# Sortering i Java-biblioteket

Peter Ljunglöf

DAT036, Datastrukturer

10 dec 2012

# static void Arrays.sort(X[])

JDK-7 innehåller flera sorteringsmetoder:

```
static void sort(byte[] a)
static void sort(char[] a)
static void sort(double[] a)
static void sort(float[] a)
static void sort(int[] a)
static void sort(long[] a)
static void sort(short[] a)

static void sort(Object[] a)
static <T> void sort(T[] a, Comparator<? super T> c)
```

- dessutom lika många där man kan ange ett intervall:

  ```
  static void sort(byte[] a, int fromIndex, int toIndex)
  ...
  ```

Se källkoden för OpenJDK-7:

http://www.docjar.com/html/api/java/util/Arrays.java.html

# Sortering av primitiva typer

De primitiva typerna (byte, char, double, float, int, long, short) sorteras såhär:

```
public static void sort(int[] a) {
    DualPivotQuicksort.sort(a);
}
```

...vilket är en variant av Quicksort, men inte bara!

- Merge sort för långa listor
- Insertion sort för korta listor
- Counting sort för långa byte-/short-/char-listor

Se källkoden:

http://www.docjar.com/html/api/java/util/DualPivotQuicksort.java.html

# Sortering av primitiva typer

”Tuning parameters” från källkoden:

```
// The maximum number of runs in merge sort.
private static final int MAX_RUN_COUNT = 67;

// The maximum length of run in merge sort.
private static final int MAX_RUN_LENGTH = 33;

// If the length of an array to be sorted is less than this constant,
// Quicksort is used in preference to merge sort.
private static final int QUICKSORT_THRESHOLD = 286;

// If the length of an array to be sorted is less than this constant,
// insertion sort is used in preference to Quicksort.
private static final int INSERTION_SORT_THRESHOLD = 47;

// If the length of a byte array to be sorted is greater than this constant,
// counting sort is used in preference to insertion sort.
private static final int COUNTING_SORT_THRESHOLD_FOR_BYTE = 29;

// If the length of a short or char array to be sorted is greater than
// this constant, counting sort is used in preference to Quicksort.
private static final int
            COUNTING_SORT_THRESHOLD_FOR_SHORT_OR_CHAR = 3200;
```

# Dual-pivot quicksort

Grundidé:

- använder 2 pivoter, $p_1$ och $p_2$.
- partitionerar listan i tre delar:
  $\{x \mid x < p_1\},\ \{x \mid p_1 \leq x \leq p_2\},\ \{x \mid p_2 < x\}$

Från källkoden:

”The sorting algorithm is a Dual-Pivot Quicksort by Vladimir Yaroslavskiy, Jon Bentley, and Joshua Bloch.

This algorithm offers O($n$ log($n$)) performance on many data sets that cause other quicksorts to degrade to quadratic performance, and is typically faster than traditional (one-pivot) Quicksort implementations.”

# Sortering av komplexa objekt

Objekt sorteras med TimSort:

```
public static void sort(Object[] a) {
    ComparableTimSort.sort(a);
}

public static <T> void sort(T[] a, Comparator<? super T> c) {
    TimSort.sort(a, c);
}
```

Källkoden finns här:

http://www.docjar.com/html/api/java/util/TimSort.java.html

...och här:

www.docjar.com/html/api/java/util/ComparableTimSort.java.html

- ”This is a near duplicate of TimSort, modified for use with arrays of objects that implement Comparable...”

# TimSort

”A stable, adaptive, iterative mergesort that requires far fewer than $n \cdot \lg(n)$ comparisons when running on partially sorted arrays, while offering performance comparable to a traditional mergesort when run on random arrays.

Like all proper mergesorts, this sort is stable and runs in $O(n \log n)$ time (worst case). In the worst case, this sort requires temporary storage space for $n/2$ object references; in the best case, it requires only a small constant amount of space.

This implementation was adapted from Tim Peters's list sort for Python, which is described in detail here:”

http://svn.python.org/projects/python/trunk/Objects/listsort.txt

En mer lättillgänglig förklaring finns här:

http://en.wikipedia.org/wiki/Timsort

# TimSort

”Tuning parameters” från källkoden:

```
// This is the minimum sized sequence that will be merged.
// Shorter sequences will be lengthened by calling binarySort.
private static final int MIN_MERGE = 32

// When we get into galloping mode, we stay there until both
// runs win less often than MIN_GALLOP consecutive times.
private static final int  MIN_GALLOP = 7

// Maximum initial size of tmp array, which is used for
// merging.  The array can grow to accommodate demand.
private static final int INITIAL_TMP_STORAGE_LENGTH = 256

// A stack of pending runs yet to be merged.
// Run i starts at address runBase[i].
private final int[] runBase

// Allocate runs-to-be-merged stack (which cannot be expanded).
int stackLen = (a.length <      120  ?   5 :
                a.length <    1542  ? 10 :
                a.length < 119151  ? 19 : 40)
runBase = new int[stackLen]
```