**Analisis de complejidad espacial y temporal de los algoritmos QuickSort y Burbuja**

Para el algoritmo QuickSort se tiene el siguiente cálculo de complejidad temporal:

Se tiene que:

n es el tamaño del arreglo de entrada, es decir, A.length

|  |  |  |  |
| --- | --- | --- | --- |
| QuickSort Sobre el Arreglo A | | | |
| Instrucción | Coste Temporal | Número de veces que se repite | |
| i = izq | C1 | 1 | |
| j = der | C2 | 1 | |
| aux = 0 | C3 | 1 | |
| pivote = A[izq] | C4 | 1 | |
| Mientras(i<j) | C5 | Mejor Caso 1 | Peor Caso 1 |
| Mientras (A[i]<=pivote && i<j) | C6 | (n/2) +1 | 2 |
| i++ | C7 | n/2 | 1 |
| Mientras (A[j]>pivote) | C8 | (n/2) +1 | n |
| j++ | C9 | n/2 | n-1 |
| Si(i<j) | C10 | 1 | 1 |
| aux= A[i] | C11 | 0 | 1 |
| A[i]=A[j] | C12 | 0 | 1 |
| A[j]=aux | C13 | 0 | 1 |
| A[izq]=A[j] | C14 | 1 | 1 |
| A[j]=pivote | C15 | 1 | 1 |
| Si(izq<j-1) | C16 | 1 | 1 |
| quicksort(A, izq,j-1) | C17 | T(n/2) | 0 |
| if(j+1 <der) | C18 | 1 | 1 |
| quicksort(A,j+1,der) | C19 | T(n/2) | T(n-1) |

Este algoritmo ordena la lista de números enteros y el peor caso es cuando el pivote termina en un extremo, haciendo analizar una lista de tamaño n-1, ejemplo, tenemos la lista (1,2,3,4,5,6), después de dividir menores a la izquierda y mayores a la derecha, y colocando el pivote en su lugar quedaría (1,2,3,4,5,6), haciendo que el próximo análisis sea de la lista (2,3,4,5,6). (Cuando la lista está ordenada)

En estos cálculos omitiremos los costos, y nos concentraremos en las cantidades.

Para el peor caso:

T(n) = T (n − 1) + (2n) + 13

Expandiendo:

T(n) = [ T (n − 2) + (2(n − 1)) + 13] + (2(n)) + 13 = T(n − 2) + 4(n) + 2(13) – 2

T(n) = [T (n − 3) + 2(n − 2) + 13] + 4(n) + 2(13) − 2 = T (n − 3) + 6(n) + 3(13) – 6

Y así sucesivamente hasta llegar a una fórmula general:

T(j) = T (j − (j− 1)) + j2(j) + j (13) − (j (j+ 1))

Reemplazando j por n, sería:

T(n) = T (1) + 2n^2 + n13 – n^2 –n

Si T (1) = C (constante), entonces:

T(n) = n^2 + 12n + C

Entonces se dice que el algoritmo tiene complejidad temporal O(n^2).

Para la complejidad espacial se tiene:

|  |  |  |  |
| --- | --- | --- | --- |
| Complejidad espacial Quick Sort | | | |
| Tipo | Nombre | Costo | Cantidad |
| ENTRADA | A | C1 | n |
| izq | C1 | 1 |
| der | C1 | 1 |
| AUXILIAR | i | C1 | 1 |
| j | C1 | 1 |
| aux | C1 | 1 |
| pivote | C1 | 1 |

El costo se mantiene en C1, porque todas las variables utilizadas son del mismo tipo, por lo cual tendrán un costo igual.

Ce = N +1 +1+ 1+ 1+ 1+ 1

Ce = n+6.

Por lo que decimos que la complejidad espacial del algoritmo es O(n).

Ahora bien, para el algoritmo burbuja, tenemos el siguiente análisis de complejidad temporal:

|  |  |  |  |
| --- | --- | --- | --- |
| Burbuja Sobre el Arreglo A | | | |
| Instrucción | Coste Temporal | Número de veces que se repite | |
| Para(i=1;i<A.length;i++) | C1 | n | |
| Para(j= A.length - 1;j >= i; j--) | C2 | n(n-1) | |
| Si(A[j-1]>A[j]) | C3 | Mejor Caso n-1 | Peor Caso n-1 |
| t = A[j-1] | C4 | 0 | n-2 |
| A[j-1] = A[j]; | C5 | 0 | n-2 |
| A[j] = t; | C6 | 0 | n-2 |

El peor caso para este algoritmo, es cuando está ordenado de manera inversa a la que se quiere ordenar y entra en cada una de las distintas verificaciones.

Sumando los valores para el peor caso:

T(n) = n + n(n-1) + n-1 + n-2 + n-2 + n-2

T(n) = n^2 + 4n -7

Por lo que decimos que el algoritmo tiene complejidad O(n^2).

Por último, analizando la complejidad espacial del algoritmo:

|  |  |  |  |
| --- | --- | --- | --- |
| Complejidad espacial Burbuja | | | |
| Tipo | Nombre | Costo | Cantidad |
| ENTRADA | A | C1 | n |
| AUXILIAR | j | C1 | 1 |
| i | C1 | 1 |
| t | C1 | 1 |

Tenemos que al igual que en el QuickSort, las variables que vamos a usar son de costo C1, dado que todas son del mismo tipo.

Ce = n + 1 + 1 + 1

Ce = n+3

Por lo que podemos decir que la complejidad espacial del algoritmo es O(n).