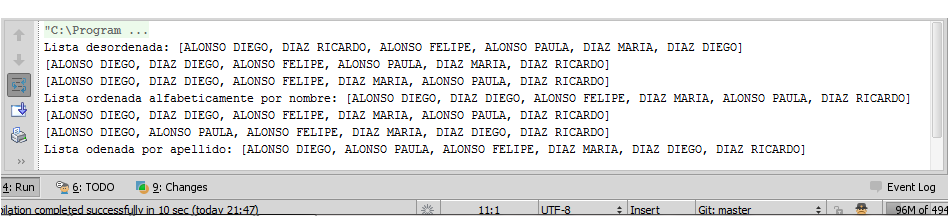
Trabajo Práctico nº 3

1) a) ¿Es shellsort estable? Justificar y dar un ejemplo.

Shellsort no es estable, no preserva el orden relativo de claves iguales. Es decir, si un conjunto de datos estaba ordenado originalmente de un modo, y se lo ordena de otro utilizando shellsort, se pierde el orden original. Esto se debe a la forma en que ordena shellsort. Al no ordenar a los elementos que se encuentran continuos entre sí, si no los que se encuentran alejados en un salto h, el orden original se pierde.

Esto se puede ver en el siguiente ejemplo:

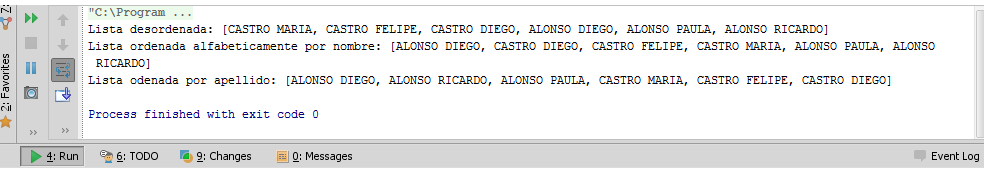


En el ejemplo podemos ver que luego de ordenarse alfabeticamente por nombre, al ordenarse por apellido, DIAZ DIEGO cambia de lugar con ALONSO PAULA, que queda ubicada antes que ALONSO FELIPE. Si el algoritmo fuera estable, hubiera ubicado a FELIPE antes que a PAULA. Lo mismo sucede con MARIA y RICARDO.

b) ¿Es quicksort estable? Justificar y dar un ejemplo.

Quicksort no es un algoritmo estable. Esto se debe a que en el proceso de particionamiento, una clave puede ser movida cruzando a otras iguales, sin haber sido examinadas, en cualquier intercambio.

Esto se puede ver en el siguiente ejemplo:



Quicksort primero ordeno la lista alfabeticamente por nombre, y luego por apellido. Se puede observar que al ordenar por apellido, se pierde el orden anterior por nombre. Si el algoritmo fuera estable, “ALONSO RICARDO” deberia estar despues de “ALONSO PAULA”, y “CASTRO DIEGO” antes que “FELIPE” y este a su vez, antes que “MARIA”.

c) El proceso de particionamiento consiste en elegir un elemento del arreglo, denominado "pívot", y de esta forma "partir" el arreglo en dos secciones. Luego de esto, procede a mandar de un lado del pívot los elementos que son menores a él, y del otro los que son mayores. Es por esto que el grafico queda así en la última escena. El arreglo quedo dividido en dos subarreglos, uno que contiene los elementos menores al pívot y otro con los mayores. Después de este proceso de partición, se ordenan los subarreglos.

3) ShellSort

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Shell Sort | | | | | | |
|  | Random - 1st Sequence (1,8,23,77,281,1073,4193,16577) | | | Random - 2nd Sequence (1,4,13,40,121,364,1093,3280,9841) | | |
| Lenght | Greater 1st | Swaps 1st | Time 1st | Greater 2nd | Swaps 2nd | Time 2nd |
| 100 | 631 | 369 | 9 | 535 | 224 | 9 |
| 1000 | 7763 | 3458 | 20 | 8519 | 3417 | 21 |
| 10000 | 98681 | 37555 | 69 | 111123 | 19339 | 70 |

A partir del gráfico, podemos observar que el tiempo crece de forma logarítmica. Vemos que la diferencia entre las dos secuencias de h es mínima, la segunda tarda un milisegundo menos en N = 1000 y N =10000.

Si comparamos la cantidad de comparaciones que hace veremos:

La segunda secuencia crece más rápido, es decir, realiza más comparaciones que la primera secuencia. Esto se debe a que se utiliza una partición más de h. Sin embargo, la segunda secuencia realiza muchos menos swaps que la primera secuencia:

Esto se debe a que al ser mayor la cantidad de h utilizadas, se h-ordena mediante un h de más, provocando que sean necesarios menos swaps. Sin embargo, el tiempo de ejecución sigue siendo muy parecido para ambas secuencias.