4.1, 4.2 Performance, with Sorting

Running Time

“As soon as an Analytic Engine exists, it will necessarily guide the future course of the science. Whenever any result is sought by its aid, the question will arise - By what course of calculation can these results be arrived at by the machine in the shortest time?” – Charles Babbage

Algorithmic Successes

N-body Simulation.
- Simulate gravitational interactions among N bodies.
- Brute force: \( N^2 \) steps.
Algorithmic Successes

N-body Simulation.
• Simulate gravitational interactions among N bodies.
• Brute force: $N^2$ steps.
• Barnes-Hut: $N \log N$ steps, enables new research.

Discrete Fourier transform.
• Break down waveform of N samples into periodic components.
• Applications: DVD, JPEG, MRI, astrophysics, ....
• Brute force: $N^2$ steps.
• FFT algorithm: $N \log N$ steps, enables new technology.

Sorting
Sorting

Sorting problem. Rearrange \( N \) items in ascending order.

Applications. Binary search, statistics, databases, data compression, bioinformatics, computer graphics, scientific computing, (too numerous to list) ...


Insertion Sort

Insertion sort.
- Brute-force sorting solution.
- Move left-to-right through array.
- Insert each element into correct position by exchanging it with larger elements to its left, one-by-one.

\[
\begin{array}{ccc|ccccccc}
  i & j & a & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
  \hline
  6 & 6 & \text{and} & \text{had} & \text{him} & \text{his} & \text{was} & \text{you} & \text{\textcolor{red}{the}} & \text{but} \\
  6 & 5 & \text{and} & \text{had} & \text{him} & \text{his} & \text{was} & \text{\textcolor{red}{the}} & \text{you} & \text{but} \\
  6 & 4 & \text{and} & \text{had} & \text{him} & \text{his} & \text{\textcolor{red}{the}} & \text{was} & \text{you} & \text{but} \\
\end{array}
\]

Inserting \( a[6] \) into position by exchanging with larger entries to its left

\[
\begin{array}{cccccccc}
  i & j & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
  \hline
  1 & 0 & \text{\textcolor{red}{had}} & \text{was} & \text{him} & \text{and} & \text{you} & \text{his} & \text{the} & \text{but} \\
  2 & 1 & \text{had} & \text{him} & \text{was} & \text{and} & \text{you} & \text{his} & \text{the} & \text{but} \\
  3 & 0 & \text{and} & \text{had} & \text{him} & \text{was} & \text{you} & \text{his} & \text{the} & \text{but} \\
  4 & 4 & \text{and} & \text{had} & \text{him} & \text{was} & \text{you} & \text{his} & \text{the} & \text{but} \\
  5 & 3 & \text{and} & \text{had} & \text{him} & \text{his} & \text{was} & \text{you} & \text{the} & \text{but} \\
  6 & 4 & \text{and} & \text{had} & \text{him} & \text{his} & \text{the} & \text{was} & \text{you} & \text{but} \\
  7 & 1 & \text{and} & \text{\textcolor{red}{but}} & \text{had} & \text{him} & \text{his} & \text{the} & \text{was} & \text{you} \\
\end{array}
\]

Inserting \( a[1] \) through \( a[N-1] \) into position (insertion sort)
**Insertion Sort: Java Implementation**

```java
public class Insertion {

    public static void sort(double[] a) {
        int N = a.length;
        for (int i = 1; i < N; i++)
            for (int j = i; j > 0; j--)
                if (a[j-1] > a[j])
                    exch(a, j-1, j);               // see text p. 70
                else break;
    }

    private static void exch(double[] a, int i, int j) {
        double swap = a[i];
        a[i] = a[j];
        a[j] = swap;
    }
}
```

**Insertion Sort: Observation**

Observe and tabulate running time for various values of N.
- Data source: N random numbers between 0 and 1.
- Machine: Apple Model XXX with lots of memory, running OS X.
- Timing: Skagen wristwatch.

<table>
<thead>
<tr>
<th>N</th>
<th>Comparisons</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>5,000</td>
<td>6.2 million</td>
<td>0.13 s</td>
</tr>
<tr>
<td>10,000</td>
<td>25 million</td>
<td>0.43 s</td>
</tr>
<tr>
<td>20,000</td>
<td>99 million</td>
<td>1.5 s</td>
</tr>
<tr>
<td>40,000</td>
<td>400 million</td>
<td>5.6 s</td>
</tr>
<tr>
<td>80,000</td>
<td>1600 million</td>
<td>23 s</td>
</tr>
</tbody>
</table>

**Insertion Sort: Empirical Analysis**

Data analysis. Plot # comparisons vs. input size on log-log scale.

Hypothesis. # comparisons grows quadratically with input size \( \sim N^2/4 \).
Analysis: Empirical vs. Mathematical

Empirical analysis.
- Measure running times, plot, and fit curve.
- Easy to perform experiments.
- Model useful for predicting, but not for explaining.

Mathematical analysis.
- Analyze algorithm to estimate # ops as a function of input size.
- May require advanced mathematics.
- Model useful for predicting and explaining.

Critical difference. Mathematical analysis is independent of a particular machine or compiler; applies to machines not yet built.

Insertion Sort: Mathematical Analysis

Worst case. [descending]
- Iteration $i$ requires $i$ comparisons.
- Total $= (0 + 1 + 2 + \ldots + N-1) \sim N^2 / 2$ compares.

Average case. [random]
- Iteration $i$ requires $i / 2$ comparisons on average.
- Total $= (0 + 1 + 2 + \ldots + N-1) / 2 \sim N^2 / 4$ compares

Insertion Sort: Lesson

Lesson. Supercomputer can’t rescue a bad algorithm.

<table>
<thead>
<tr>
<th>Computer</th>
<th>Comparisons Per Second</th>
<th>Thousand</th>
<th>Million</th>
<th>Billion</th>
</tr>
</thead>
<tbody>
<tr>
<td>laptop</td>
<td>$10^7$</td>
<td>instant</td>
<td>1 day</td>
<td>3 centuries</td>
</tr>
<tr>
<td>super</td>
<td>$10^{12}$</td>
<td>instant</td>
<td>1 second</td>
<td>2 weeks</td>
</tr>
</tbody>
</table>

Moore’s Law

Moore’s law. Transistor density on a chip doubles every 2 years.

Variants. Memory, disk space, bandwidth, computing power per $.
Moore’s Law and Algorithms

Quadratic algorithms do not scale with technology.
• New computer may be 10x as fast.
• But, has 10x as much memory so problem may be 10x bigger.
• With quadratic algorithm, takes 10x as long!

“Software inefficiency can always outpace Moore’s Law. Moore’s Law isn’t a match for our bad coding.” – Jaron Lanier

Lesson. Need linear (or linearithmic) algorithm to keep pace with Moore’s law.

Mergesort

Mergesort.
• Divide array into two halves.
• Recursively sort each half.
• Merge two halves to make sorted whole.

input
  was had him and you his the but
sort left
  and had him was you his the but
sort right
  and had him was but his the you
merge
  and but had him his the was you

Mergesort: Example

Top-down mergesort
Merging. Combine two pre-sorted lists into a sorted whole.

How to merge efficiently? Use an auxiliary array.

```
String[] aux = new String[N];
// Merge into auxiliary array.
int i = lo, j = mid;
for (int k = 0; k < N; k++)
{
    if (i == mid) aux[k] = a[j++];
    else if (j == hi) aux[k] = a[i++];
    else if (a[j].compareTo(a[i]) < 0) // String compare: text p. 523
        aux[k] = a[j++];
    else aux[k] = a[i++];
}
// Copy back.
for (int k = 0; k < N; k++)
a[lo + k] = aux[k];
```

Trace of the merge of the sorted left half with the sorted right half

---

Mergesort: Java Implementation

```java
public class Merge {
    public static void sort(String[] a)
    {
        sort(a, 0, a.length);
    }

    // Sort a[lo, hi).
    public static void sort(String[] a, int lo, int hi)
    {
        int N = hi - lo;
        if (N <= 1) return;

        // Recursively sort left and right halves.
        int mid = lo + N/2;
        sort(a, lo, mid);
        sort(a, mid, hi);

        // Merge sorted halves (see previous slide).
    }
}
```

Mergesort: Empirical Analysis

Experimental hypothesis. Number of comparisons \( \approx 20N \).
Mergesort: Prediction and Verification

**Experimental hypothesis.** Number of comparisons \( \approx 20N \).

**Prediction.** 80 million comparisons for \( N = 4 \) million.

**Observations.**

<table>
<thead>
<tr>
<th>( N )</th>
<th>Comparisons</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 million</td>
<td>82.7 million</td>
<td>3.13 sec</td>
</tr>
<tr>
<td>4 million</td>
<td>82.7 million</td>
<td>3.25 sec</td>
</tr>
<tr>
<td>4 million</td>
<td>82.7 million</td>
<td>3.22 sec</td>
</tr>
</tbody>
</table>

**Prediction.** 400 million comparisons for \( N = 20 \) million.

**Observations.**

<table>
<thead>
<tr>
<th>( N )</th>
<th>Comparisons</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 million</td>
<td>460 million</td>
<td>17.5 sec</td>
</tr>
<tr>
<td>50 million</td>
<td>1216 million</td>
<td>45.9 sec</td>
</tr>
</tbody>
</table>

Mergesort: Mathematical Analysis

**Mathematical analysis.**

<table>
<thead>
<tr>
<th></th>
<th>comparisons</th>
</tr>
</thead>
<tbody>
<tr>
<td>worst</td>
<td>( N \log_2 N )</td>
</tr>
<tr>
<td>average</td>
<td>( N \log_2 N )</td>
</tr>
<tr>
<td>best</td>
<td>( \frac{1}{2} N \log_2 N )</td>
</tr>
</tbody>
</table>

**Validation.** Theory agrees with observations.

<table>
<thead>
<tr>
<th>( N )</th>
<th>actual</th>
<th>predicted</th>
</tr>
</thead>
<tbody>
<tr>
<td>10,000</td>
<td>120 thousand</td>
<td>133 thousand</td>
</tr>
<tr>
<td>20 million</td>
<td>460 million</td>
<td>485 million</td>
</tr>
<tr>
<td>50 million</td>
<td>1,216 million</td>
<td>1,279 million</td>
</tr>
</tbody>
</table>

Mergesort: Lesson

**Lesson.** Great algorithms can be more powerful than supercomputers.

<table>
<thead>
<tr>
<th>Computer</th>
<th>Comparisons Per Second</th>
<th>Insertion</th>
<th>Mergesort</th>
</tr>
</thead>
<tbody>
<tr>
<td>laptop</td>
<td>( 10^7 )</td>
<td>3 centuries</td>
<td>3 hours</td>
</tr>
<tr>
<td>super</td>
<td>( 10^{12} )</td>
<td>2 weeks</td>
<td>instant</td>
</tr>
</tbody>
</table>

\( N = 1 \) billion
Binary Search

Idea:
• Sort the array
• Play "20 questions" to determine the index associated with a given key.

Ex. Dictionary, phone book, book index, credit card numbers, ...

Binary search.
• Examine the middle key.
• If it matches, return its index.
• Otherwise, search either the left or right half.

Java library implementation: Arrays.binarySearch()
**Binary Search: Mathematical Analysis**

**Analysis.** To binary search in an array of size $N$: do one comparison, then binary search in an array of size $N/2$.

$N \rightarrow N/2 \rightarrow N/4 \rightarrow N/8 \rightarrow \ldots \rightarrow 1$

**Q.** How many times can you divide a number by 2 until you reach 1?

**A.** $\log_2 N$.

**Order of Growth Classification**

**Observation.** A small subset of mathematical functions suffice to describe running time of many fundamental algorithms.

```plaintext
while (N > 1) {
  N = N / 2;
  ... 
}
```

$$\lg N$$

```plaintext
for (int i = 0; i < N; i++)
  ...
$$N \lg N$$

```plaintext
N
```

```plaintext
for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++)
    ...
$$N^2$$

```plaintext
public static void g(int N) {
  if (N == 0) return;
  g(N/2);
  g(N/2);
  for (int i = 0; i < N; i++)
    ...
}
```

```plaintext
public static void f(int N) {
  if (N == 0) return;
  f(N-1);
  f(N-1);
  ...
}
```

$2^N$

**Summary**

**Q.** How can I evaluate the performance of my program?

**A.** Computational experiments, mathematical analysis

**Q.** What if it’s not fast enough? Not enough memory?

- Understand why.
- Buy a faster computer.
- Learn a better algorithm (COS 226, COS 423).
- Discover a new algorithm.

<table>
<thead>
<tr>
<th>attribute</th>
<th>better machine</th>
<th>better algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>cost</td>
<td>$$$ or more</td>
<td>$ or less</td>
</tr>
<tr>
<td>applicability</td>
<td>makes &quot;everything&quot; run faster</td>
<td>does not apply to some problems</td>
</tr>
<tr>
<td>improvement</td>
<td>incremental quantitative improvements expected</td>
<td>dramatic qualitative improvements possible</td>
</tr>
</tbody>
</table>
Q. What's the fastest way to sort 1 million 32-bit integers?