What do we know about Quicksort?

a brief summary of facts learned since the 1960s

Robert Sedgewick
Princeton University

What do we know about Quicksort?

C.A.R. Hoare invented it in 1959 and wrote a definitive paper introducing it in 1962.

"*There's one good thesis left in Quicksort*" [Knuth, 1973]

"*I hope this is the last paper we see about Quicksort*" [Anonymous, 1975]

This talk: Some important facts learned since the 1960s

- Students should read Hoare's paper (still).
- Randomization is critical (and must be done carefully).
- Attention must be paid to equal keys.
- Quicksort is optimal.
- Substantial improvents are possible for a common key type.
- The limiting distribution is *not* normal
- Multiway partitioning is effective on modern machines.
- Quicksort remains the method of choice.

next talk

Quicksort

By C. A. R. Hoare

A description is given of a new method of sorting in the random-access store of a computer. The method compares very favourably with other known methods in speed, in economy of storage, and in ease of programming. Certain refinements of the method, which may be useful in the optimization of inner loops, are described in the second part of the paper.

Part One: Theory

The sorting method described in this paper is based on the principle of resolving a problem into two simpler subproblems. Each of these subproblems may be resolved to produce yet simpler problems. The process is repeated until all the resulting problems are found to be trivial. These trivial problems may then be solved by known methods, thus obtaining a solution of the original more complex problem.

Partition

The problem of sorting a mass of items, occupying consecutive locations in the store of a computer, may be reduced to that of sorting two lesser segments of data, provided that it is known that the keys of each of the items held in locations lower than a certain dividing line are less than the keys of all the items held in locations above this dividing line. In this case the two segments may be sorted separately, and as a result the whole mass of data will be sorted.

In practice, the existence of such a dividing line will be rare, and even if it did exist its position would be unknown. It is, however, quite easy to rearrange the items in such a way that a dividing line is brought into existence, and its position is known. The method of doing this has been given the name *partition*. The description given below is adapted for a computer which has an *exchange* instruction; a method more suited for computers without such an instruction will be given in the second part of this paper.

The first step of the partition process is to choose a particular key value which is known to be within the range of the keys of the items in the segment which is to be sorted. A simple method of ensuring this is to choose the actual key value of one of the items in the segment. The chosen key value will be called the *bound*. The aim is now to produce a situation in which the keys of all items below a certain dividing line are equal to or less than the bound, while the keys of all items above the dividing line are equal to or greater than the bound. Fortunately, we do not need to know the position of the dividing line in advance; its position is determined only at the end of the partition process.

The items to be sorted are scanned by two pointers; one of them, the *lower pointer*, starts at the item with lowest address, and moves upward in the store, while the other, the *upper pointer*, starts at the item with the highest address and moves downward. The lower pointer starts first. If the item to which it refers has a key which is equal to or less than the bound, it moves up to point to the item in the next higher group of locations. It continues to move up until it finds an item with key value greater than the bound. In this case the lower pointer stops, and the upper pointer starts its scan. If the item to which it refers has a key which is equal to or greater than the bound, it moves down to point to the item in the next lower locations. It continues to move down until it finds an item with key value less than the bound. Now the two items to which the pointers refer are obviously in the wrong positions, and they must be exchanged. After the exchange, each pointer is stepped one item in its appropriate direction, and the lower pointer resumes its upward scan of the data. The process continues until the pointers cross each other, so that the lower pointer refers to an item in higher-addressed locations than the item referred to by the upper pointer. In this case the exchange of items is suppressed, the dividing line is drawn between the two pointers, and the partition process is at an end.

An awkward situation is liable to arise if the value of the bound is the greatest or the least of all the key values in the segment, or if all the key values are equal. The danger is that the dividing line, according to the rule given above, will have to be placed outside the segment which was supposed to be partitioned, and therefore the whole segment has to be partitioned again. An infinite cycle may result unless special measures are taken. This may be prevented by the use of a method which ensures that at least one item is placed in its correct position as a result of each application of the partitioning process. If the item from which the value of the bound has been taken turns out to be in the lower of the two resulting segments, it is known to have a key value which is equal to or greater than that of all the other items of this segment. It may therefore be exchanged with the item which occupies the highest-addressed locations in the segment, and the size of the lower resulting segment may be reduced by one. The same applies, *mutatis mutandis*, in the case where the item which gave the bound is in the upper segment. Thus the sum of the numbers of items in the two segments, resulting from the partitioning process, is always one less than the number of items in the original segment, so that it is

10

Students should read Hoare's paper (still)

Why? It is a quintessential example of **algorithm science**.

Algorithm science

The application of the scientific method to the design and analysis of algorithms

- Create a mathematical model.
- Develop hypotheses about real-world performance.
- Run experiments to validate the hypotheses.
- Iterate on the basis of facts learned.

The theory of algorithms is typically *not* algorithm science

TofA: "*Running time is* $O(N \log N)$" ✗

AS: "*Running time is* $\sim cN \ln N$ *for some constant* c" ✓

Students should read Hoare's paper (still)

Why? It is a quintessential example of **algorithm science**.

Theorem. Hoare (1961): Quicksort running time for *N* random inputs is

$$\boxed{\sim cN \ln N}$$

where c is a *machine-dependent constant.*

Hypothesis. Doubling *N* will increase the running time by a factor of about $2 + (2 \ln 2)/\ln N$.

Proof: $$\frac{2cN \ln(2N)}{cN \ln N} = 2 + \frac{2 \ln 2}{\ln N}$$

Validation.

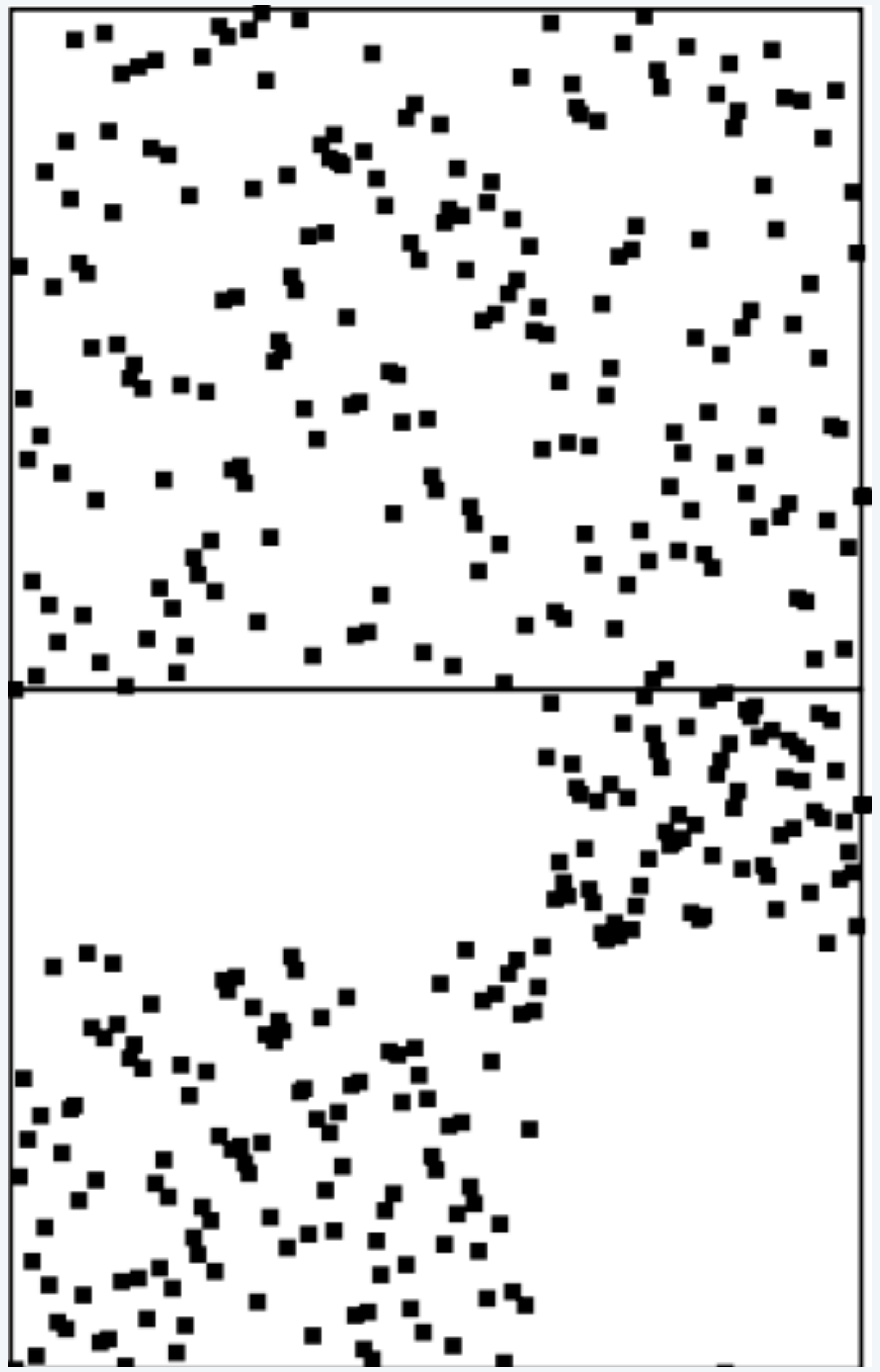
N	T_N	$T_N/T_{N/2}$	$2 + (2 \ln 2)/\ln N$
500	81		
1000	188	2.32	2.12
2000	407	2.16	2.18

Results can be (and have been) validated in countless real-world applications ever since.

Randomization is critical (and must be done carefully).

Analysis assumes

- Partitioning element is chosen at random.
- Keys are distinct.
- Subfiles after partitioning are randomly ordered.



Early (problematic) examples

- Too-clever partitioning on the median of three.
- Partition on the value of a random element.

Developer: Can't randomize—too difficult to test and maintain.

Response: Randomly permute input before the sort.

Bigger problem: Keys are not necessarily distinct in real applications.

Attention must be paid to equal keys

Initial goal: avoid quadratic performance (happens in typical implementations if all keys are equal)

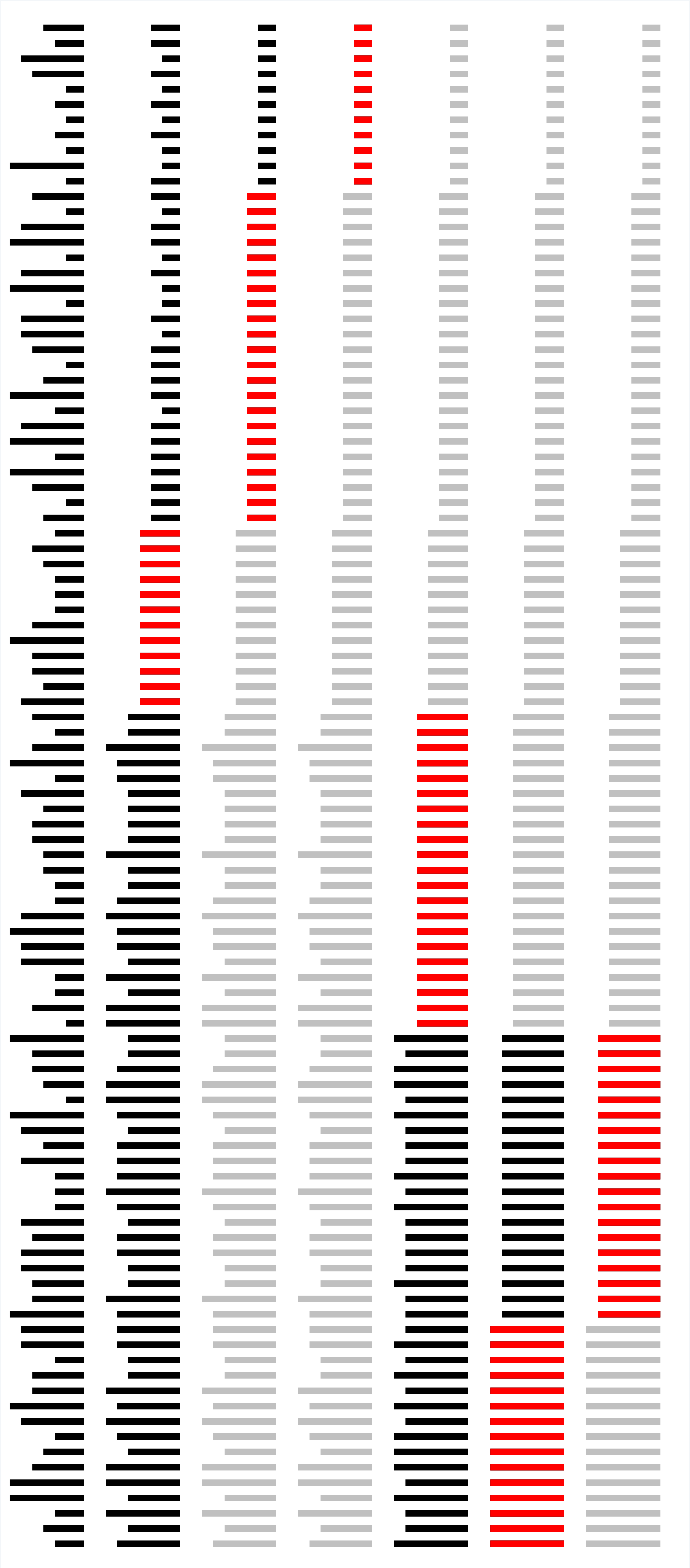
Two easy fixes (RS, 1977)

- Stop pointers on equal keys (trivial to implement).
- Three-way partitioning (worth the expense?).

Elegant fix (Bentley-McIlroy, 1993) ← *in response to a user complaint*

- Swap equal keys to ends, then swap into place.
- Uses just N-1 (three-way) compares.
- Only one "extra" exchange per equal key.
- Still in use today.

```
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 is optimal

Starting point: N keys having n different values with multiplicities $x_1, x_2, \ldots, x_n$.

Assume keys are randomly ordered and that three-way partitioning is used.

Let C be the average number of compares.

Lower bound (information theory): $C > NH - N$ where $H = -\sum_{1 \le i \le n} p_i \lg p_i$ with $p_i = x_i/N$

Exact value (Sedgewick, 1975): $C = N - n + 2QN$ where $Q = \sum_{1 \le k < j \le n} \frac{p_k p_j}{p_k + \ldots + p_j}$

Theorem. [Bentley-Sedgewick] *Average number of compares is within a constant factor of optimal.*

Proof: $Q < H \ln 2$

Conjecture (proven true by Wild in 2015): With sampling, constant approaches 1.

Substantial improvents are possible for a common key type.

Suppose that keys are strings (sequences of characters)

3-way string Quicksort (Bentley-Sedgewick, 1997)

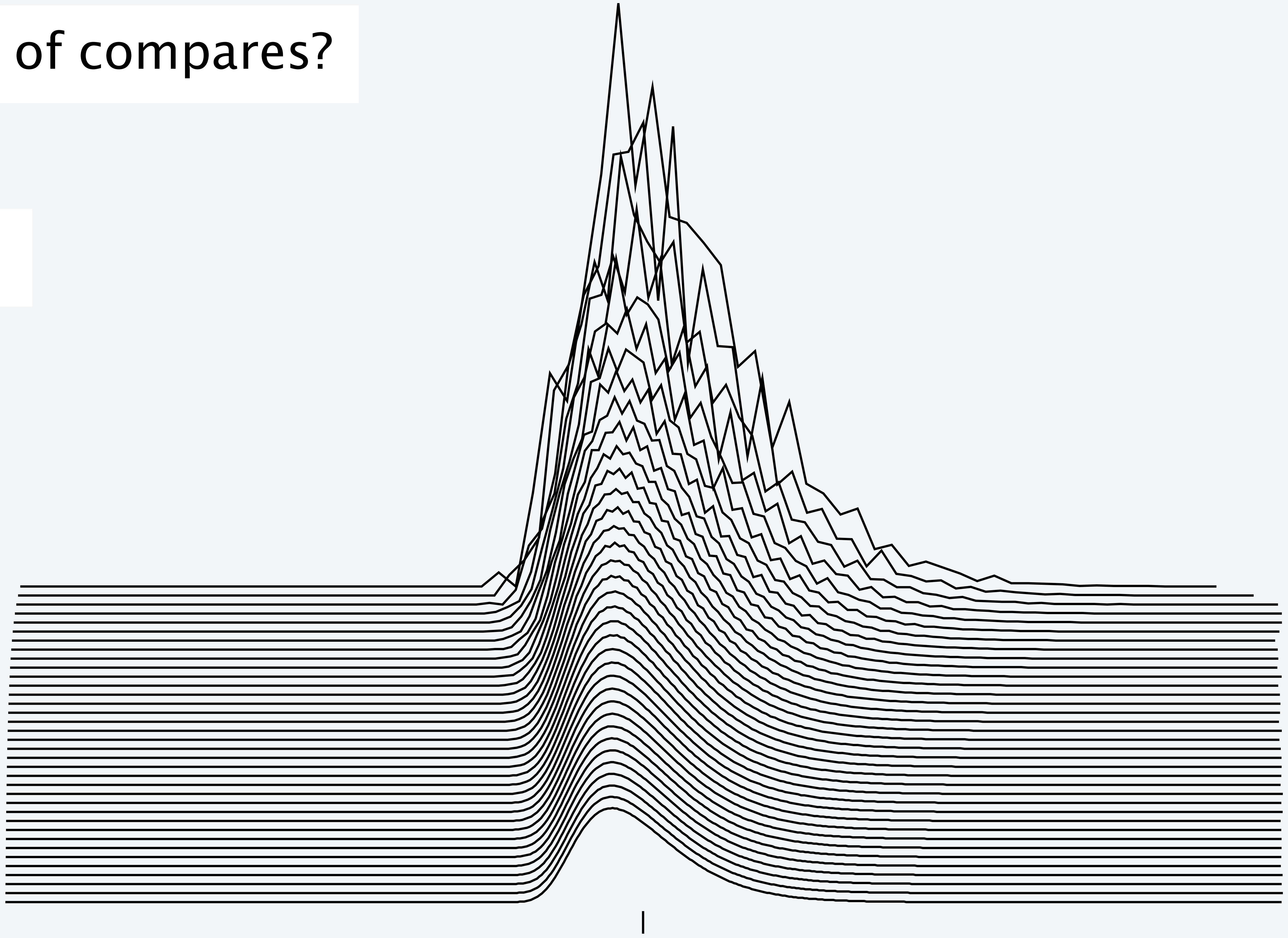
- 3-way partition on first character of each key.
- Recursively sort 3 subfiles.
- Use substring excluding first char in the middle.
- Sorts random strings with 2N ln N *character* compares
- Optimal

Ternary search trees

- Apply same idea to binary search trees.
- Simple algorithm, writes itself
- Faster than hashing.

The limiting distribution is *not* normal (and *is* a challenge to characterize)

What is the *limiting distribution* of the number of compares?



If it exists, it is not normal (Henniquin, 1987).

It exists (Regnier, 1989).

Is a unique fixed point of an explicit distributional identity (Rosler, 1991).

Has extremely small tails (several authors, 1991-2015)

Explicit tail bounds (Janson, 2015):

$$\exp\left(-e^{\gamma x+\ln\ln x+O(1)}\right) \le \Pr(Z \le -x) \le \exp\left(-e^{\gamma x+O(1)}\right) \qquad \gamma = \frac{1}{2-\frac{1}{\ln 2}}$$

$$\exp\left(-x\ln x - x\ln\ln x + O(x)\right) \le \Pr(Z \ge x) \le \exp\left(-x\ln x + O(x)\right)$$

What do we know about Quicksort?

C.A.R. Hoare invented it in 1959 and wrote a definitive paper introducing it in 1962.

Quicksort

By C. A. R. Hoare

A description is given of a new method of sorting in the random-access store of a computer. The method compares very favourably with other known methods in speed, in economy of storage, and in ease of programming. Certain refinements of the method, which may be useful in the optimization of inner loops, are described in the second part of the paper.

Part One: Theory

The sorting method described in this paper is based on the principle of resolving a problem into two simpler subproblems. Each of these subproblems may be resolved to produce yet simpler problems. The process is repeated until all the resulting problems are found to be trivial. These trivial problems may then be solved by known methods, thus obtaining a solution of the original more complex problem.

Partition

The problem of sorting a mass of items, occupying consecutive locations in the store of a computer, may be reduced to that of sorting two lesser segments of data, provided that it is known that the keys of each of the items held in locations lower than a certain dividing line are less than the keys of all the items held in locations above this dividing line. In this case the two segments may be sorted separately, and as a result the whole mass of data will be sorted.

In practice, the existence of such a dividing line will be rare, and even if it did exist its position would be unknown. It is, however, quite easy to rearrange the items in such a way that a dividing line is brought into existence, and its position is known. The method of doing this has been given the name *partition*. The description given below is adapted for a computer which has an *exchange* instruction; a method more suited for computers without such an instruction will be given in the second part of this paper.

The first step of the partition process is to choose a particular key value which is known to be within the range of the keys of the items in the segment which is to be sorted. A simple method of ensuring this is to choose the actual key value of one of the items in the segment. The chosen key value will be called the *bound*. The aim is now to produce a situation in which the keys of all items below a certain dividing line are equal to or less than the bound, while the keys of all items above the dividing line are equal to or greater than the bound. Fortunately, we do not need to know the position of the dividing line in advance; its position is determined only at the end of the partition process.

The items to be sorted are scanned by two pointers; one of them, the *lower pointer*, starts at the item with lowest address, and moves upward in the store, while the other, the *upper pointer*, starts at the item with the highest address and moves downward. The lower pointer starts first. If the item to which it refers has a key which is equal to or less than the bound, it moves up to point to the item in the next higher group of locations. It continues to move up until it finds an item with key value greater than the bound. In this case the lower pointer stops, and the upper pointer starts its scan. If the item to which it refers has a key which is equal to or greater than the bound, it moves down to point to the item in the next lower locations. It continues to move down until it finds an item with key value less than the bound. Now the two items to which the pointers refer are obviously in the wrong positions, and they must be exchanged. After the exchange, each pointer is stepped one item in its appropriate direction, and the lower pointer resumes its upward scan of the data. The process continues until the pointers cross each other, so that the lower pointer refers to an item in higher-addressed locations than the item referred to by the upper pointer. In this case the exchange of items is suppressed, the dividing line is drawn between the two pointers, and the partition process is at an end.

An awkward situation is liable to arise if the value of the bound is the greatest or the least of all the key values in the segment, or if all the key values are equal. The danger is that the dividing line, according to the rule given above, will have to be placed outside the segment which was supposed to be partitioned, and therefore the whole segment has to be partitioned again. An infinite cycle may result unless special measures are taken. This may be prevented by the use of a method which ensures that at least one item is placed in its correct position as a result of each application of the partitioning process. If the item from which the value of the bound has been taken turns out to be in the lower of the two resulting segments, it is known to have a key value which is equal to or greater than that of all the other items of this segment. It may therefore be exchanged with the item which occupies the highest-addressed locations in the segment, and the size of the lower resulting segment may be reduced by one. The same applies, *mutatis mutandis*, in the case where the item which gave the bound is in the upper segment. Thus the sum of the numbers of items in the two segments, resulting from the partitioning process, is always one less than the number of items in the original segment, so that it is

10

Some important facts learned since the 1960s

- Students should read Hoare's paper (still).
- Randomization is critical (and must be done carefully).
- Attention must be paid to equal keys.
- Quicksort is optimal.
- Substantial improvents are possible for a common key type.
- The limiting distribution is *not* normal
- Multiway partitioning is effective on modern machines.
- Quicksort remains the method of choice.

next talk

What do we know about Quicksort?

a brief summary of facts learned since the 1960s

Robert Sedgewick
Princeton University