CS062
DATA STRUCTURES AND ADVANCED PROGRAMMING
19: Quicksort

Alexandra Papoutsaki

Lectures

Mark Kampe

Labs

SORTING
Lecture 19: Quicksort

- Quicksort
Mergesort and Quicksort: the classics

- Mergesort used in Java to sort objects.
- Quicksort used in Java to sort primitives.
- Quicksort was invented by Sir Tony Hoare in 1959.
  - Wanted to sort Russian words before looking them up in dictionary.
  - Came up with quicksort but did not know how to implement it.
  - Learned Algol 60 and recursion and implemented it.
  - Won the 1980 Turing Award.
- Bob Sedgewick (author of your textbook) refined and analyzed many versions of quicksort.
QUICKSORT

Basic Plan

- Shuffle the array.
- Partition so that, for some pivot $j$:
  - Entry $a[j]$ is in place.
  - There is no larger entry to the left of $j$.
  - No smaller entry to the right of $j$.
- Sort each subarray recursively.
QUICKSORT

Partition

- Partition the subarray $a[lo...hi]$ so that $a[lo...j-1] \leq a[j] \leq a[j+1...hi]$
- Start with pointer $i$ at $lo$ and pointer $j$ at $hi+1$.
- Repeat the following until pointers $i$ and $j$ cross:
  - Scan $i$ from left to right as long as $a[i] < a[lo]$.
  - Scan $j$ from right to left as long as $a[j] > a[lo]$.
  - Exchange $a[i]$ with $a[j]$.
QUICKSORT

Partition Example

| i | j | k | r | a | t | e | l | e | p | u | i | m | q | c | x | o | s |
| 0 | 16| K | R | A | T | E | L | E | P | U | I | M | Q | C | X | O | S |

Initial values

Scan left, scan right

Exchange

Scan left, scan right

Exchange

Scan left, scan right

Exchange

Scan left, scan right

Final exchange

Result

Partitioning trace (array contents before and after each exchange)
QUICKSORT

Partition Code

// partition the subarray a[lo..hi] so that a[lo..j-1] <= a[j] <= a[j+1..hi]
// and return the index j.
private static int partition(Comparable[] a, int lo, int hi) {
    int i = lo;
    int j = hi + 1;
    Comparable v = a[lo];
    while (true) {
        // find item on lo to swap
        while (less(a[++i], v)) {
            if (i == hi) break;
        }

        // find item on hi to swap
        while (less(v, a[--j])) {
            if (j == lo) break; // redundant since a[lo] acts as sentinel
        }

        // check if pointers cross
        if (i >= j) break;

        exch(a, i, j);
    }

    // put partitioning item v at a[j]
    exch(a, lo, j);

    // now, a[lo .. j-1] <= a[j] <= a[j+1 .. hi]
    return j;
}
/**
 * Rearranges the array in ascending order, using the natural order.
 * @param a the array to be sorted
 */

class Quicksort {
    public static void sort(Comparable[] a) {
        StdRandom.shuffle(a);
        sort(a, 0, a.length - 1);
    }

    // quicksort the subarray from a[lo] to a[hi]
    private static void sort(Comparable[] a, int lo, int hi) {
        if (hi <= lo) return;
        int j = partition(a, lo, hi);
        sort(a, lo, j-1);
        sort(a, j+1, hi);
    }
}
Quicksort Demo

<table>
<thead>
<tr>
<th>lo</th>
<th>j</th>
<th>hi</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td>15</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>9</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>13</td>
<td>15</td>
</tr>
<tr>
<td>10</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
</tbody>
</table>

Initial values
random shuffle

Quicksort Example

K R A T  E  L  Q  C  X  U  I  P  M  T  U  M  Q  R  X  O  S
E C A I E K L P U T M Q R X O S
E C A E I K L P U T M Q R X O S
A C E E E I K L P U T M Q R X O S
A C E E I K L P U T M Q R X O S
A C E E I K L P U T M Q R X O S
A C E E I K L P U T M Q R X O S
A C E E I K L M O P T Q R X U S
A C E E I K L M O P T Q R X U S
A C E E I K L M O P T Q R X U S
A C E E I K L M O P S Q R T U X
A C E E I K L M O P R Q S T U X
A C E E I K L M O P Q R S T U X
A C E E I K L M O P Q R S T U X
A C E E I K L M O P Q R S T U X
A C E E I K L M O P Q R S T U X
A C E E I K L M O P Q R S T U X

No partition for subarrays of size 1

Result

A C E E I K L M O P Q R S T U X
Quicksoort Considerations

- **Partitioning in-place**: Using an extra array makes partitioning easier (and stable), but it is not worth the cost.

- **Terminating the loop**: Testing whether the pointers cross is trickier than it might seem.

- **Equal keys**: When duplicate keys are present, it is (counter-intuitively) better to stop scans on keys equal to the partitioning item’s key.

- **Preserving randomness**: Shuffling is needed for performance guarantee.
  - **Equivalent alternative**: Pick a random partitioning item in each subarray.
Great algorithms are better than good ones

- Your laptop executes $10^8$ comparisons per second
- A supercomputer executes $10^{12}$ comparisons per second

<table>
<thead>
<tr>
<th></th>
<th>Insertion sort</th>
<th>Mergesort</th>
<th>Quicksort</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Computer</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Home</td>
<td>Instant</td>
<td>2 hours</td>
<td>300 years</td>
</tr>
<tr>
<td></td>
<td></td>
<td>instant</td>
<td>1 sec</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>15 min</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Instant</td>
<td>0.5 sec</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>10 min</td>
</tr>
<tr>
<td>Supercomputer</td>
<td>Instant</td>
<td>1 second</td>
<td>1 week</td>
</tr>
<tr>
<td></td>
<td></td>
<td>instant</td>
<td>instant</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>instant</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Instant</td>
</tr>
</tbody>
</table>
Quicksort analysis: best case

- Quicksort divides everything exactly in half.
- Similar to merge sort
- Number of compares is $\sim n \log n$
Quicksort analysis: worst case

- Data are already sorted.
- Number of compares is $\sim 1/2n^2$ - quadratic!
- Extremely unlikely if we shuffle and our shuffling is not broken.
Quicksort - things to remember

- \( \sim 2n \ln n \) or \( 1.39n \log n \) compares on average
  - 39% more compares than merge sort but in practice is faster because it does not move data much.
  - If good implementation, even in sorted arrays it can be linearithmic. If not, we end up with quadratic.
- \( \frac{1}{3}n \ln n \) exchanges.
- We won’t do the analysis.
- In-place sorting.
- **Not** stable
Quicksort practical improvements

- Use insertion sort for small subarrays.
  - Too much overhead for tiny subarrays.
  - Cutoff to insertion sort usually around 10 items.
- Best choice of pivot is the median
Lecture 19: Quicksort

- Quicksort
Readings:

- Textbook:
  - Chapter 2.3 (Pages 288-296)

- Website:

Practice Problems:

- 2.3.1-2.3.4