ISIS 1105 DISEÑO Y ANÁLISIS DE ALGORITMOS
PROYECTO
Jorge Andrés de los Rios (201011164)
Laura Camila Mojica López (201212513)

PROBLEMA C

Se busca un algoritmo que permita saber si una línea de texto es permutación de otra.

Algoritmo de solución:

1. Se le asigna un valor dependiendo de la letra que se deba procesar a cada posición.
2. Se crea un ArrayList para cada conjunto de valores.
3. Dependiendo del valor que tenga cada posición se agrega en su ArrayList correspondiente.
4. Se agrega de forma ascendente los valores de cada ArrayList al ArrayList ordenado.
5. Llama el método de ordenamiento teniendo como parámetro las líneas de entrada.
6. Convierte cada ArrayList a un String.
7. Compara las dos cadenas de texto.

Para solucionar el problema se escogió como posible entrada únicamente las letras minúsculas de 'a' hasta la 'z' y el ' '; se convertía a una variable numérica que iba desde el 0 hasta 28 dependiendo a qué carácter fuera para que después se crearan 28 listas en donde se agregaban las diferentes posiciones del arreglo de números dependiendo de su valor, posteriormente si la lista no era vacía se agregaban sus valores a la lista ya ordenada, esto permite que se deba recorrer el arreglo solo una vez para ser convertido en una línea de texto con los valores ya ordenados.

El algoritmo propuesto tiene como especificación:

$$\ulcorner ctx: line1\{' ', a \dots z\}: array\ of\ char, line2\ \{' ', a \dots z\}: array\ of\ char$$
$$\{Q: n, m > 0\}$$
$$S$$
$$\{R: permute = true\}$$

Donde en S se hacen llamadas a los métodos sortInvertido que devuelve un ArrayList<Integer> y a convertidor que devuelve int[].

Análisis de complejidad:

El objetivo es optimizar la complejidad temporal y espacial con respecto a la solución ingenua entregada, es por esto que para el algoritmo desarrollado la complejidad temporal se comporta de la siguiente manera:

P_i: Hace referencia a cada una de las partes dividida en el código del archivo ProblemaC_1.java

$$P1: O(n) \qquad P5: O(2)$$

$$P2: O(28) \qquad P6: O(n)$$

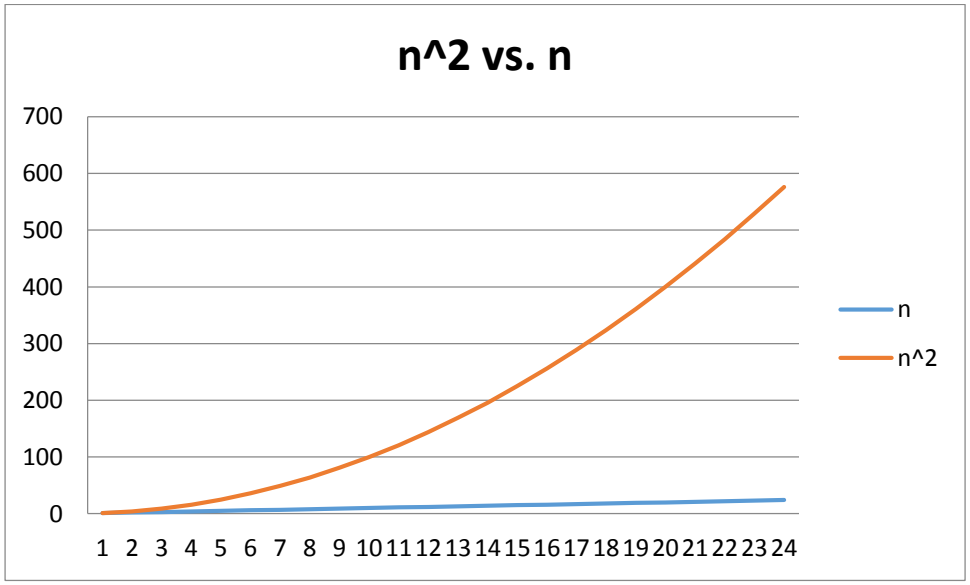
$$P3: O(n) \qquad P7: O(n)$$

$$P4: O(28) \qquad P8: O(1)$$

$$T(n) = O(4n) + O(59)$$

$$T(n) = O(n)$$

Comparando la solución ingenua para el problema y el algoritmo propuesto, podemos encontrar una mejora significativa sobre el algoritmo, ya que pasamos de tener una complejidad cuadrática a una complejidad lineal, donde el límite entre ambas funciones será cuando se tengan arreglos de tamaño 1, esto se puede ver evidenciado en la gráfica que se muestra a continuación.



Gráfica 1. Comparación complejidades

Se va a tener una complejidad espacial de orden O(n), ya que en el peor de los casos se van a guardar los n caracteres coincidientes en uno de los arreglos inicializados, haciendo así n asignaciones.

Comentarios finales:

Para mejorar la solución propuesta es necesario agregar los 228 caracteres que no fueron tomados en cuenta como variables de entrada que fueron ya especificadas, esto no tendría repercusión alguna en la complejidad del problema, ya que es el número de asignaciones las que se ven afectadas.

La complejidad del algoritmo propuesto $O(n)$ es lineal, esto significa que para una entrada del tamaño n, el tiempo de ejecución aumenta linealmente con el tamaño de la entrada, donde n es el número de caracteres de la línea de texto.

En cuanto a los caracteres Unicode según la solución propuesta sería aumentar el número de variables a comparar teniendo únicamente en cuenta las que no están en ASCII, pero si en Unicode. De esta manera la complejidad se mantendría ya que aumentarían las asignaciones que son $O(1)$, pero el tamaño del recorrido no es modificado porque sigue dependiendo del tamaño de la línea de texto.