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.

![Graph showing the increase in time with respect to the number of bodies.](image)
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.

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

Inserting a[1] through a[N-1] into position (insertion sort)
**Insertion Sort Demo**

*Iteration i.* Repeatedly swap element i with the one to its left if smaller.

*Property.* After ith iteration, a[0] through a[i] contain first i+1 elements in ascending order.

<table>
<thead>
<tr>
<th>Array index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>2.78</td>
<td>1.42</td>
<td>0.56</td>
<td>0.12</td>
<td>0.17</td>
<td>0.32</td>
<td>0.21</td>
<td>0.42</td>
<td>0.14</td>
<td>0.11</td>
</tr>
</tbody>
</table>

**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);
                } else break; // see text p. 70
        }
    }

    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 seconds</td>
</tr>
<tr>
<td>10,000</td>
<td>25 million</td>
<td>0.43 seconds</td>
</tr>
<tr>
<td>20,000</td>
<td>99 million</td>
<td>1.5 seconds</td>
</tr>
<tr>
<td>40,000</td>
<td>400 million</td>
<td>5.6 seconds</td>
</tr>
<tr>
<td>80,000</td>
<td>1600 million</td>
<td>23 seconds</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 ~ $N^{1/4}$.
Insertion Sort: Empirical Analysis

Observation. Number of compares depends on input family.
- Descending: $\sim N^2/2$.
- Random: $\sim N^2/4$.
- Ascending: $\sim N$.

![Comparison Chart]

Input Size

Comparisons (millions)

0.001
0.010
0.100
1.000
10.000
100.000
1000.000
10000.000
100000.000

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.

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 number of 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: 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.
Mergesort: Example

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>MER</td>
<td>GE</td>
<td>SO</td>
<td>RT</td>
<td>EX</td>
<td>AM</td>
<td>PLE</td>
<td>E</td>
<td>MER</td>
<td>GE</td>
<td>SO</td>
<td>RT</td>
<td>EX</td>
<td>AM</td>
<td>PLE</td>
<td>E</td>
</tr>
<tr>
<td>53</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Top-down mergesort

Merging

Merging. Combine two pre-sorted lists into a sorted whole.

How to merge efficiently? Use an auxiliary array.

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>k</th>
<th>aux[k]</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>0</td>
<td>and</td>
<td>had</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>1</td>
<td>but</td>
<td>had</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>2</td>
<td>had</td>
<td>had</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>3</td>
<td>him</td>
<td>had</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>4</td>
<td>his</td>
<td>had</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>5</td>
<td>the</td>
<td>had</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>6</td>
<td>was</td>
<td>had</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>7</td>
<td>you</td>
<td>had</td>
</tr>
</tbody>
</table>

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

Merging Demo

Merge.
- Keep track of smallest element in each sorted half.
- Choose smaller of two elements.
- Repeat until done.

\[
\begin{array}{c}
\downarrow \\
A G L O R \\
\hline
H I M S T
\end{array}
\]

\[
\begin{array}{c}
A G L O R \\
\hline
H I M S T
\end{array}
\]

\[
\begin{array}{c}
A G L O R \\
\hline
H I M S T
\end{array}
\]

\[
\begin{array}{c}
A G H I L M O R S T
\end{array}
\]
Merging

Merging. Combine two pre-sorted lists into a sorted whole.

How to merge efficiently? Use an auxiliary array.

```java
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];
```

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 \( \approx 20N \).

![Graph showing comparison counts and input sizes](image)

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 mill</td>
<td>82.7 million</td>
<td>3.13 sec</td>
</tr>
</tbody>
</table>

Agrees.

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 mill</td>
<td>460 million</td>
<td>17.5 sec</td>
</tr>
<tr>
<td>50 mill</td>
<td>1216 million</td>
<td>45.9 sec</td>
</tr>
</tbody>
</table>

Not quite.
**Analysis.** To mergesort array of size $N$, mergesort two subarrays of size $N/2$, and merge them together using at most $N$ comparisons. We assume $N$ is a power of 2.

Mathematical analysis.

<table>
<thead>
<tr>
<th>Analysis</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>$1/2 N \log_2 N$</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>

**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>$10^9$</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
**Twenty Questions**

**Intuition.** Find a hidden integer.

<table>
<thead>
<tr>
<th>interval</th>
<th>size</th>
<th>Q</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>128</td>
<td>&lt; 64 ?</td>
<td>no</td>
</tr>
<tr>
<td>64</td>
<td>128</td>
<td>&lt; 96 ?</td>
<td>yes</td>
</tr>
<tr>
<td>64</td>
<td>96</td>
<td>&lt; 80 ?</td>
<td>yes</td>
</tr>
<tr>
<td>64</td>
<td>80</td>
<td>&lt; 72 ?</td>
<td>no</td>
</tr>
<tr>
<td>72</td>
<td>80</td>
<td>&lt; 76 ?</td>
<td>no</td>
</tr>
<tr>
<td>76</td>
<td>80</td>
<td>&lt; 78 ?</td>
<td>yes</td>
</tr>
<tr>
<td>76</td>
<td>78</td>
<td>&lt; 77 ?</td>
<td>no</td>
</tr>
<tr>
<td>77</td>
<td></td>
<td></td>
<td>1 = 77</td>
</tr>
</tbody>
</table>

**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.

**Invariant.** Algorithm maintains $a[lo] \leq \text{key} \leq a[hi-1]$.

**Ex.** Binary search for 33.

<table>
<thead>
<tr>
<th>6</th>
<th>10</th>
<th>13</th>
<th>14</th>
<th>25</th>
<th>33</th>
<th>36</th>
<th>43</th>
<th>51</th>
<th>53</th>
<th>64</th>
<th>72</th>
<th>84</th>
<th>93</th>
<th>95</th>
<th>96</th>
<th>97</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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 $N/2$.

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

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

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

<table>
<thead>
<tr>
<th>$N$</th>
<th>$\to$</th>
<th>$N/2$</th>
<th>$\to$</th>
<th>$N/4$</th>
<th>$\to$</th>
<th>$N/8$</th>
<th>$\to$</th>
<th>$\ldots$</th>
<th>\to 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>2</td>
<td></td>
<td>1</td>
<td></td>
<td>4</td>
<td></td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>4</td>
<td></td>
<td>2</td>
<td>1</td>
<td>8</td>
<td></td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>8</td>
<td></td>
<td>4</td>
<td>2</td>
<td>16</td>
<td></td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>16</td>
<td></td>
<td>8</td>
<td>4</td>
<td>32</td>
<td></td>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>32</td>
<td></td>
<td>16</td>
<td>8</td>
<td>64</td>
<td></td>
<td>32</td>
<td>16</td>
</tr>
<tr>
<td>32</td>
<td></td>
<td>64</td>
<td></td>
<td>32</td>
<td>16</td>
<td>128</td>
<td></td>
<td>64</td>
<td>32</td>
</tr>
<tr>
<td>64</td>
<td></td>
<td>128</td>
<td></td>
<td>64</td>
<td>32</td>
<td>256</td>
<td></td>
<td>128</td>
<td>64</td>
</tr>
<tr>
<td>128</td>
<td></td>
<td>256</td>
<td></td>
<td>128</td>
<td>64</td>
<td>512</td>
<td></td>
<td>256</td>
<td>128</td>
</tr>
<tr>
<td>256</td>
<td></td>
<td>512</td>
<td></td>
<td>256</td>
<td>128</td>
<td>1024</td>
<td></td>
<td>512</td>
<td>256</td>
</tr>
<tr>
<td>512</td>
<td></td>
<td>1024</td>
<td></td>
<td>512</td>
<td>256</td>
<td>2048</td>
<td></td>
<td>1024</td>
<td>512</td>
</tr>
<tr>
<td>1024</td>
<td></td>
<td>2048</td>
<td></td>
<td>1024</td>
<td>512</td>
<td>4096</td>
<td></td>
<td>2048</td>
<td>1024</td>
</tr>
<tr>
<td>2048</td>
<td></td>
<td>4096</td>
<td></td>
<td>2048</td>
<td>1024</td>
<td>8192</td>
<td></td>
<td>4096</td>
<td>2048</td>
</tr>
<tr>
<td>4096</td>
<td></td>
<td>8192</td>
<td></td>
<td>4096</td>
<td>2048</td>
<td>16384</td>
<td></td>
<td>8192</td>
<td>4096</td>
</tr>
<tr>
<td>8192</td>
<td></td>
<td>16384</td>
<td></td>
<td>8192</td>
<td>4096</td>
<td>32768</td>
<td></td>
<td>16384</td>
<td>8192</td>
</tr>
<tr>
<td>16384</td>
<td></td>
<td>32768</td>
<td></td>
<td>16384</td>
<td>8192</td>
<td>65536</td>
<td></td>
<td>32768</td>
<td>16384</td>
</tr>
<tr>
<td>32768</td>
<td></td>
<td>65536</td>
<td></td>
<td>32768</td>
<td>16384</td>
<td>131072</td>
<td></td>
<td>65536</td>
<td>32768</td>
</tr>
<tr>
<td>65536</td>
<td></td>
<td>131072</td>
<td></td>
<td>65536</td>
<td>32768</td>
<td>262144</td>
<td></td>
<td>131072</td>
<td>65536</td>
</tr>
<tr>
<td>131072</td>
<td></td>
<td>262144</td>
<td></td>
<td>131072</td>
<td>65536</td>
<td>524288</td>
<td></td>
<td>262144</td>
<td>131072</td>
</tr>
<tr>
<td>262144</td>
<td></td>
<td>524288</td>
<td></td>
<td>262144</td>
<td>131072</td>
<td>1048576</td>
<td></td>
<td>524288</td>
<td>262144</td>
</tr>
<tr>
<td>524288</td>
<td></td>
<td>1048576</td>
<td></td>
<td>524288</td>
<td>262144</td>
<td>2097152</td>
<td></td>
<td>1048576</td>
<td>524288</td>
</tr>
<tr>
<td>1048576</td>
<td></td>
<td>2097152</td>
<td></td>
<td>1048576</td>
<td>524288</td>
<td>4194304</td>
<td></td>
<td>2097152</td>
<td>1048576</td>
</tr>
<tr>
<td>2097152</td>
<td></td>
<td>4194304</td>
<td></td>
<td>2097152</td>
<td>1048576</td>
<td>8388608</td>
<td></td>
<td>4194304</td>
<td>2097152</td>
</tr>
</tbody>
</table>

### Order of Growth Classifications

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

```java
while (N > 1) {
    // ...
    lg N = log2 N
}

for (int i = 0; i < N; i++)
    // ...

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

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

N lg 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 “everything” 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?