QuickSort

Quicksort utiliza divide y vencerás, por lo que es un algoritmo recursivo. Selecciona un elemento como pivote y divide el conjunto dado alrededor del pivote elegido.

**El algoritmo trabaja de la siguiente forma:**

* Elegir un elemento de la lista de elementos a ordenar, al que llamaremos pivote.
* El proceso clave en quickSort es la partición. El objetivo de las particiones es, dado una lista y un elemento x de la lista como pivote, poner x en su posición correcta en la lista ordenada y poner todos los elementos más pequeños (menores que x) antes de x, y poner todos los elementos mayores (mayores que x) después x.
* La lista queda separada en dos sublistas, una formada por los elementos a la izquierda del pivote, y otra por los elementos a su derecha.
* Repetir este proceso de forma recursiva para cada sublista mientras éstas contengan más de un elemento. Una vez terminado este proceso todos los elementos estarán ordenados.

La selección de pivote es una parte importante de la clasificación rápida y existen muchas técnicas, todas con pros y contras.  
  
**¿Por qué importa el pivote?**

En promedio, la ordenación rápida se ejecuta en O (n log n) pero si seleccionando consistentemente pivotes defectuosos, su rendimiento se degrada a O (n ^ 2).  
Esto sucede si el pivote se selecciona de manera consistente de modo que todos (o muchos de) los elementos en la lista sean el pivote.  
(Un ejemplo de esto es cuando el primer o el último elemento se seleccionando como un pivote y los datos ya están ordenados, o casi ordenados).  
  
**Las técnicas utilizadas para elegir un pivote en este proyecto fueron las siguientes:**

* Seleccionando el elemento medio:  
  Pros: simple de codificar, rápido de calcular, pero un poco más lento que las demás técnicas.  
  Contras: Todavía puede degradarse a O (n ^ 2). Es fácil para alguien construir una lista que hará que se degrade a O (n ^ 2).
* Seleccionando el pivote aleatoriamente (usando la función integrada incorporada):  
  Pros: simple de codificar. Más difícil para alguien construir una lista que hará que se degrade a O (n ^ 2)  
  Contras: La selección de un pivote aleatorio es bastante lenta. Aún puede degradarse a O (n ^ 2).

**Otras técnicas que pueden ser implementadas para la selección de pivotes son:**

* Seleccionando el elemento más a la izquierda o más a la derecha.  
  Pros: simple de codificar, rápido de calcular  
  Contras: si los datos están ordenados o casi ordenados, la ordenación rápida se degradará a O (n ^ 2)
* Seleccionando el pivote aleatoriamente (usando una función aleatoria personalizada):  
  Pros: Mucho más difícil para alguien construir un arreglo que lo hará degradarse a O (n ^ 2), si no sabe cómo está eligiendo los números aleatorios.  
  Contras: puede ser complicado codificar. Seleccionar un pivote aleatorio es bastante lento. Todavía teóricamente es posible que pueda degradarse a O (n ^ 2).
* Seleccionando la mediana de tres:  
  Pros: bastante simple de codificar, razonablemente rápido de calcular, pero un poco más lento que los métodos anteriores  
  Contras: Todavía puede degradarse a O (n ^ 2). Es bastante fácil para alguien construir una lista que hará que se degrade a O (n ^ 2).
* Usando el método de la mediana de las medianas para seleccionar un pivote  
  Pros: El pivote está garantizado para ser bueno. El QuickSort ahora es O (n log n) el peor caso.  
  Contras: código complicado. Típicamente, mucho más lento que los métodos anteriores.

**¿Cuál técnica se recomienda usar a la hora de seleccionar el pivote?**

* Es poco probable que los datos estén ordenados, y usted está dispuesto a aceptar O (n ^ 2) en los raros casos en que se ordena la lista, utilice el elemento situado más a la izquierda o más a la derecha.
* Si hay una posibilidad razonable de que sus datos estén ordenados, utilice el elemento medio o la mediana de tres.
* Si existen malos arreglos para ordenar (usados ​​como ataque de denegación de servicio), utiliza pivotes aleatorios.
* Si está realmente desea garantizar que el quicksort se ejecute es O (n log n), utilice la mediana de las medianas. En este punto, es posible que desee considerar seriamente el uso de un método de clasificación diferente, como el tipo de fusión.

**Tiempo de corrida del algoritmo:**

**Peor caso:**

Cuando el QuickSort tiene las particiones más desequilibradas posibles, entonces la llamada original toma **c\*n**, c tiempo constante, la llamada recursiva en **n - 1** elementos toma **c\*(n-1)** tiempo, la llamada recursiva en **n - 2** elementos toma **c\*(n-2)** tiempo, y así sucesivamente hasta llegar a una lista de tamaño 2, la cual toma **2\*c** tiempo.

**Imagen que contiene texto, mapa

Descripción generada con confianza muy alta**

*c\*n*+ c\*(*n*−1) +c\*(*n*−2) +… +2\*c = *c\** (n + (n-1) +(n-2) +... +2)

= c \* ((n+1) \* (n/2)-1)

= c \* ((n^2) /2 + (n/2) – 1)

Y se concluye por la jerarquía de O-grande que el tiempo de corrida del algoritmo es O(n^2)

​=E