3.5 Applications

Sorting Applications

Applications.

obvious applications

- Sort a list of names.
- Organize an MP3 library.
- Display Google PageRank results.
- List RSS news items in reverse chronological order.

problems become easy once items are in sorted order

- Find the median.
- Find the closest pair.
- Binary search in a database.
- Identify statistical outliers.
- Find duplicates in a mailing list.

non-obvious applications

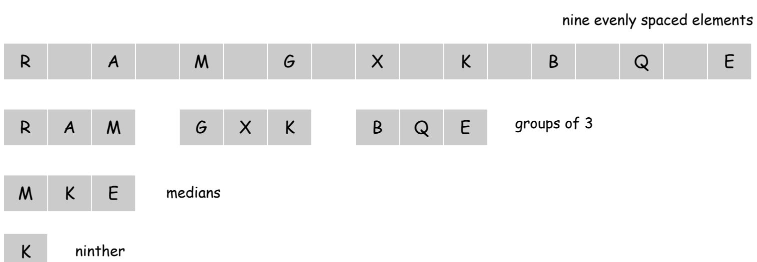
- Data compression.
- Computer graphics.
- Computational biology.
- Supply chain management.
- Book recommendations on Amazon.
- Load balancing on a parallel computer.

. . .

Engineering a System Sort

Bentley-McIlroy. [Engineeering a Sort Function]

- Original motivation: improve `qsort` function in C.
- Basic algorithm = 3-way quicksort with cutoff to insertion sort..
- Partition on Tukey's ninther: Approximate median-of-9.
  - used median-of-3 elements, each of which is median-of-3
  - idea borrowed from statistics, useful in many disciplines



Java System Sorts

Java's system sort.

- Can sort array of type `Comparable` or any primitive type.
- Uses Bentley-McIlroy quicksort for primitive types.
- Uses mergesort for objects.

```
import java.util.Arrays;
public class IntegerSort {
   public static void main(String[] args) {
      int N = Integer.parseInt(args[0]);
      int[] a = new int[N];
      for (int i = 0; i < N; i++)
         a[i] = StdIn.readInt();
      Arrays.sort(a);
      for (int i = 0; i < N; i++)
         System.out.println(a[i]);
   }
}
```

Q. Why difference between objects and primitive types?

Breaking Java's System Sort

Is it possible to make system sort go quadratic?

- No, for mergesort.
- Yes, for deterministic quicksort.

so, why are most system implementations of quicksort deterministic?

McIlroy's devious idea. [A Killer Adversary for Quicksort]

- Construct malicious input while running system quicksort in response to elements compared.
- If `p` is pivot, commit to `(x < p)` and `(y < p)`, but don't commit to `(x < y)` or `(x > y)` until `x` and `y` are compared.

Consequences.

- Confirms theoretical possibility.
- Algorithmic complexity attack: you enter linear amount of data; server performs quadratic amount of work.

Breaking Java's System Sort

A killer input. Blows function call stack in Java and crashes program.

more disastrous possibilities in C

```
% more 250000.txt
0
218750
222662
11
166672
247070
83339
156253
...
```

250,000 integers between 0 and 250,000

```
% java IntegerSort < 250000.txt
Exception in thread "main"
java.lang.StackOverflowError
   at java.util.Arrays.sort1(Arrays.java:562)
   at java.util.Arrays.sort1(Arrays.java:606)
   at java.util.Arrays.sort1(Arrays.java:608)
   at java.util.Arrays.sort1(Arrays.java:608)
   at java.util.Arrays.sort1(Arrays.java:608)
   . . .
```

Java's sorting library crashes, even if you give it as much stack space as Windows allows.

Natural Order

```
public class Date implements Comparable<Date> {
   private int month, day, year;

   public Date(int m, int d, int y) {
      month = m;
      day   = d;
      year  = y;
   }

   public int compareTo(Date b) {
      Date a = this;
      if (a.year  < b.year ) return -1;
      if (a.year  > b.year ) return +1;
      if (a.month < b.month) return -1;
      if (a.month > b.month) return +1;
      if (a.day   < b.day  ) return -1;
      if (a.day   > b.day  ) return +1;
      return 0;
   }
}
```

only compare dates to other dates

Sorting Different Types of Data

Goal. Sort objects with no natural order or with a different orders.

Ex. Sort strings by:

- Natural order. `Now is the time`
- Case insensitive. `is Now the time`
- French. `real réal rico`
- Spanish. `café cuidado champiñón dulce`

ch and rr are single letters

```
String[] a;
...
Arrays.sort(a);
Arrays.sort(a, String.CASE_INSENSITIVE_ORDER);
Arrays.sort(a, Collator.getInstance(Locale.FRENCH));
Arrays.sort(a, Collator.getInstance(Locale.SPANISH));
```

`import java.text.Collator;`

Comparator

Comparator interface. Class promises to implement a method `compare` so that `compare(v, w)` is a total order and behaves like `compareTo`.

Advantage. Separates the definition of the data type from what it means to compare two objects of that type.

- Add a new order to a data type.
- Add an order to a library data type with no natural order.

```
public class ReverseOrder implements Comparator<String> {
   public int compare(String a, String b) {
      return -a.compareTo(b);
   }
}
```

```
Arrays.sort(a, new ReverseOrder());
```


Insertion Sort: Comparator Version

Sorting library. Easy modification to support comparators.

```
public static void sort(Object[] a, Comparator comparator) {
   int N = a.length;
   for (int i = 0; i < N; i++)
      for (int j = i; j > 0; j--)
         if (less(comparator, a[j], a[j-1])) exch(a, j, j-1);
         else break;
}

private static boolean less(Comparator c, Object v, Object w) {
   return c.compare(v, w) < 0;
}
private static void exch(Object[] a, int i, int j) {
   Object t = a[i]; a[i] = a[j]; a[j] = t;
}
```

insertion sort

Sorting By Different Fields

Design challenge: enable sorting students by name or by section.

```
Arrays.sort(students, Student.BY_NAME);
Arrays.sort(students, Student.BY_SECT);
```

sort by name ↓

Andrews	3	A	664-480-0023	097 Little
Battle	4	C	874-088-1212	121 Whitman
Chen	2	A	991-878-4944	308 Blair
Fox	1	A	884-232-5341	11 Dickinson
Furia	3	A	766-093-9873	101 Brown
Gazsi	4	B	665-303-0266	22 Brown
Kanaga	3	B	898-122-9643	22 Brown
Rohde	3	A	232-343-5555	343 Forbes

then sort by section ↓

Fox	1	A	884-232-5341	11 Dickinson
Chen	2	A	991-878-4944	308 Blair
Andrews	3	A	664-480-0023	097 Little
Furia	3	A	766-093-9873	101 Brown
Kanaga	3	B	898-122-9643	22 Brown
Rohde	3	A	232-343-5555	343 Forbes
Battle	4	C	874-088-1212	121 Whitman
Gazsi	4	B	665-303-0266	22 Brown

Sorting By Different Fields

```
import java.util.Arrays;

public class Student {
   private String name;
   private int section;

   public static final Comparator<Student> BY_NAME = new ByName();
   public static final Comparator<Student> BY_SECT = new BySect();

   ...

   private static class ByName implements Comparator<Student> {
      public int compare(Student a, Student b) {
         return a.name.compareTo(b.name);
      }
   }

   private static class BySect implements Comparator<Student> {
      public int compare(Student a, Student b) {
         return a.section - b.section;
      }
   }
}
```

Stability

A stable sort preserves the relative order of records with equal keys.

sort by name

Andrews	3	A	664-480-0023	097 Little
Battle	4	C	874-088-1212	121 Whitman
Chen	2	A	991-878-4944	308 Blair
Fox	1	A	884-232-5341	11 Dickinson
Furia	3	A	766-093-9873	101 Brown
Gazsi	4	B	665-303-0266	22 Brown
Kanaga	3	B	898-122-9643	22 Brown
Rohde	3	A	232-343-5555	343 Forbes

then sort by section

Fox	1	A	884-232-5341	11 Dickinson
Chen	2	A	991-878-4944	308 Blair
Kanaga	3	B	898-122-9643	22 Brown
Andrews	3	A	664-480-0023	097 Little
Furia	3	A	766-093-9873	101 Brown
Rohde	3	A	232-343-5555	343 Forbes
Battle	4	C	874-088-1212	121 Whitman
Gazsi	4	B	665-303-0266	22 Brown

@#%&@!! Students in section 3 no longer in order by name.

Stability

Q. Which sorts are stable?
- Selection sort.
- Insertion sort.
- Quicksort.
- Mergesort.

Annoying fact. Many useful sorting algorithms are unstable.

Lots of Sorting Algorithms

Internal sorts.
- Insertion sort, selection sort, bubblesort, shaker sort.
- Quicksort, mergesort, heapsort, samplesort, introsort, shellsort.
- Solitaire sort, red-black sort, splaysort, Dobosiewicz sort, psort, ...

External sorts. Poly-phase mergesort, cascade-merge, oscillating sort.

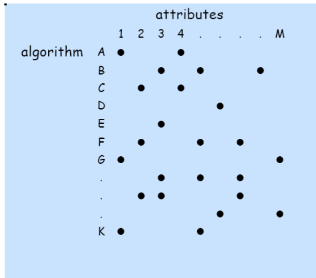
Radix sorts.
- Distribution, MSD, LSD.
- 3-way radix quicksort.

Parallel sorts.
- Bitonic sort, Batcher even-odd sort.
- Smooth sort, cube sort, column sort.
- GPUsort.

Lots of Sorting Attributes

Q. Isn't the system sort good enough.

A. Maybe.
- Stable?
- Multiple keys?
- Deterministic?
- Keys all distinct?
- Multiple key types?
- Linked list or arrays?
- Large or small records?
- Is your file randomly ordered?
- Need guaranteed performance?



many more combinations of attributes than algorithms

A. An elementary sorting algorithm may be the method of choice.
A. Use well understood topic to study basic issues.

3.6 Complexity

Computational Complexity

Computational complexity. Framework to study efficiency of algorithms for solving a particular problem X.

Machine model. Count fundamental operations.

Upper bound. Cost guarantee provided by some algorithm for X.
Lower bound. Proven limit on cost guarantee of any algorithm for X.
Optimal algorithm. Algorithm with best cost guarantee for X.

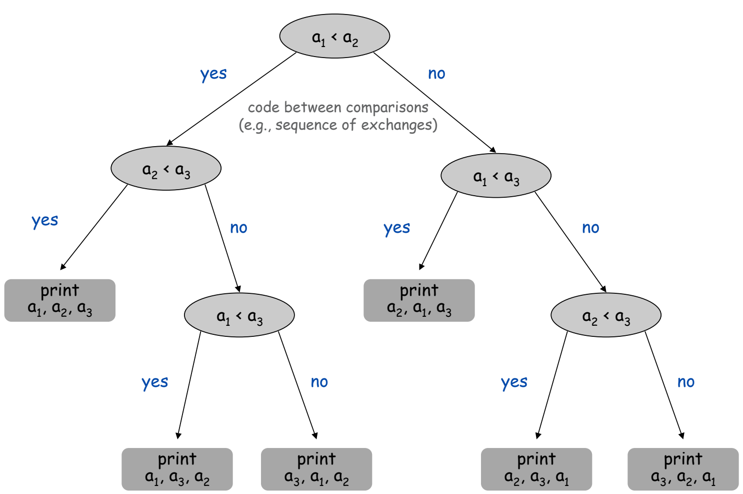
lower bound ≈ upper bound

Ex: sorting.

- Machine model = # comparisons in decision tree.
- Upper bound = $N \log_2 N$ from mergesort.
- Lower bound = $N \log_2 N - N \log_2 e$.
- Optimal algorithm = mergesort.

access information only through compares

Decision Tree



Comparison Based Sorting Lower Bound

Theorem. Any comparison based sorting algorithm must use $\Omega(N \log_2 N)$ comparisons.

Pf.

- Suffices to establish lower bound when input consists of N distinct values a_1 through a_N.
- Worst case dictated by tree height h.
- N! different orderings.
- (At least) one leaf corresponds to each ordering.
- Binary tree with N! leaves must have height

$$\begin{aligned} h &\geq \log_2(N!) \\ &\geq \log_2(N/e)^N \\ &= N \log_2 N - N \log_2 e \end{aligned}$$

← Stirling's formula

Comparison Based Sorting Lower Bound

Q. What if we have information about the keys to be sorted or their initial arrangement?

Partially ordered arrays. Depending on the initial order of the input, we may not need N log N compares. ← insertion sort requires O(N) compares on an already sorted array

Duplicate keys. Depending on the input distribution of duplicates, we may not need N log N compares. ← 3-way quicksort requires O(N) compares if there are only 17 distinct keys

Digital property of keys. We can use digit/character comparisons instead of key comparisons for numbers and strings. ← stay tuned for radix sort