Luis Palomo - greywar
viernes 11 de febrero del 2010
Concepción, Chile

Tarea de investigación, semana 2
Algoritmos y Estructuras de datos P2PU.

---

http://p2pu.org/general/algority-estructuras-de-datos

- **Insert Sort:**
  Índice de complejidad: O(n²).
  Estable: Sí.
  Ventajas: Ordenamiento interno, requiere una variable adicional para realizar los intercambios, fácil implementación, requerimientos de memoria mínimos.
  Desventajas: Lento, realiza numerosas comparaciones.
  Conclusión: Tiene mejor orden de complejidad que bubble sort (ya que bubble sort es O(n²) sólo en el mejor de los casos), pero es muy ineficiente comparado con Quick Sort. Es una buena opción para listas pequeñas. Puede ser una buena opción para listas que está ordenadas o semi ordenadas.
- **Select Sort:**

Descipción: Un bucle en que, mientras la sub-lista sea de tamaño 2 o más, en cada pasada se busca el menor elemento de la sub-lista generada por la pasada anterior y luego se intercambia por el primer elemento de la sub-lista. En cada pasada la sublista se genera cortandole la primera posición a la lista de la pasada anterior.
Índice de complejidad: O(n²).
Estable: No.
Ventajas: Ordenamiento interno, sólo requiere una variable adicional para realizar los intercambiios, fácil implementación, realiza pocos intercambios, rendimiento constante (poca diferencia entre el mejor y el peor caso).
Desventajas: Lento, realiza numerosas comparaciones, no mejora en nada su rendimiento cuando la lista ya está parcialmente ordenada.
Conclusión: Ligeramente mejor que el bubble sort. En el caso de ordenar listas de enteros esta mejora no es muy grande, pero cuando se trata de estructuras complejas puede ser notable. Puede ser una buena opción para listas con registros grandes y claves pequeñas.

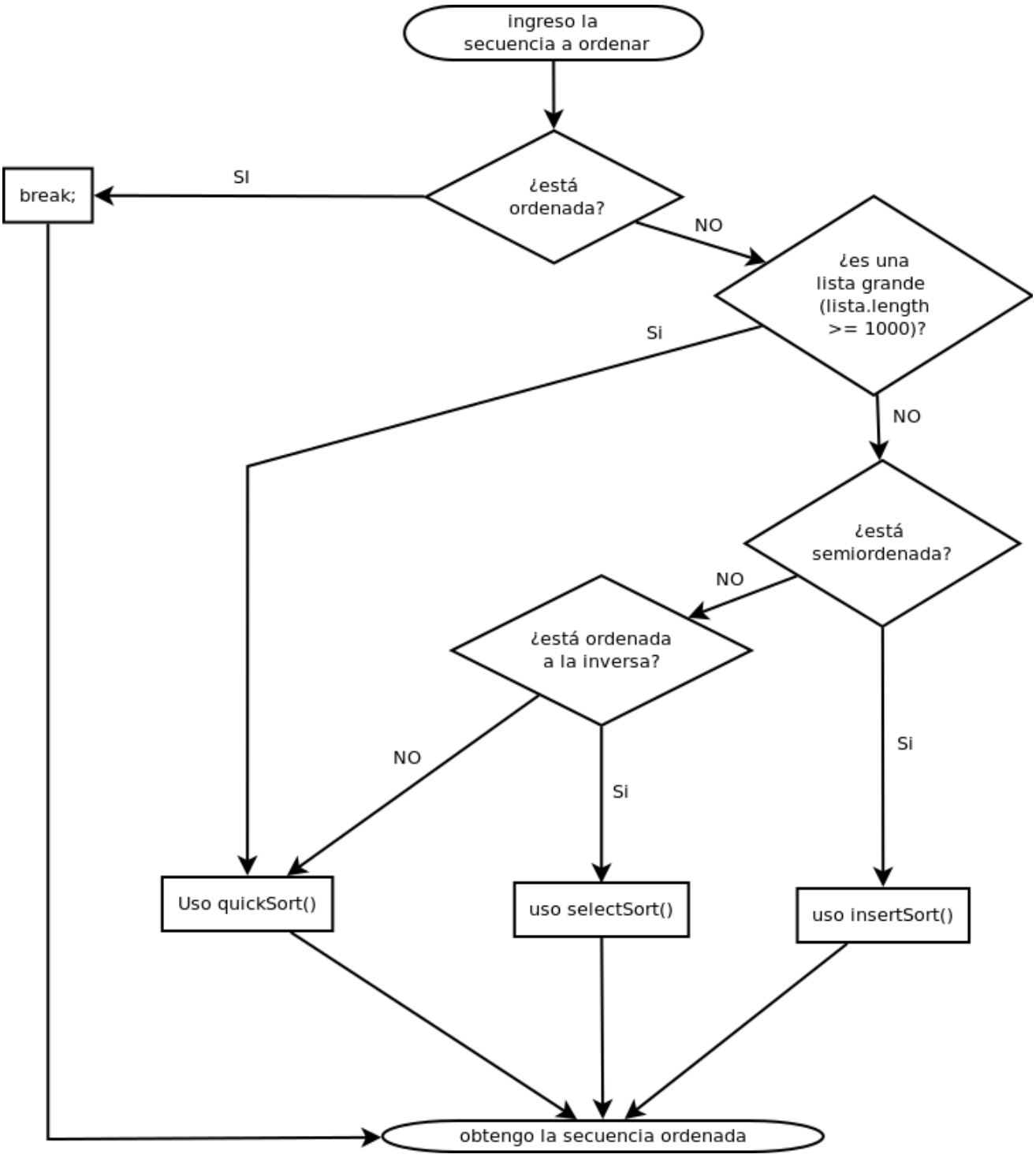
- **Bubble Sort:**
  Descripción: Se recorre la lista comparando pares de elementos contiguos e interambiandolos si es necesario.
  Índice de complejidad: O(n²).
  Estable: Sí.
  Ventajas: Ordenamiento interno, sólo requiere una variable adicional para realizar los intercambios, fácil implementación.
  Desventajas: Muy lento, realiza numerosas comparaciones, realiza numerosos intercambios, realiza todas las comparaciones aunque la lista ya esté ordenada, sólo es eficiente con listas de tamaño muy reducido.
  Conclusión: Es extremádamente ineficiente, aun dentro del grupo de los algoritmos O(n²). Es recomendable no usarlo nunca.

Quick Sort:
Descripción: Se elige un elemento de la lista, se pone en la posición que le corresponde en la lista ordenada y luego se acomodan todos los elementos menores que él y al otro todos los mayores con él. Se repite el procedimiento en un bucle hasta que las sublistas generadas sean de un elemento, en este punto la lista ya estará ordenada.
Estable: No.
Índice de complejidad: O(n*log2(n)) en un caso promedio y O(n²) en el peor caso.
Ventajas: Ordenamiento interno, no requiere memoria adicional, es muy rápido.
Desventajas: Implementación un poco complicada, es un algoritmo recursivo (en general, los algoritmos recursivos son más lentos y consumen más recursos), mucha diferencia entre el mejor y el peor caso. El peor caso es cuando la lista está

inicialmente ordenada, caso en el que el rendimiento se degrada a $O(n^2)$.
Conclusión: Es un algoritmo que se puede usar en la práctica (confinando a todos los anteriormente nombrados a la teoría nada más). Es muy eficiente y, en general, será la mejor opción.
Quick Sort aleatorio: Una forma es reordenar de forma aleatoria la secuencia inicial y luego aplicar el algoritmo, otra es modificar el algoritmo para que cada partición de la secuencia original se reordene de manera aleatoria. Haciendo esto se logra que ninguna secuencia caiga en el peor caso del algoritmo (tampoco en el mejor), con lo que el índice de complejidad del algoritmo sería siempre $O(n*log2(n))$.

Algoritmo de selección de método de ordenamiento de menor a mayor

- Diagrama de flujo:



- Código del algoritmo: https://docs.google.com/leaf?id=0B5S-

zOnhceU1ZWQ4ZjE2YzQtYTNkYi00MjAzLTk1MmQtMzJhYTM1YWQ1ZDNl&hl=es