Informe del Laboratorio 1

Autores:

Jordi Lazo Florensa

DNI: 47694432E

Alejandro Clavera Poza

DNI: 49384205P

Grado en Ingeniería Informática

Estructura de datos

Universitat de Lleida

Bubble sort

Como podemos observar en la Tabla 1 el número de comparaciones en todos los casos son los mismas ya que estamos tratando el mismo array de N elementos en todos los casos. Por lo tanto, el número de swaps depende del grado de desorden del array. Con excepción del array invertido, ya que el número de comparaciones es igual al número de swaps, es decir, por cada comparación hay un swap.

Ordenaciones array [50]	Comparaciones	Swaps
Ordenado	1225	0
Invertido	1225	1225
Poco desordenado	1225	507
Muy desordenado	1225	637

Tabla 1 – Resultados análisis Bubble Sort

Después de los resultados obtenidos y el análisis de los bucles hemos llegado a la conclusión que el algoritmo es de $O(n^2)$. Es útil sólo para pequeños conjuntos de datos.

Insertion sort

Tras analizar los datos de la Tabla 2, observamos que el número de comparaciones va a depender del grado de desorden del array. Por eso el número de comparaciones es muy similar al número de swaps ya que prácticamente siempre se realizan tantas comparaciones como swaps a excepción del elemento que se está moviendo y que no se pueda mover más (encontrar un elemento más pequeño o que no tenga más elementos para comparar).
Cuando el array esta ordenado las comparaciones serán n-1 ya que el algoritmo no comparará la primera posición con sí misma.
En el array invertido el número de comparaciones es igual al número de swaps porque por cada elemento comparado se realiza un cambio

Ordenaciones array [50]	Comparaciones	Swaps
Ordenado	49	0
Invertido	1225	1225
Poco desordenado	553	507
Muy desordenado	682	637

Tabla 2 – Resultado análisis Insertion sort

Al igual que en el algoritmo bubble sort, insertion sort no es el algoritmo de ordenación más eficiente porque usa 2 bucles anidados para cambiar las posiciones en su lugar correcto. Es útil sólo para pequeños conjuntos de datos. Por tanto, el algoritmo es $O(n^2)$.

Selection sort

Como podemos observar en la Tabla 3, el número de comparaciones siempre será el mismo ya que siempre se recorrerá completamente toda la parte del array que no esté ordenado puesto que siempre está buscando el elemento más pequeño en cada sección.
Además, el número de swaps siempre será menor que el número de comparaciones debido a que sólo se realizará un swap por cada elemento no ordenado a excepción del último elemento que estará ordenado siempre que el resto este ordenado.
En este algoritmo como se puede observar, a mayor desorden se realizan menos swaps ya que hay más probabilidad de encontrarse antes el valor más pequeño dentro de la parte no ordenada del array.

Ordenaciones array [50]	Comparaciones	Swaps
Ordenado	1225	0
Invertido	1225	25
Poco desordenado	1225	47
Muy desordenado	1225	45

Tabla 3 – Resultado análisis Selection sort

No es el algoritmo de ordenación más eficiente porque usa varios bucles anidados para cambiar las posiciones en su lugar correcto. Es útil sólo para pequeños conjuntos de datos. Por tanto, el algoritmo es $O(n^2)$.

Quicksort

Tras analizar los datos de la Tabla 4, podemos observar como el número de comparaciones que realiza el algoritmo Quiksort son en la mayoría de casos menor, aunque a medida que se ampliando el tamaño del array la diferencia empieza a ser mas notable que en los casos de arrays con pocos elementos. Por tanto es muy eficiente para grandes conjuntos de datos, el problema reside a la hora de seleccionar el pivote. Si se selecciona un pivote que este situado en uno de los extremos el algoritmo o de forma aleatoria podrán tener en algunos casos una estructura de $O(n^2)$ operaciones. No obstante en la mayoría de los casos la estructura será $O(n \log n)$.

Ordenaciones array [50]	Comparaciones	Swaps
Ordenado	286	0
Invertido	336	25
Poco desordenado	510	73
Muy desordenado	503	75

Tabla 4 – Resultado análisis Quicksort

Con esto llegamos a la conclusión que es el algoritmo más eficiente para trabajar con conjuntos numéricos muy grandes ya que a medida que vamos ordenando el array se va reduciendo el tamaño del problema en particiones más pequeñas consiguiendo así que el algoritmo sea $O(n \log n)$. Aunque en los casos con conjuntos de elementos pequeños tampoco hay mucha diferencia entre los algoritmos analizados anteriormente.