NEW RESEARCH
on
THEORY and PRACTICE
of
SORTING and SEARCHING

R. Sedgewick
Princeton University

J. Bentley
Bell Laboratories

Context

Layers of abstraction in modern computing

- Applications
- Programming Environment
- Algorithm Implementations
- Operating System
- Hardware

Ongoing research and development at all levels

Sorting and searching

- fundamental algorithms
- still the bottleneck in modern applications
- primitive in modern programming environments
- methods in use based on 1970s research

BASIC RESEARCH on algorithm analysis

Motivation

MOORE'S LAW: Processing Power Doubles every 18 months

similar maxims:

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

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

- sorts take longer to complete on new processors

  old: $N \lg N$

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

Other compelling reasons to study sorting

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

Simple fundamental algorithms: the ultimate portable software

Quicksort

Recursive procedure based on PARTITIONING

to PARTITION an array, divide it so that

- some element a[i] is in its final position
- no larger element left of i
- no smaller element right of i

After partitioning, sort the left and right parts recursively

PARTITIONING METHOD:

- pick a partitioning element
- scan from right for smaller element
- scan from left for larger element
- exchange
- repeat until pointers cross

Quicksort example

A	S	O	R	T	I	N	G	E	X	A	M	P	L	E
A	A	E	E	T	I	N	G	O	X	S	M	P	L	R
A	A	E												
A	A													
A														
				L	I	N	G	O	P	M	R	X	T	S
				L	I	G	M	O	P	N				
				G	I	L								
					I	L								
					I									
								N	P	O				
									O	P				
										P				
												S	T	X
													T	X
													T	
A	A	E	E	G	I	L	M	N	O	P	R	S	T	X

Partitioning examples

A S O R T I N G E X A M P L E

A S

A M P L

A A S M P L E

O

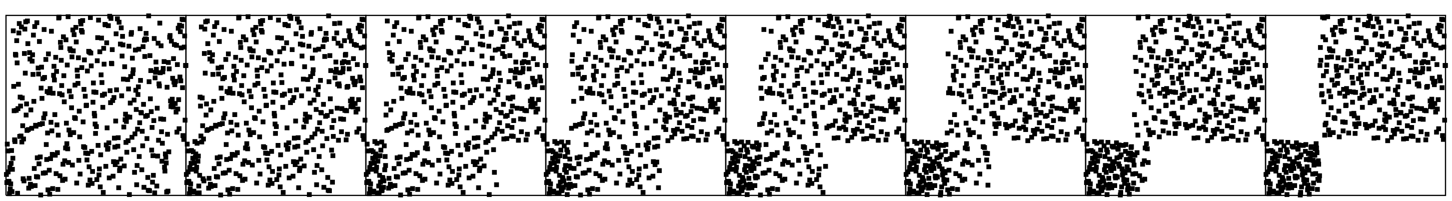
E X

A A E O X S M P L E

R

E R T I N G

A A E E T I N G O X S M P L R



Partitioning implementation

Use Item to embody records-with-keys abstraction

- less: compare two keys
- exch: exchange two records

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

Detail (?)

- how to handle equal keys [stay tuned]

Quicksort implementation

```
quicksort(Item a[], int l, int r)
  { int i;
    if (r > l)
      {
        i = partition(a, l, r);
        quicksort(a, l, i-1);
        quicksort(a, i+1, r);
      }
  }
```

Issues

- overhead for recursion?
- small files
- running time depends on input
- worst-case time cost (quadratic, a problem)
- worst-case space cost (linear, a serious problem)

Quicksort analysis (distinct keys)

BEST case: split in the middle, O(N lg N) compares

- C(N) = N + 2 C(N/2)

WORST case: split at one end, O(N^2) compares

- C(N) = C(N-1) + N

AVERAGE case: split at random position, ~2 N ln N compares

- C(N) = N + 2 (C(0) + ... + C(N-1))/N

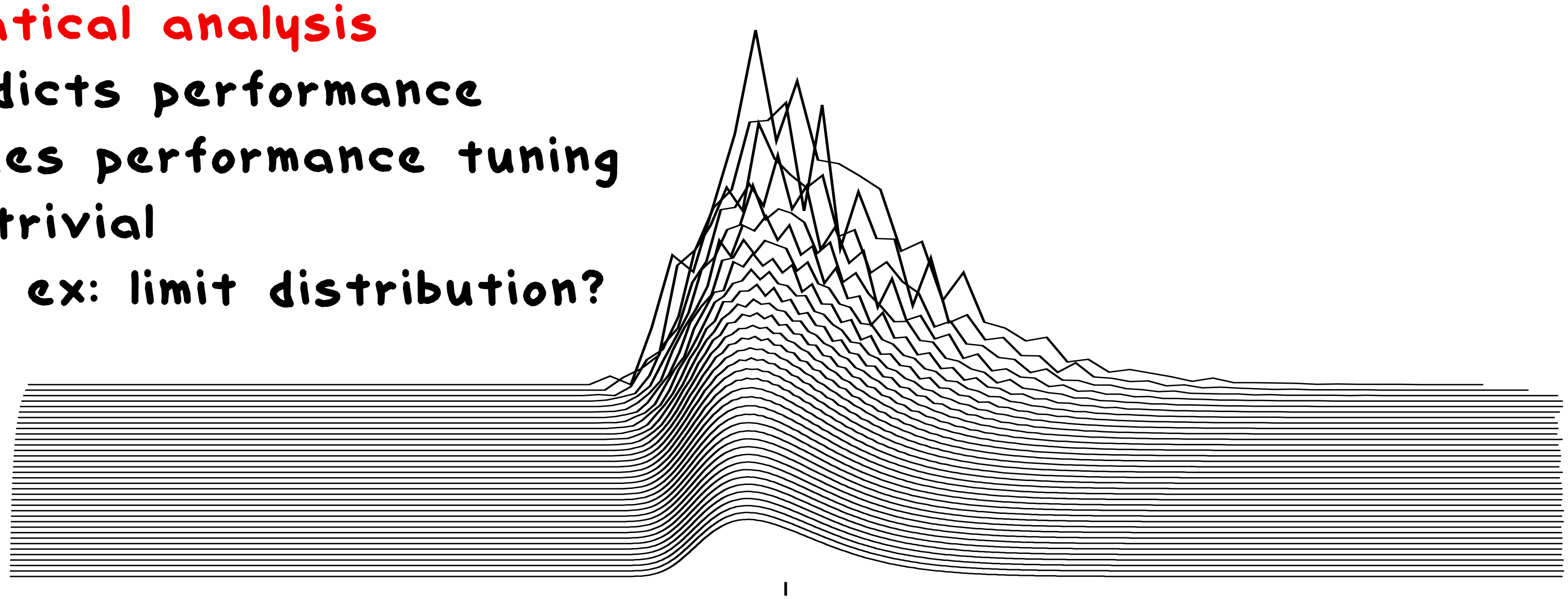
Defense against worst case:

- choose random partitioning element
- N log N randomized algorithm (Hoare, 1960)

Mathematical analysis

- predicts performance
- guides performance tuning
- nontrivial

ex: limit distribution?



Quicksort with equal keys

N keys, n distinct key values, N >> n
How to handle keys equal to PE?
DANGER: quadratic performance pitfalls

Method A: Put equal keys all on one

.   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?

.   1 4 1 4 1 4 1 4 4
.   1 4 1 4 1 4 1 4 4
.   1 1 1 4 1 4 4

NO: quadratic for n = 2 (linear for n = 1)

recursion GUARANTEES that above cases WILL occur for small n
randomization provides NO protection (!!)

Quicksort with equal keys (continued)

Method C. special case for small n?

- guaranteed O(N) for small n
- O(N) overhead even if no equal keys

Method D. stop both pointers on equal keys?

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

- guaranteed O(N lg N) for small n
- no overhead if no equal keys
- state of the art for library qsorts (through 1990s)

Not all library qsorts use Method D

Run qsort on huge file with two different keys

- doesn't finish: A or B
- quick: C
- immediate: D

Can be inhibiting factor in library utility

Three-way partitioning

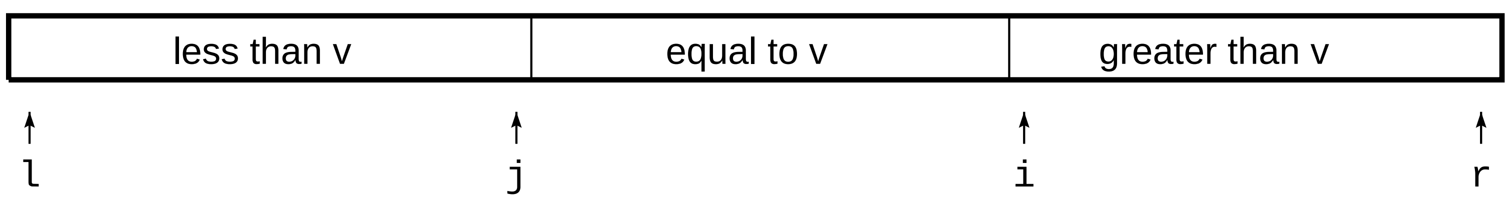
PROBLEM: Sort files with 3 distinct key values

Natural and appealing problem

- Hoare, 1960
- Dijkstra, ``Dutch National Flag Problem``

Immediate application to quicksort

- put ALL keys equal to the PE into position



Early solutions cumbersome and/or expensive

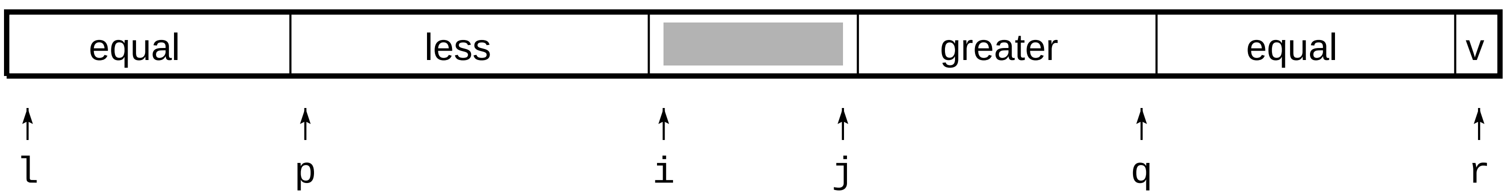
- not used in practical sorts before mid-1990s

Bentley-McIlroy three-way partitioning (1993)

FOUR-part partition

- some elements between i and j equal to v
- no larger element left of i
- no smaller element right of j
- more elements between i and j equal to v

Swap equal keys into center



All the right properties

- easy to implement
- linear if keys all equal
- no extra compares if no equal keys (always N-1)

Expands utility of system qsort

- old: N lg N (or quadratic!) for small n
- new: LINEAR for small n

Three-way partitioning implementation

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

Analysis of 3-way partitioning

Average running time of Quicksort with 3-way partitioning?

Empirical studies (Bentley, 1993)

- LINEAR number of compares for small n

ONE key value

- N - 1 compares

TWO key values: x_1 instances of first, x_2 instances of second

- with probability x_1/N: $(N-1) + (x_2-1)$ compares
- with probability x_2/N: $(N-1) + (x_1-1)$ compares
- total avg:

$$N - 2 + 2 x_1 x_2 / N$$

max at $x_1 = x_2$: $1.5 N - 2$

THREE key values

- [analysis looks complicated]

Detailed analysis of 3-way partitioning

Burge (1975): analysis of search trees with equal keys
Sedgewick (1975): lower bound on Quicksort with equal keys

- n distinct key values
- x_i instances of key i, for i from 1 to n
- $x_1 + x_2 + \dots + x_n = N$

THM: Average number of compares is

$$C = N - n + 2\,Q\,N$$

Q is "Quicksort entropy"

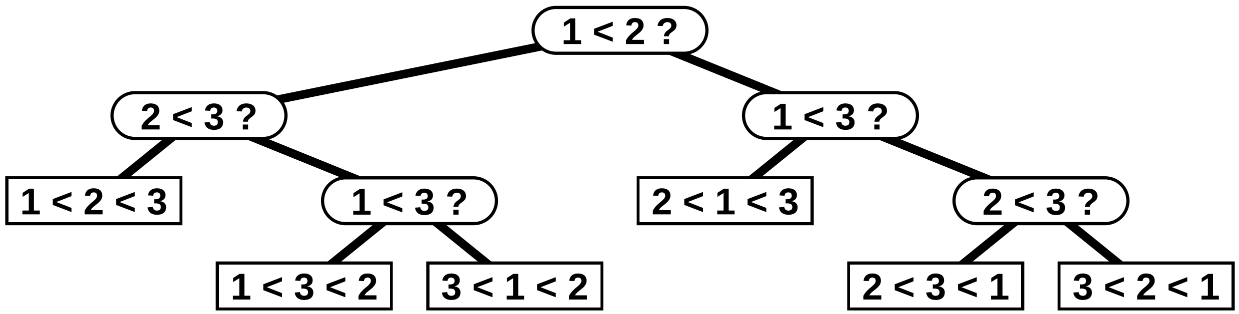
- $p_i = x_i/N$ (convert to probabilities)
- $q_{ij} = p_i\,p_j/(p_i + \dots p_j)$
- $r_{ij} = q_{ij} + \dots + q_{jj}$
- $Q = r_{1n} + r_{2n} + \dots + r_{nn}$

Ex: x_i all equal (to N/n)

- $p_i = 1/n$
- $q_{ij} = (1/n)(1/(i-j+1))$
- $r_{ij} = (1/n)(1 + 1/2 + \dots + 1/(i-j+1))$
- $Q = \ln n + O(1)$
- $C = 2\,N \ln n + O(N)$

Information-theoretic sorting lower bound

DECISION TREE describes all possible sequences of compares



number of leaves > $N!/(x_1!\ x_2!\ x_3!\ \ldots\ x_n!)$ [multinomial coefficient]

take lg for bound on compares

- $C > \lg N! - \lg x_1! - \ldots - \lg x_n!$
- $C > N \lg N - N - x_1 \lg x_1 - \ldots - x_n \lg x_n$

  (Stirling's approximation)

ENTROPY:

- $H = (x_1/N)\lg(N/x_1) + \ldots + (x_n/N)\lg(N/x_n)$
- $N\ H = N \lg N - x_1 \lg x_1 - \ldots - x_n \lg x_n$

THM: $C > N\ H - N$

Entropy comparison

Relationship between Q and H??

Standard entropy H

- equal to lg n if all freqs equal
- maximized when all freqs equal (H never exceeds lg n)

"Quicksort entropy" Q

- approaches ln n if all freqs equal
- NOT maximized when all freqs equal

Ex: $x_1 = x_2 = x_3 = N/3$

- $Q = .4444...$

Ex: $x_1 = x_3 = .34N$, $x_2 = .32N$

- $Q = .4453...$

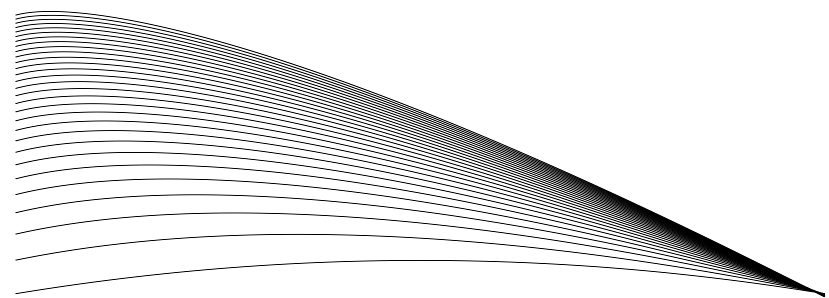
Entropy comparison (continued)

Ex: x2 through xn all equal

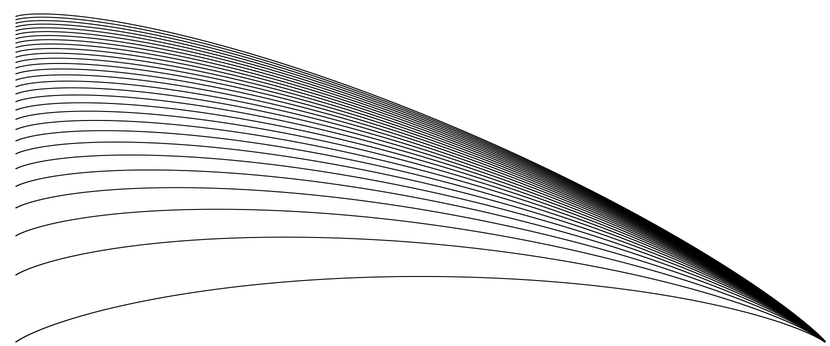
horizontal axis: x1 (ranges from 0 to N)

N = 512, curve for each n from 2 to 30

"Quicksort entropy" Q



Standard entropy H



General result relating Q and H?

- answer found in basic research by Melhorn (1978)

Quicksort is optimal

"Quicksort entropy" function arises in analysis
of "self-organizing" binary search trees

- Allen and Munro, 1978

THM (Melhorn, 1978): $Q < (\ln 2)\ H$

THM (1999): Quicksort is optimal (!)

Proof:

$$N\ H - N < C < (2 \ln 2)\ N\ H + N$$

[C grows asymptotically with NH]

conjecture: with sampling, C*/NH -> 1

NO sorting method can use fewer compares (asymptotically)
for ANY distribution of key values

Extensions and applications

Optimality of Quicksort
- underscores intrinsic value of algorithm
- resolves basic theoretical question
- analysis shows qsort to be sorting method of choice for
  - randomly ordered keys, abstract compare
  - small number of key values

Real-world applications
- nonuniform key values?
- varying key length?
- arbitrary distribution?

Extension 1: Adapt for varying key length
- Multikey Quicksort
- SORTING method of choice

Extension 2: Adapt algorithm to searching
- Ternary search trees
- SEARCHING method of choice

MSD radix sort

Sort files where keys are sequences of BYTES

- each byte has value less than M
- typical: group of bits

METHOD:

- Partition file into M buckets

  all keys with first byte 0

  all keys with first byte 1

  all keys with first byte 2

  ...

  all keys with first byte M-1
- Sort M pieces recursively

Tradeoff

- large M: space for buckets (too many empty buckets)
- small M: too many passes (too many keys per bucket)

MSD radix sort potential fatal flaw

each pass ALWAYS takes time proportional to N+M

- initialize the buckets
- scan the keys

Ex: (ASCII bytes) M = 256

- 100 times slower than insertion sort for N = 2

Ex: (UNICODE) M = 65536

- 30,000 times slower than insertion sort for N = 2

TOO SLOW FOR SMALL FILES

recursive structure GUARANTEES sort is used for small files

Solution: cut to insertion sort for small files

Practical problems for library sort

- choice of radix
- cutoff point
- nonuniformity in keys

Three-way radix Quicksort

PROBLEM:

- long keys that differ slightly can be costly to compare
- this is the common case!

```
absolutism
absolutely
```

SOLUTION:

- Do three-way partitioning on key characters
- Sort three parts recursively
  (increment char ptr on middle subfile)

Ex: N records with huge (w-byte) keys

- Byte comparisons for pointer sort
  MSD radix sort: Nw
  3-way radix quicksort: 2 N ln N
- SUBLINEAR sort

Multikey Quicksort

- same algorithm, keys are VECTORS
- Unicode (16-bit chars) blurs distinction

String sort example

actinian	coenobite	actinian
jeffrey	conelrad	bracteal
coenobite	actinian	coenobite
conelrad	bracteal	conelrad
secureness	secureness	cumin
cumin	dilatedly	chariness
chariness	inkblot	centesimal
bracteal	jeffrey	cankerous
displease	displease	circumflex
repertoire	repertoire	repertoire
dourness	dourness	dourness
centesimal	southeast	southeast
dilatedly	cumin	secureness
inkblot	chariness	dilatedly
southeast	centesimal	inkblot
cankerous	cankerous	jeffrey
circumflex	circumflex	displease

Perspective on radix sorting

Three-way radix quicksort

- blends quicksort and MSD radix sort

quicksort

- leading part of keys used in all compares
- short inner loop otherwise

MSD radix sort

- empty bins on small files
- adapts poorly to variable-length keys
- long inner loop

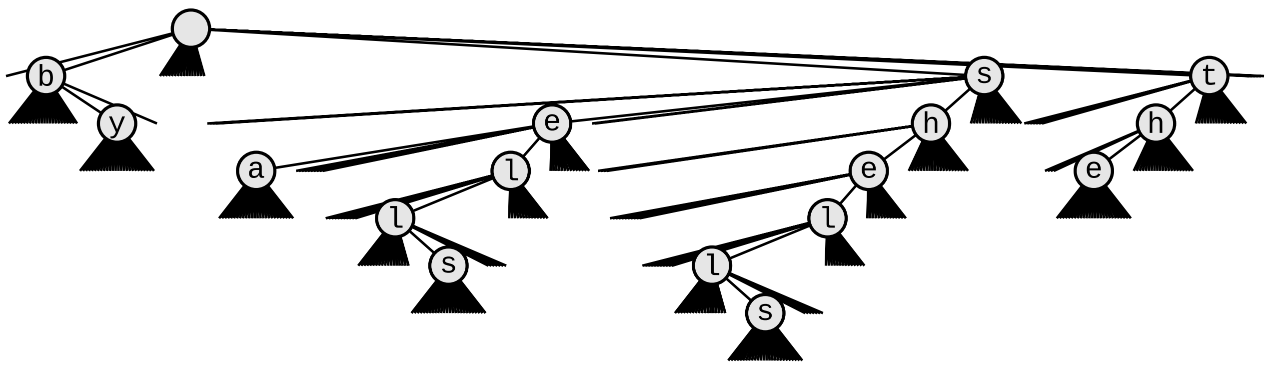
Three-way radix quicksort

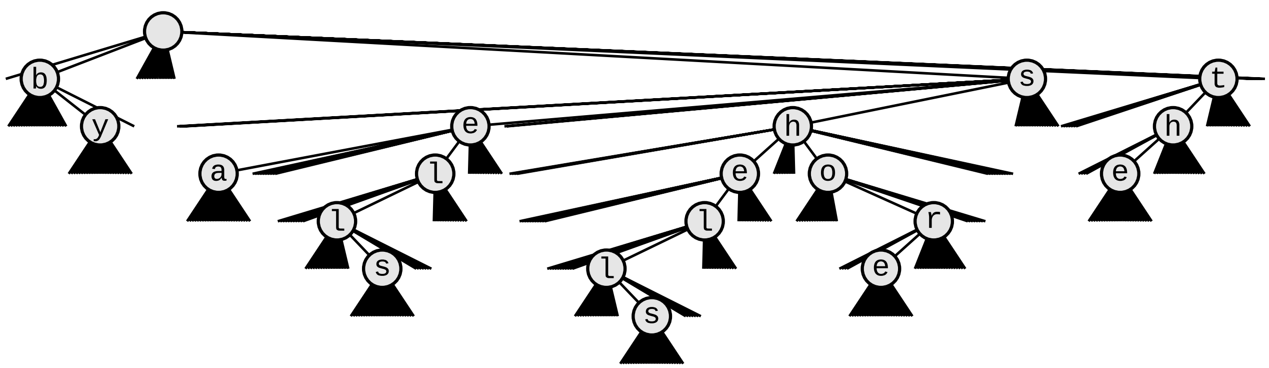
- compares characters, not strings
- short inner loop
- adapts to multikey
- METHOD of CHOICE for sorting long keys
  - easy to implement
  - works well on nonuniform keys
  - fastest in practice

M-way trie

SEARCH data structure corresponding to MSD radix sort

Nodes contain characters/links to implement M-way branching





M-way trie analysis

Assumptions

- N keys, total of C characters in keys
- approx. N trie nodes (or more, details omitted)
- M links per node

Space: N*M + C

Time: lgN/lgM CHARACTER comparisons (constant in practice)

Ex: M=26, N=20000

520,000 links, tree height 3-4

Ex: M=16, N=1M

16M links, tree height 5

Faster than hashing

- successful search: no arithmetic
- unsuccessful search: don't need to examine whole key

DRAWBACKS

- good implementation nontrivial
- too much space for null links

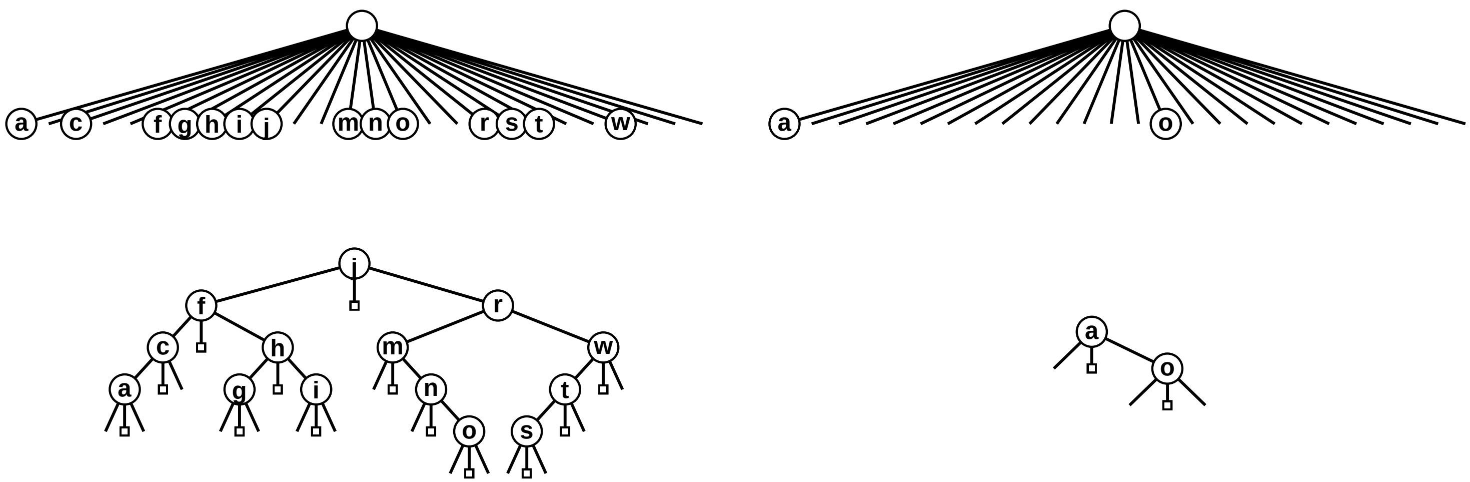
Ternary search trees (TSTs)

Search algorithm corresponding to 3-way radix Quicksort

Nodes contain characters and links for three-way branching

- left: key character less
- middle: key character equal
- right: key character greater

Equivalent to TRIE with BST implementation of trie nodes



TST implementation

Search algorithm writes itself

```
int RSTsearchR(RSTptr x, char *v)
  {
    if (x == NULL) return 0;
    if ((*v == ' ') && (x->ch == ' ')) return 1;
    if (*v < x->ch)
      return RSTsearchR(x->l, v);
    if (*v == x->ch)
      return RSTsearchR(x->m, v+1);
    if (*v > x->ch)
      return RSTsearchR(x->r, v);
  }
```

Optimal (fully balanced) tree

- SUCCESSFUL search: lg N + [key length] character compares
- UNSUCCESSFUL search: lg N character compares

Idea dates at least to 1962

- practical impact unnoticed until late 1990s
- casualty of compare abstraction

Perspective on radix searching

TSTs blend binary search trees (BSTs) and tries

BSTs (correspond to Quicksort)

- leading part of keys always used in compares
- short inner loop otherwise

tries

- too many null links for large radix
- long inner loop for small radix

TSTs

- compares characters, not strings
- equivalent to using BSTs for trie nodes
- automatically adapts radix to keys
- METHOD of CHOICE for searching
  - faster than hashing
  - gracefully grows and shirnks
  - support partial match, near-neighbor search, ...

AVERAGE-CASE ANALYSIS?

TST and multikey quicksort analysis

Clement, Flajolet, Valle (1999)

- unifies classical tree/trie analyses
- generalizes to nonuniform models
- extends to cover TSTs
- exploits powerful tools
  - generalized Ruelle operators
  - Mellin transforms

Eight theorems

- algebraic and asymptotic analysis
- Poisson and Bernoulli models
- path lengths and height

THM: Asymptotic TST search cost: (Q/H) lg N

Open problems

- TST height?
- concentration of distribution?
- limit distributions?

Perspective

New research on fundamental algorithms

- 3-way quicksort
  - method of choice for small keys
- multikey quicksort
  - method of choice for large keys
- TSTs
  - searching method of choice

Direct practical impact

- new applications demand fast algorithms
- new algs improve performance for all apps

old basic research results establish optimality of new algs

Deep new theory analyzes new algorithms

- predict performance
- set parameters

Future challenges

- similar refinements for other classic fundamental algorithms

partial BIBLIOGRAPHY

Allen and Munro, Self-organizing search trees

- JACM, 1978

Hoare, Quicksort

- Computer Journal, April 1962

Clampett, Randomized binary searching with trees

- CACM, March 1964

devroye, A probabilistic analysis of the height of tries

- Acta Informatica, 1984

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
- Dr. Dobbs Journal, April and November, 1998

Clement, Flajolet, and Vallee, Analysis of Tries

- Algorithmica, 1999

Average number of compares for QUICKSORT with distinct keys

Recurrence from recursive program

$$C_N = N - 1 + \frac{1}{N}\sum_{1\le j\le N}(C_{j-1} + C_{N-j})$$

Change j to $N+1-j$ in second sum

$$C_N = N - 1 + \frac{2}{N}\sum_{1\le j\le N}C_{j-1}.$$

Multiply both sides by N

$$NC_N = N(N-1) + \frac{2}{N}\sum_{1\le j\le N}C_{j-1}.$$

Subtract same equation for $N-1$

$$NC_N - (N-1)C_{N-1} = 2N + 2C_{N-1}$$

Rearrange terms

$$NC_N = (N+1)C_{N-1} + 2N$$

Divide by $N(N+1)$

$$\frac{C_N}{N+1} = \frac{C_{N-1}}{N} + \frac{2}{N+1}$$

Telescope

$$\frac{C_N}{N+1} = 2(H_{N+1} - 1)$$

Approximate

$$C_N \approx 2N\ln N$$

Average number of compares for QUICKSORT with equal keys

Recurrence for average number of comparisons

$$C(x_1,\ldots,x_n) = N + 1 + \frac{1}{N}\sum_{1\le j\le N} x_j\big(C(x_1,\ldots,x_{j-1}) + C(x_{j+1}\ldots x_n)\big)$$

Multiply both sides by $N = x_1 + \ldots + x_n$

$$NC(x_1,\ldots,x_n) = N(N-1) + \sum_{1\le j\le N} x_j C(x_1,\ldots,x_{j-1}) + \sum_{1\le j\le N} x_j C(x_{j+1},\ldots,x_n).$$

Subtract same equation for $x_2,\ldots,x_n$ **(with** $D(x_1\ldots x_n) \equiv C(x_1,\ldots,x_n) - C(x_2,\ldots,x_n)$**)**

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

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

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

Simplify, divide by N

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

Telescope (twice)

$$C(x_1,\ldots,x_n) = N - n + 2\sum_{1\le k\le j\le n}\frac{x_kx_j}{x_k+\ldots+x_j}$$

Upper bound on QUICKSORT entropy

Quicksort entropy definition

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

Separate double sum

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

Substitute $q_{ij} = (p_i + \ldots + p_j / p_i)$ **(note:** $1 = q_{ii} \le q_{i(i+1)} \le \ldots \le q_{in} < 1/p_i$**)**

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

Bound with integral

$$Q < \sum_{1 \le k < n} p_k \int_{q_{kk}}^{q_{kn}} \frac{1}{x} dx$$

Simplify

$$Q < \sum_{1 \le k < n} p_k \ln q_{kn} \le \sum_{1 \le k < n} p_k (-\ln p_k) = H \ln 2$$