Cuestiones

Cuestiones sobre la DFT

Añadir código Python y figuras cuando se juzgue necesario.

Cuestión 1:

Con nuestras funciones podríamos ser al parecer capaces de multiplicar enteros con cualquier cantidad de dígitos. Esto nos lleva a la siguiente cuestión: ¿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

El tamaño máximo de un entero en Python 3.X no está definido. En python 2.X el tamaño mínimo y máximo era de [-2^63, 2^63 – 1] (si se usa python sobre un entorno de 64b, siendo [-2^3, 2^31 -1] el máximo usando un entorno de 32b). Esto se encontraba gracias a unas constantes definidas por Python llamadas sys.maxint y sys.maxsize, habiendo sido eliminada de Python 3.X sys.maxint, quedando sys.maxsize.

Ejecutando sys.maxsize, se ve que el tamaño máximo es 2^63 – 1, la cual define el tamaño máximo de memoria de la plataforma (64b), definiendo así el tamaño máximo de estructuras de datos como strings o listas, siendo los enteros manejados por software.

Haciendo la siguiente prueba, efectivamente pudimos comprobar que no existe límite, por parte de python, en el tamaño de los enteros, siendo manejado el cómputo de enteros mediante software.

```
x = 100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
print(x**2)
```

```
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

Ya que en Python el manejo de memoria es automático, y al estar escrito sobre C (al menos la versión más utilizada, luego hay python corriendo sobre .NET, la máquina virtual de Java…), donde el manejo de memoria corre por parte del programador, había que adecuarlo para ello. Entonces la solución propuesta en Python 3.X fue la de unificar los tipos long int y short int, haciendo un nuevo tipo abstracto, integer, el cual cuando sobrepasa el tamaño limite de short int, en vez de lanzar una excepción de "OverflowError", retorna el long int correspondiente.

Bibliografía usada:

https://www.python.org/dev/peps/pep-0237/

https://stackoverflow.com/questions/7604966/maximum-and-minimum-values-for-ints

Cuestión 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 ejecución recursiva. ¿Como implementarías esta posibilidad?

Dado que el cálculo senos y cosenos viene dado por el cálculo del término exponencial en la letra e, bastaría con calcular al principio la mitad de valores de la “rueda” para los tamaños sucesivos de la lista (primero sobre el tamaño de la lista entera, después sobre la mitad, así sucesivamente hasta que el tamaño de la lista sea 1), espejar dicha lista para conseguir los valores negativos y guardarlas en una variable constante tipo lista, de manera que según qué tamaño, se tendría que consultar una sublista u otra.

Cuestiones sobre QuickSelect

Cuestión 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 número de comparaciones de clave necesarias? En función del correspondiente número mínimo de cdcs, ¿cuál sería el caso peor WQSelect(N) en comparaciones de clave?

En una tabla de 5 elementos por ejemplo [5, 7, 9, 3, 1] dividiríamos la tabla en dos conjuntos [5, 7] y [9, 3, 1], iremos dividiendo los subconjuntos hasta que darnos con conjuntos de un elemento. Cuando lleguemos a ese momento empezaremos a comparar las claves y colocarlas de forma que las subtablas queden ordenadas hasta llegar a la tabla total. Utilizando el ejemplo anterior compararemos el 5 con el 7 y meteremos el elemento menor en la subtabla, en este caso es el 5 y como no hay mas con quien comparar metemos todos los elementos de la otra tabla, y esa comparación de el 5 y el 7 es una comparación de claves. Lo anterior lo realizaremos para todo el resto de los conjuntos quedando asi:

5, 7, 9, 3, 1

5, 7 9, 3, 1

5 7 9 3, 1

3 1

1 comparación 3 y 1

5 7 9 1, 3

2 comparación 5, 7

3 comparación 1, 9

4 comparación 3, 9

5,7 1, 3, 9

5 comparación 5, 1

6 comparación 5, 3

7 comparación 5, 9

8 comparación 7,9

1, 3, 5, 7, 9

El peor caso de Qselect es O(N^2) utilizando esa forma de comparación de claves.

Cuestión 2:

¿Qué 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

El crecimiento en el caso peor que tendría sería como el de mergesort ya que en qsort_5 realizamos la mediana de las medianas y la usamos como pivote para ordenar la tabla de esta forma mejoramos el caso peor de qsort haciéndolo mas rápido, ya que con tablas de gran tamaño el caso pero de qsort llegaba a ser igual al caso peor de bubblesort.

El caso peor de qsort pasa de ser un crecimiento cuadrático O(N^2) a ser un crecimiento logarítmico O(N log N).