Quicksort

Paolo Costa

20 maggio 2004

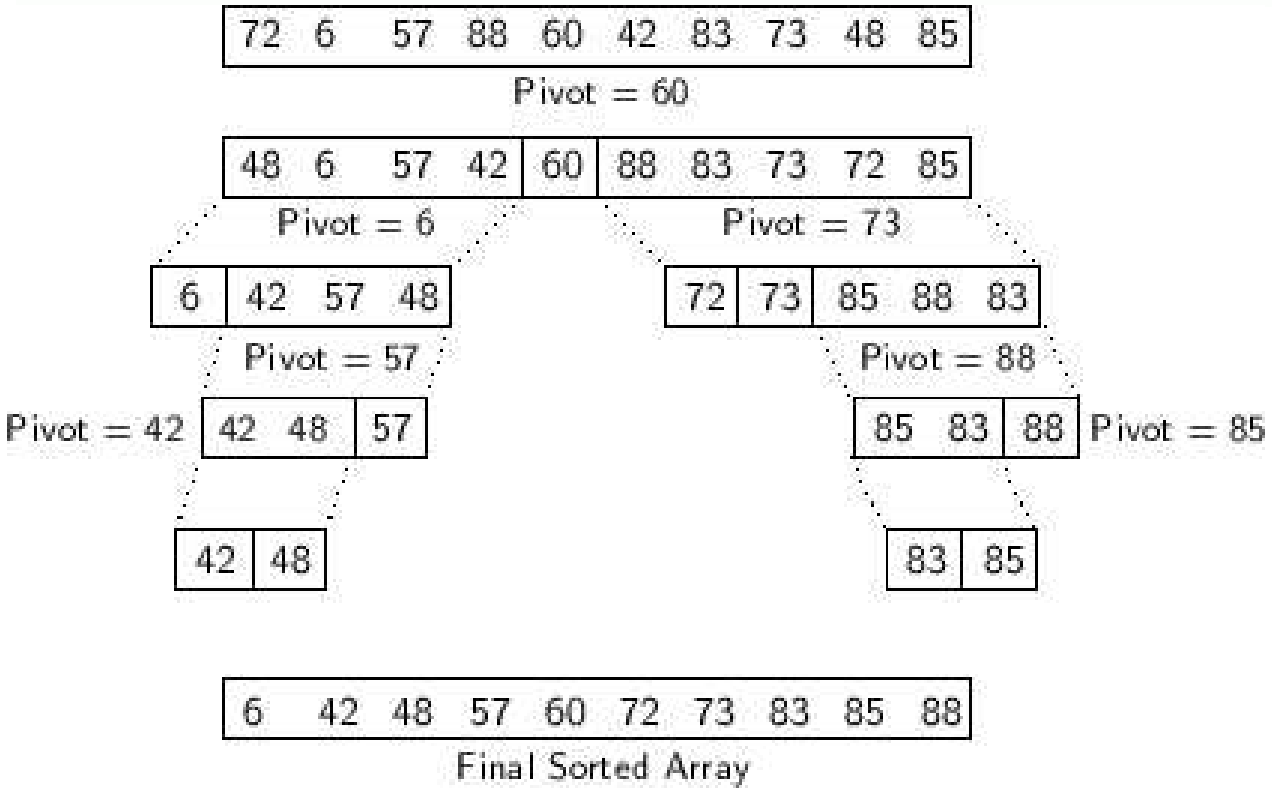
parte dei lucidi sono tratti dal materiale di Clifford Shaffer

Algorithm Details

When properly implemented, it is the fastest known general purpose sorting algorithm in the average case. The Quicksort algorithm uses a recursive divide and conquer strategy to sort a list. The steps are:

1. Pick a pivot element from the list
2. Reorder the list so that all elements less than the pivot precede all elements greater than the pivot. This means that the pivot is in its final place; the algorithm puts at least one element in its final place on each pass over the list. This step is commonly referred to as *partitioning*
3. Recursively sort the sub-list of elements less than the pivot and the sub-list of elements greater than the pivot. If one of the sub-lists is empty or contains one element, it can be ignored.

Example


72 6 57 88 60 42 83 73 48 85
Pivot = 60
48 6 57 42 60 88 83 73 72 85
Pivot = 6
Pivot = 73
6 42 57 48
72 73 85 88 83
Pivot = 57
Pivot = 88
Pivot = 42
42 48 57
85 83 88
Pivot = 85
42 48
83 85
6 42 48 57 60 72 73 83 85 88
Final Sorted Array

The code

```
static void qsort(Elem[] array, int i, int j) {
  int pivotindex = findpivot(array, i, j); // Pick piv
  DSutil.swap(array, pivotindex, j); // Stick at end
  // k will be the first position in the right subarray
  // NB i-1 and j are out of the segment to partition
  int k = partition(array, i-1, j, array[j].key());
  DSutil.swap(array, k, j);    // Put pivot in place
  if ((k-i) > 1) qsort(array, i, k-1);  // Sort left
  if ((j-k) > 1) qsort(array, k+1, j);  // Sort right
}

static int partition(Elem[] array, int l, int r, int pivot)
 { //l and r initially outside segment to partition;
  do {// move l and r inward until they meet
    while (array[++l].key() < pivot); // Move it right
    while ((r!=0) && (array[--r].key()>pivot));
    DSutil.swap(array, l, r); // Swap out-of-place vals
  } while (l < r);             // Stop when they cross
  DSutil.swap(array, l, r);   // Reverse wasted swap
  return l; }// Return first pos in right partition
```

Cost for Quicksort: Best and Worst Case

Cost of findpivot: $\Theta(1)$ Cost of partition: $\Theta(n)$ Best Case: Always partition in half.

- $\log n$ levels of partitions with size $\frac{n}{2^i}$
- for each level total partition cost is $\Theta(n)$

$\Rightarrow$ total cost $\Theta(n \log n)$

$$\begin{aligned} T(n) &= 2T(\frac{n}{2}) + n = 2(2T(\frac{n}{4}) + \frac{n}{2}) + n = 4T(\frac{n}{4}) + 2n \\ &= c \log n + \sum_{k=0}^{\log n} n = c \log n + n(\log n + 1) = \Theta(n \log n) \end{aligned}$$

Worst Case: Bad partition (quite unlikely).

$$\begin{aligned} T(n) &= T(n-1) + T(1) + n = T(n-2) + T(1) + (n-1) + T(1) + n \\ &= nT(1) + \sum_{k=1}^{n} k = \Theta(n^2) \end{aligned}$$

Cost for Quicksort: Average Case

Average Case: reasonable simplifying assumption, pivot position after sorting equally likely to be 0, 1, 2, ... $n-1$.

$T(n) = cn + \frac{1}{n}\sum_{k=0}^{n-1}(T(k) + T(n-1-k))$

$T(0) = T(1) = c$

but symmetric terms are equal: $T(n) = cn + \frac{2}{n}\sum_{k=0}^{n-1} T(k))$

This is a *recurrence relation*: solve it using the *shifting method*

$$\begin{aligned} T(n) &= n + 1 + \frac{1}{n-1}\sum_{k=1}^{n-1}(T(k) + T(n-k)) \\ &= \Theta(n \log n) \end{aligned}$$

Solving the Qsort recurrence

[multiply both sides by n] $nT(n) = cn^2 + 2\sum_{k=0}^{n-1} T(k))$

[formula for $n+1$]

$(n+1)T(n+1) = c(n+1)^2 + 2\sum_{k=0}^{n} T(k))$

[subtract $nT(n)$ from both sides]

$(n+1)T(n+1) - nT(n) = c(n+1)^2 - cn^2 + 2T(n)$

$(n+1)T(n+1) - nT(n) = c(2n+1) + 2T(n)$

$(n+1)T(n+1) = c(2n+1) + (n+2)T(n)$

$T(n+1) = \frac{c(2n+1)}{n+1} + \frac{n+2}{n+1}T(n)$

[NB: $\frac{c(n+1)}{n+1} < 2c \Rightarrow$ **T(n)** $\leq 2c + \frac{n+1}{n}T(n-1)$**]**

Solving the Qsort recurrence - 2

[expand the recurrence]

$$
\begin{aligned}
T(n+1) &\leq 2c + \frac{n+2}{n+1} T(n) \\
&\leq 2c + \frac{n+2}{n+1} \left(2c + \frac{n+1}{n} T(n-1) \right) \\
&\leq 2c + \frac{n+2}{n+1} \left(2c + \frac{n+1}{n} \left(2c + \frac{n}{n-1} T(n-2) \right) \right) \\
&\leq 2c + \frac{n+2}{n+1} \left(2c + \cdots + \frac{4}{3} \left(2c + \frac{3}{2} T(1) \right) \right) \\
&\leq 2c \left(1 + \frac{n+2}{n+1} + \frac{(n+2)(n+1)}{(n+1)n} + \cdots + \frac{(n+2)(n+1)\cdots 3}{(n+1)n\cdots 2} \right)
\end{aligned}
$$

Solving the Qsort recurrence - 3

$$\begin{aligned} T(n+1) &\leq 2c\left(1+(n+2)\left(\frac{1}{n+1}+\frac{1}{n}+\cdots+\frac{1}{2}\right)\right) \\ &\leq 2c+2c(n+2)(H_{n+1}-1) \end{aligned}$$

Where H_{n+1} is the Harmonic series.

H_{n+1} is $\Theta(n \log n)$, so $T(n)$ is $\Theta(n \log n)$

Finding a better pivot

Quicksort's worst case arises when pivot does a poor job of splitting the array into equal size subarrays. Sorted or partially sorted data is quite common in practice and the naive implementation which selects the first element as the pivot does poorly with such data. To avoid this problem the middle element can be used.

Other optimizations

Better algorithm for small sublists : do nothing for small arrays (e.g., for $n \leq 9$: this reduces # of function calls by 80% - 90%) $\Rightarrow$ obtain an array that is "nearly sorted" then final sorting pass using insertion sort (which in this case is nearly linear).

Eliminate recursion ($\Rightarrow$ use iteration + stack): no copy of the array is needed, only array bounds in the stack.