 = [a3] L4 = [a4] L5 = [a5]

Se pueden comparar los primeros dos pares de listas (lo que implica las primeras 2 comparaciones de clave).

L1 = [a1, a2] L2 = [a3, a4] L3 = [a5]

A continuación se puede fusionar L2 con L3, y las comparaciones de clave de esta fusión y ordenación es 1 (tamaño de L3) + 2 (tamaño de L2) – 1 (el último elemento a fusionar se queda sin comparar), lo que da igual a otras 2 comparaciones de clave y las sublistas ordenadas resultantes son:

L1 = [a1, a2] L2 [a3, a4, a5]

Por último, fusionaremos y ordenaremos las sublistas restantes, y el número de comparaciones de clave se sacará con la fórmula anterior, 2 (tamaño de L1) + 3 (tamaño de L2) – 1 (último elemento a fusionar que no se compara), lo que da igual a 4 comparaciones de clave. De manera concluyente, se puede observar que la suma de todas las comparaciones de clave (en el peor de los casos) es de **8**.

* ¿Reducir número de comparaciones de clave con qselect\_5?:

Sí, ya que nos ahorramos comparaciones de clave al no ordenar la tabla y el split no las realiza (el coste medio de QuickSelect es , asumiendo P = 1 y U = N, nQSel(T, 1, N, k) = N − 1 + ν, siendo N – 1 el coste del método split y ‘v’ el número de comparaciones de clave recursivas, y en el mejor de los casos, si resulta que m (valor devuelto por split) es igual a k, v = 0.

**2) ¿Que tipo de crecimiento cabría esperar en el caso peor para los tiempos de ejecución de nuestra función qsort\_5? Intenta justificar experimental y analíticamente tu respuesta.**