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

PROBLEMA B

El problema consiste en encontrar la longitud mínima de un subarreglo a ordenar con el fin de ordenar un arreglo usando el mínimo número de operaciones posibles.

Algoritmo de solución:

Para lograrlo se ha desarrollado un algoritmo que se comporta de la siguiente manera:

1. Encuentra la posición del límite inferior del arreglo donde deja de estar ordenado ascendentemente.
2. Encuentra la posición del límite superior del arreglo donde deja de estar ordenado descendentemente.
3. Asigna el valor de la posición del límite superior e inferior como máximo valor y mínimo valor respectivamente.
4. Verifica que entre los límites ya establecidos no haya un número mayor o menor dependiendo de la variable dentro del arreglo, de no ser así estas variables cambian.
5. Halla la posición en la cual una variable es mayor al valor mínimo.
6. Halla la posición en la cual una variable es menor al valor máximo.
7. Para hallar el tamaño del subarreglo se resta la posición del límite superior al inferior.

El algoritmo propuesto tiene las siguientes especificaciones:

$$\ulcorner ctx: S: int, N: int, array[N-1]: array\ of\ int$$
$$\{Q: N > 0\}$$
$$\{R: \text{sminarray} = \text{limSuperior} - \text{limInferior} + 1\}$$

Donde sminarray es el tamaña del mínimo subarreglo obtenido por la diferencia de los límites, mas 1.

Se escogió implementar este algoritmo porque permite dividir el arreglo entre su límite superior e inferior, de esta manera se hacen recorridos solo hasta donde termina el máximo y mínimo valor organizado y así hallar el tamaño del arreglo mediante la diferencia de posiciones.

Análisis de complejidad:

En el momento de analizar la complejidad de la solución ingenua otorgada con el problema, se puede notar que el algoritmo debe realizar 3 ciclos para completar su ejecución, estos se van a encargar de tomar el primer (i) y segundo (j) elemento e ir avanzando hasta el final del arreglos, así como también ir comparando mínimos y máximos entre las distancias (k). Esto da como resultado una complejidad $O(n^3)$, debido a que los 3 ciclos deben realizarse para llegar a una solución posible.

El algoritmo propuesto fue diseñado con el fin de reducir esta complejidad, ya que llega al punto de obtener una complejidad lineal para el problema. Esto se hace mediante una comparación simultánea y sin recurrencias, se puede encontrar el subarreglo mínimo a ordenar obteniendo la diferencia entre los resultados obtenidos por la búsqueda.

P_i: Hace referencia a cada una de las partes dividida en el código del archivo ProblemaB_1.java

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

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

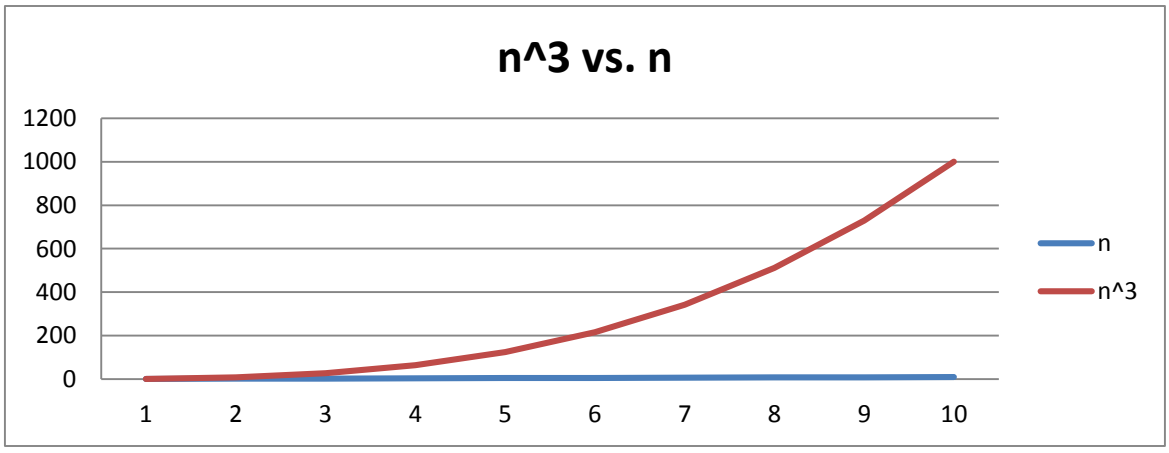
$$P3: O(2) \qquad P7: O(1)$$

$$P4: O(n)$$

$$T(n) = O(5n) + O(3)$$

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

La complejidad espacial del algoritmo también será de orden O(n), ya que se tiene que guardar n resultados para realizar las comparaciones de los límites.



Gráfica 1. Comparación complejidades

Analizando las complejidades de ambos algoritmos, podemos graficar ambas funciones sobre un plano para llegar a conclusiones en temas de la mejora y la eficiencia que hay de un algoritmo a otro. En la gráfica podemos ver que en el único caso en que ambos algoritmos tendrá la misma eficiencia es cuando se tenga un arreglo de tamaño 1, en caso diferente, será mejor el algoritmo propuesto con complejidad O(n).

Comentarios finales:

Se puede concluir que el algoritmo propuesto reduce drásticamente la complejidad de la solución ingenua que se proponía en el inicio del problema. Gracias a que se realiza solo una busca simultanea a lo largo del arreglo, y se realizan operación como calculos de minimos y maximos y asignaciones, la complejidad se mantiene lineal para todo el algoritmo, siendo esta complejidad temporal y espacial. El algoritmo funciona en casos extremos de manera correcta, siendo estos cuando el arreglo es de gran tamaño.