

# **QUICKSORT IS OPTIMAL**

**Robert Sedgewick  
Jon Bentley**

# MOTIVATION

**MOORE'S LAW:** Processing Power Doubles every 18 months  
but also:

- ◇ memory capacity doubles every 18 months
- ◇ problem size expands to fill memory

**Sedgewick's Corollary:** Need Faster Sorts every 18 months!  
(annoying to wait longer, even to sort twice as much, on new machine)

old:  $N \lg N$

new:  $(2N \lg 2N)/2 = N \lg N + N$

Other compelling reasons to study sorting

- ◇ cope with new languages, machines, and applications
- ◇ rebuild obsolete libraries
- ◇ intellectual challenge of basic research

**Simple fundamental algorithms:** the ultimate portable software

# Quicksort

```
void quicksort(Item a[], int l, int r)
{ int i = l-1, j = r; Item v = a[r];
  if (r <= l) return;
  for (;;)
  {
    while (a[++i] < v) ;
    while (v < a[--j]) if (j == l) break;
    if (i >= j) break;
    exch(a[i], a[j]);
  }
  exch(a[i], a[r]);
  quicksort(a, l, i-1);
  quicksort(a, i+1, r);
}
```

Detail (?): How to handle keys equal to the partitioning element

## Partitioning with equal keys

How to handle keys equal to the partitioning element?

**METHOD A:** Put equal keys all on one side?

|   |   |   |   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |

NO: quadratic for  $n=1$  (all keys equal)

**METHOD B:** Scan over equal keys? (linear for  $n=1$ )

|   |   |   |   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 4 | 1 | 1 | 4 | 4 | 4 | 1 | 4 | 1 | 1 | 4 | 4 |
| 1 | 1 | 1 | 1 | 4 | 4 | 4 | 1 | 4 | 1 | 4 | 4 | 4 |

NO: quadratic for  $n=2$

**METHOD C:** Stop both pointers on equal keys?

|   |   |   |   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 9 | 4 | 4 | 1 | 4 | 4 | 4 | 9 | 4 | 4 | 1 | 4 |
| 1 | 4 | 4 | 4 | 1 | 4 | 4 | 4 | 9 | 4 | 9 | 4 | 4 |

YES:  $N \lg N$  guarantee for small  $n$ , no overhead if no equal keys

## Partitioning with equal keys

How to handle keys equal to the partitioning element?

**METHOD C:** Stop both pointers on equal keys?

|   |   |   |   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 9 | 4 | 4 | 1 | 4 | 4 | 4 | 9 | 4 | 4 | 1 | 4 |
| 1 | 4 | 4 | 4 | 1 | 4 | 4 | 4 | 9 | 4 | 9 | 4 | 4 |

YES:  $N \lg N$  guarantee for small  $n$ , no overhead if no equal keys

**METHOD D (3-way partitioning):** Put all equal keys into position?

|   |   |   |   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 9 | 4 | 4 | 1 | 4 | 4 | 4 | 9 | 4 | 4 | 1 | 4 |
| 1 | 1 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 9 | 9 |

yes, BUT: early implementations cumbersome and/or expensive

# Quicksort common wisdom (last millennium)

## 1. Method of choice in practice

- ◇ tiny inner loop, with locality of reference
- ◇  $N \log N$  worst-case “guarantee” (randomized)
- ◇ but use a radix sort for small number of key values

## 2. Equal keys can be handled (with care)

- ◇  $N \log N$  worst-case guarantee, using proper implementation

## 3. Three-way partitioning adds too much overhead

- ◇ “Dutch National Flag” problem

## 4. Average case analysis with equal keys is intractable

- ◇ keys equal to partitioning element end up in both subfiles

## Changes in Quicksort common wisdom

### 1. Equal keys abound in practice.

- ◇ never can anticipate how clients will use library
- ◇ linear time required for huge files with few key values

### 2. 3-way partitioning is the method of choice.

- ◇ greatly expands applicability, with little overhead
- ◇ easy to adapt to multikey sort
- ◇ no need for separate radix sort

### 3. Average case analysis already done!

- ◇ Burge, 1975
- ◇ Sedgewick, 1978
- ◇ Allen, Munro, Melhorn, 1978

## Bentley-McIlroy 3-way partitioning

Partitioning invariant

|       |      |  |         |       |
|-------|------|--|---------|-------|
| equal | less |  | greater | equal |
|-------|------|--|---------|-------|

- ◇ move from left to find an element that is not less
- ◇ move from right to find an element that is not greater
- ◇ stop if pointers have crossed
- ◇ exchange
- ◇ if left element equal, exchange to left end
- ◇ if right element equal, exchange to right end

Swap equals to center after partition

|      |       |         |
|------|-------|---------|
| less | equal | greater |
|------|-------|---------|

### KEY FEATURES

- ◇ always uses  $N-1$  (three-way) compares
- ◇ no extra overhead if no equal keys
- ◇ only one "extra" exchange per equal key



## Quicksort with 3-way partitioning

```
void quicksort(Item a[], int l, int r)
{ int i = l-1, j = r, p = l-1, q = r; Item v = a[r];
  if (r <= l) return;
  for (;;)
  {
    while (a[++i] < v) ;
    while (v < a[--j]) if (j == l) break;
    if (i >= j) break;
    exch(a[i], a[j]);
    if (a[i] == v) { p++; exch(a[p], a[i]); }
    if (v == a[j]) { q--; exch(a[j], a[q]); }
  }
  exch(a[i], a[r]); j = i-1; i = i+1;
  for (k = l; k < p; k++, j--) exch(a[k], a[j]);
  for (k = r-1; k > q; k--, i++) exch(a[i], a[k]);
  quicksort(a, l, j);
  quicksort(a, i, r);
}
```

## Information-theoretic lower bound

**Definition:** An  $(x_1, x_2, \dots, x_n)$ -file has

$N = x_1 + x_2 + \dots + x_n$  keys,

$n$  distinct key values, with

$x_i \equiv$  number of occurrences of the  $i$ -th smallest key

$p_i \equiv x_i / N$

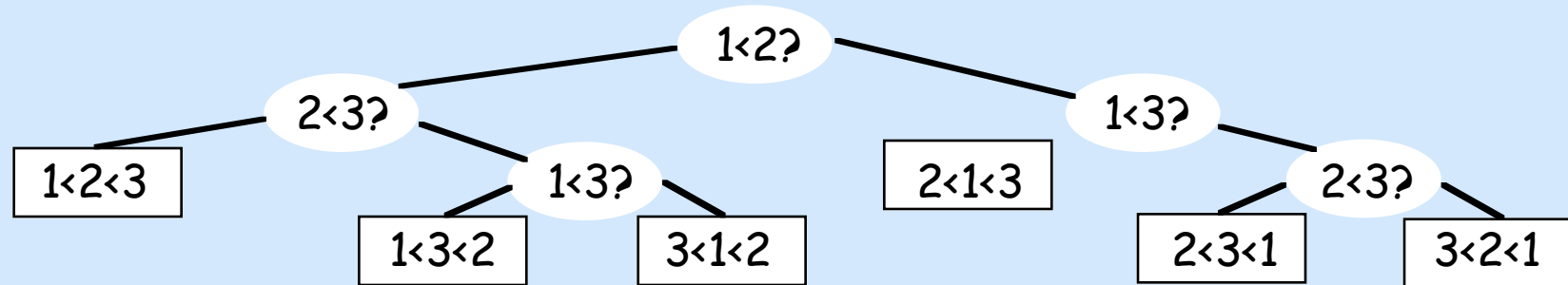
**THEOREM.** Any sorting method uses at least

$NH - N$  compares (where  $H = -\sum_{1 \leq k \leq n} p_k \lg p_k$  is the entropy)

to sort an  $(x_1, x_2, \dots, x_n)$ -file, on the average.

# Information-theoretic lower-bound proof

DECISION TREE describes all possible sequences of comparisons



Number of leaves must exceed number of possible files

$$\binom{N}{x_1 x_2 \dots x_n} = \frac{N!}{x_1! x_2! \dots x_n!}$$

Avg. number of compares is minimized when tree is balanced

$$C > \lg \frac{N!}{x_1! x_2! \dots x_n!} = \lg N! - \lg x_1! - \lg x_2! - \dots - \lg x_n!$$

By Stirling's approximation,

$$\begin{aligned} C &> N \lg N - N - x_1 \lg x_1 - x_2 \lg x_2 - \dots - x_n \lg x_n \\ &= (x_1 + \dots + x_n) \lg N - N - x_1 \lg x_1 - x_2 \lg x_2 - \dots - x_n \lg x_n \\ &= NH - N \end{aligned}$$

## Analysis of Quicksort with equal keys

1. Define  $C(x_1, \dots, x_n) \equiv C(1, n)$  to be the mean # compares to sort the file

$$C(1, n) = N - 1 + \frac{1}{N} \sum_{1 \leq j \leq n} x_j (C(1, j-1) + C(j+1, n))$$

2. Multiply both sides by  $N = x_1 + \dots + x_n$

$$NC(1, n) = N(N-1) + \sum_{1 \leq j \leq n} x_j C(1, j-1) + \sum_{1 \leq j \leq n} x_j C(j+1, n)$$

3. Subtract same equation for  $x_2, \dots, x_n$  and let  $D(1, n) \equiv C(1, n) - C(2, n)$

$$(x_1 + \dots + x_n)D(1, n) = x_1^2 - x_1 + 2x_1(x_2 + \dots + x_n) + \sum_{2 \leq j \leq n} x_j D(1, j-1)$$

4. Subtract same equation for  $x_1, \dots, x_{n-1}$

$$(x_1 + \dots + x_n)D(1, n) - (x_1 + \dots + x_{n-1})D(1, n-1) = 2x_1x_n + x_nD(1, n-1)$$

## Analysis of Quicksort with equal keys (cont.)

$$(x_1 + \dots + x_n)D(1, n) - (x_1 + \dots + x_{n-1})D(1, n-1) = 2x_1x_n + x_nD(1, n-1)$$

5. Simplify, divide both sides by  $N = x_1 + \dots + x_n$

$$D(1, n) = D(1, n-1) + \frac{2x_1x_n}{x_1 + \dots + x_n}$$

6. Telescope (twice)

$$C(1, n) = N - n + \sum_{1 \leq k < j \leq n} \frac{2x_kx_j}{x_k + \dots + x_j}$$

**THEOREM.** Quicksort (with 3-way partitioning, randomized) uses

$$N - n + 2QN \text{ compares (where } Q = \sum_{1 \leq k < j \leq n} \frac{p_k p_j}{p_k + \dots + p_j}, \text{ with } p_i = x_i/N)$$

to sort an  $(x_1, \dots, x_n)$ -file, on the average .

## Basic properties of quicksort “entropy”

$$Q = \sum_{1 \leq k < j \leq n} \frac{p_k p_j}{p_k + \dots + p_j} \quad \text{with } p_i = x_i / N$$

Example: all frequencies equal ( $p_i = 1/n$ )

$$Q = \sum_{1 \leq k < n} \frac{1}{n} \sum_{k < j \leq n} \frac{1}{j - k + 1} = \ln n + O(1)$$

Conjecture:  $Q$  maximized when all keys equal?

**NO:**

$$Q = .4444... \text{ for } x_1 = x_2 = x_3 = N / 3$$

$$Q = .4453... \text{ for } x_1 = x_3 = .34N, x_2 = .32N$$

## Upper bound on quicksort "entropy"

$$Q = \sum_{1 \leq k < j \leq n} \frac{p_k p_j}{p_k + \dots + p_j}$$

1. Separate double sum

$$Q = \sum_{1 \leq k < n} p_k \sum_{k < j \leq n} \frac{p_j}{p_k + \dots + p_j}$$

2. Substitute  $q_{ij} = (p_i + \dots + p_j)/p_i$  (note:  $1 = q_{ii} \leq q_{i(i+1)} \leq \dots \leq q_{in} < 1/p_i$ )

$$Q = \sum_{1 \leq k < n} p_k \sum_{k < j \leq n} \frac{q_{kj} - q_{k(j-1)}}{q_{kj}}$$

3. Bound with integral

$$Q = \sum_{1 \leq k < n} p_k \int_{q_{kk}}^{q_{kn}} \frac{1}{x} dx < \sum_{1 \leq k < n} p_k \ln q_{kn} < \sum_{1 \leq k \leq n} p_k (-\ln p_k) = H \ln 2$$

## Quicksort is optimal

The average number of compares per element  $C/N$  is always within a constant factor of the entropy  $H$

lower bound:  $C > NH - N$  (information theory)

upper bound:  $C < 2\ln 2NH + N$  (Burge analysis, Melhorn bound)

No comparison-based algorithm can do better.

**Conjecture:** With sampling,  $C/N \rightarrow H$  as sample size increases.



# Extensions and applications

## Optimality of Quicksort

- ◇ underscores intrinsic value of algorithm
- ◇ resolves basic theoretical question

Analysis shows Quicksort to be sorting method of choice for

- ◇ randomly ordered keys, abstract compare
- ◇ small number of key values

**Extension 1:** Adapt for varying key length`

## Multikey Quicksort

`SORTING` method of choice:  $(Q/H)N \lg N$  byte accesses

**Extension 2:** Adapt algorithm to searching

## Ternary search trees (TSTs)

`SEARCHING` method of choice:  $(Q/H) \lg N$  byte accesses

Both conclusions validated by

- ◇ Flajolet, Clément, Valeé analysis
- ◇ practical experience

## References

- Allen and Munro, Self-organizing search trees, *JACM*, 1978
- Hoare, Quicksort, *Computer Journal*, April 1962
- Clampett, Randomized binary searching with trees, *CACM*, March 1964
- Knuth, *The Art of Computer Programming*, vol. 3, Addison-Wesley, 1975
- Sedgewick, Quicksort with equal keys, *SICOMP*, June 1977
- Wegner, Quicksort for equal keys, *IEEE Trans. on Computers*, April 1985
- Bentley and McIlroy, Engineering a sort function, *Software Practice and Experience*, Jan. 1993
- Bentley and Sedgewick, Sorting/searching strings, *SODA*, January 1997
- and Dr. Dobbs Journal, April and November, 1998
- Clement, Flajolet, and Vallee, Analysis of Tries, *Algorithmica*, 1999