3.5 Applications

Engineering a System Sort

Bentley-McIlroy. [Engineering 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’s system sort.

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

Q. Why difference between objects and primitive types?

Applications.

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

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

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

...
Breaking Java's System Sort

Is it possible to make system sort go quadratic?
- No, for mergesort.
- Yes, for deterministic quicksort.

```
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

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.

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.

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;
    }
}
```

Breaking Java's System Sort

Breaking Java's System Sort

Consequences.
- Sorts objects with no natural order or with a different orders.

Ex.
- Natural order: Now is the time
- Case insensitive: is Now is the time
- French: real réal rico
- Spanish: café cuidado champiñón dulce
  ch and rr are single letters

```
import java.text.Collator;
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));
```
**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.

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

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

**Sorting By Different Fields**

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

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

**Insertion Sort: Comparator Version**

**Sorting library.** Easy modification to support comparators.

```java
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;
}
```

```java
private static boolean less(Comparator c, Object v, Object w) {
    return c.compare(v, w) < 0;
}
```

```java
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**

```java
import java.util.Arrays;
public class Student {
    private String name;
    private int section;

    public static final Comparator<Student> BY_NAME = newByName();
    public static final Comparator<Student> BY_SECT = newBy Sect();

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

    private static class By Sect implements Comparator<Student> {
        public int compare(Student a, Student b) {
            return a.section - b.section;
        }
    }
}
```
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, Batched even-odd sort.
- Smooth sort, cube sort, column sort.
- GPUsort.

Stability

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

<table>
<thead>
<tr>
<th>sort by name</th>
<th>then sort by section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Andrews</td>
<td>Fox</td>
</tr>
<tr>
<td>Battle</td>
<td>Chen</td>
</tr>
<tr>
<td>Chen</td>
<td>Fox</td>
</tr>
<tr>
<td>Fox</td>
<td>Furia</td>
</tr>
<tr>
<td>Furia</td>
<td>Gazi</td>
</tr>
<tr>
<td>Gazi</td>
<td>Kanaga</td>
</tr>
<tr>
<td>Kanaga</td>
<td>Rohde</td>
</tr>
<tr>
<td>Rohde</td>
<td>Andrews</td>
</tr>
<tr>
<td></td>
<td>Battle</td>
</tr>
<tr>
<td></td>
<td>Fox</td>
</tr>
</tbody>
</table>

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.

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?

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.

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.

**Decision Tree**

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

$$h \geq \log_2(N!)$$
$$\geq \log_2(N/e)^N$$
$$= N \log_2 N - N \log_2 e$$

**Comparison Based Sorting Lower Bound**

Stirling's formula

Princeton EDU / ~cos226
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.

**Duplicate keys.** Depending on the input distribution of duplicates, we may not need $N \log N$ compares.

**Digital property of keys.** We can use digit/character comparisons instead of key comparisons for numbers and strings.