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!
(old: N lg N
(new: (2N lg 2N)/2 = N lg N + N)

Other compelling reasons to study sorting

Simple fundamental algorithms: the ultimate portable software
- intellectual challenge of basic research
- rebuild obsolete libraries
- cope with new languages, machines, and applications
- need to sort twice as much on new machine
- coping with wait longer, even to sort twice as much, on new machine

Other compelling reasons to study sorting
cope with new languages, machines, and applications
- need to sort twice as much on new machine
- annoying to wait longer, even to sort twice as much, on new machine
- need faster sorts every 18 months!

Sedgewick's Corollary: Need Faster Sorts every 18 months!

- problem size expands to fill memory
- memory capacity doubles every 18 months
- but also:

MOORE'S LAW: Processing Power Doubles every 18 months

MOTIVATION
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);
}

Quicksort

Detail (?): How to handle keys equal to the partitioning element
**METHOD A**: Put equal keys all on one side?

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

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

- NO: quadratic for $n=2$

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

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

```
4 9 4 9 4 4 4 9 4 4 1 4
4 9 4 9 4 4 4 9 4 4 1 4
```

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

```
1 4 4 1 4 4 4 1 4 4 4 1
```

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

```
1 4 4 1 4 4 4 1 4 4 4 1
```

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

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

```
4 4 4 4 4 4 4 4 4 4 4 4
```

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

```
4 4 4 4 4 4 4 4 4 4 4 4
```

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

```
4 4 4 4 4 4 4 4 4 4 4 4
```

**Partitioning with equal keys**

How to handle keys equal to the partitioning element?
METHOD C: Stop both pointers on equal keys?

YES: NlgN guarantee for small n, no overhead if no equal keys

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

YES: NlgN guarantee for small n, no overhead if no equal keys

METHOD C: Stop both pointers on equal keys?

How to handle keys equal to the partitioning element?

Partitioning with equal keys
1. Method of choice in practice
   • tiny inner loop, with locality of reference
   • Quicksort common wisdom (last millenium)

2. Equal keys can be handled (with care)
   • NlogN worst-case guarantee, using proper implementation
   • but use a radix sort for small number of key values
   • NlogN worst-case, guarantee randomized

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 subtrees
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
   - Equal keys abound in practice

2. 3-way partitioning is the method of choice.
   - No need for separate radix sort
   - Easy to adapt to multikey sort
   - Greatly expands applicability, with little overhead

3. Average case analysis already done!

Burge, 1975
Sedgewick, 1978
Allen, Munro, Melhorn, 1978
Bentley-McIlroy 3-way partitioning

<table>
<thead>
<tr>
<th>greater</th>
<th>equal</th>
<th>less</th>
</tr>
</thead>
</table>

**Partitioning invariant**

- Swap equals to center after partition
- If right element equal, exchange to right end
- If left element equal, exchange to left end
- Exchange
- Stop if pointers have crossed
- Move from right to find an element that is not greater
- Move from left to find an element that is not less

**Key Features**

- Only one "extra" exchange per equal key
- No extra overhead if no equal keys
- Always uses N-1 (three-way) compares

- Moves from left to find an element that is not greater
- Moves from right to find an element that is not less
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);
}

Quicksort with 3-way partitioning
Information-theoretic lower bound

**Definition**: An \((x_1, x_2, \ldots, x_n)-\text{file}\) has \(N\) keys, \(n\) distinct key values, with
\[ x_i = \text{number of occurrences of the } i\text{-th smallest key} \]
\[ N = x_1 + x_2 + \cdots + x_n \text{ keys}, \]
\[ N! = \prod_{i=1}^{N} x_i \]
\[ \prod_{i=1}^{N} x_i! \]

**Theorem**: Any sorting method uses at least \(N - N^H\) compares (where \(H = - \sum_{k=1}^{n} p_k \log p_k\) is the entropy).

**Note**: An \((x_1, x_2, \ldots, x_n)-\text{file}\) has
Information-theoretic lower-bound proof

\[ \sum_{x_1 \in \mathbb{X}} \frac{1}{x_1} = \log N - N \log N \]

By Stirling's approximation,

\[ \log N = \frac{N \log N - N \log \log x + \log x}{x} \]

Avg. number of compares is minimized when tree is balanced.

Number of leaves must exceed number of possible files.

Decision tree describes all possible sequences of comparisons.
Analysis of Quicksort with equal keys

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

II. Multiply both sides by $N$ to get:

$$\sum_{i=1}^{n} N \sum_{j=1}^{n} C(i, j) x_i x_j = N \sum_{i=1}^{n} (i - N) x_i x_j$$

III. Subtract same equation for $x_1, \ldots, x_n$ and let $D(1, n) = C(1, n) - C(2, n)$.

IV. Subtract same equation for $x_1, \ldots, x_{n-1}$.

V. Multiply both sides by $N$ to get:

$$\sum_{i=1}^{n} N \sum_{j=1}^{n} C(i, j) x_i x_j = N \sum_{i=1}^{n} (i - N) x_i x_j$$

VI. Subtract same equation for $x_1, \ldots, x_{n-1}$.

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

VIII. Multiply both sides by $N$ to get:

$$\sum_{i=1}^{n} N \sum_{j=1}^{n} C(i, j) x_i x_j = N \sum_{i=1}^{n} (i - N) x_i x_j$$

IX. Subtract same equation for $x_1, \ldots, x_{n-1}$.

X. Multiply both sides by $N$ to get:

$$\sum_{i=1}^{n} N \sum_{j=1}^{n} C(i, j) x_i x_j = N \sum_{i=1}^{n} (i - N) x_i x_j$$

1. Multiply both sides by $N$ to get:

$$\sum_{i=1}^{n} N \sum_{j=1}^{n} C(i, j) x_i x_j = N \sum_{i=1}^{n} (i - N) x_i x_j$$

2. Subtract same equation for $x_1, \ldots, x_{n-1}$.

3. Multiply both sides by $N$ to get:

$$\sum_{i=1}^{n} N \sum_{j=1}^{n} C(i, j) x_i x_j = N \sum_{i=1}^{n} (i - N) x_i x_j$$

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

5. Multiply both sides by $N$ to get:

$$\sum_{i=1}^{n} N \sum_{j=1}^{n} C(i, j) x_i x_j = N \sum_{i=1}^{n} (i - N) x_i x_j$$

6. Subtract same equation for $x_1, \ldots, x_{n-1}$.
Analysis of Quicksort with equal keys (cont.)

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

\[ C(1,n) = N - n + 2N \]

comparisons (where \( N \) compares on the average).

6. Telescope (twice)

\[ \sum_{\pi \in \mathcal{P}} \frac{1}{\pi^2} \left( x_1 + \cdots + x_n \right) \]

5. Simplify, divide both sides by \( N \) = \( x_1 + \cdots + x_n \)

\[ C(1,n) = D(1,n) + D(I',n - I) \]

(\( n \)-way partitioning, randomized) uses...
Basic properties of quicksort "entropy"

Conjecture: $Q$ maximized when all keys equal?

Example: all frequencies equal ($p_i = \frac{1}{n}$)

$$Q = \sum_{i=1}^{n} \left( \sum_{j=i+1}^{n} \sum_{k=j+1}^{n} \frac{1}{\ln(n) + O(1)} \right)$$

$$Q = \frac{1}{N!} \sum_{p_1, \ldots, p_N} \left( \sum_{k=j=n}^{p_k} \frac{1}{p_k!} \right)$$
Upper bound on quicksort "entropy"
Quicksort is optimal

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

$C/N > NH - N$ (information theory)

No comparison-based algorithm can do better.

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

upper bound: $C > 2LN2N + N$ (Burge analysis, Melhorn bound)
lower bound: $C < N(2LN + N)$ (information theory)

within a constant factor of the entropy $H$
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: \((Q/H)N\log N\) 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, Clèment, Valeé analysis
- practical experience
Allen and Munro, Self-organizing search trees, JACM, 1978

Climent, Flajolet, and Vuilleme, Analysis of Tries, Algorithmica, 1999

and Dr. Dobbs Journal, April and November, 1998

Bentley and Sedgewick, Sorting/searching strings, SODA, January 1997

Software Practice and Experience, Jan. 1993

Bentley and McIlroy, Engineering a sort function,

Wegner, Quicksort for equal keys, IEE Trans. on Computers, April 1985

Sedgewick, Quicksort with equal keys, SICOMP, June 1977


Clepeit, Randomized binary searching with trees, CACM, March 1964

Hoare, Quicksort, Computer Journal, April 1962

Allen and Munro, Self-organizing search trees, JACM, 1978