4.1, 4.2 Performance and Sorting

“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.

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

Running Time

Charles Babbage (1864)

Analytic Engine
Algorithmic Successes

Discrete Fourier transform.
• Break down waveform of $N$ samples into periodic components.
• Applications: DVD, JPEG, MRI, astrophysics, ....
• Brute force: $N^2$ steps.

Brute force: $N^2$ steps.
FFT algorithm: $N \log N$ steps, enables new technology.

Algorithmic Successes

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) ...

Hanley
Haskell
Horn
Hayes
Hauser
Hornet
Hsu
Haskell
Hanley
Horn
Hayes
Hauser
Hornet
Hsu
Insertion Sort

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

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>6</td>
<td>and had him his was you the but</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>and had him his was the you but</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>and had him his the was you but</td>
</tr>
</tbody>
</table>

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

Insertion Sort: Java Implementation

```java
public class Insertion {
    public static void sort(String[] 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;
    }

    private static void exch(String[] a, int i, int j) {
        String 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 G5 1.8GHz with 1.5GB 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 $\sim N^2/4$.

**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 = \((0 + 1 + 2 + ... + N-1) \approx N^2 / 2\) compares.

![Descending order](image)

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

![Random order](image)

---

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](http://en.wikipedia.org/wiki/Moore%27s_law)

---

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.
Exam 1 looms.

Written exam Tuesday 3/13 during your lecture time. Room TBD.

Programming exam Tuesday 3/13 or Wednesday 3/14 in your precept.

Review session will be held.

Rooms, rules, details on Exams page of website.

---

Mergesort

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

Mergesort: Example

<table>
<thead>
<tr>
<th>input</th>
<th>was had him and you his the but</th>
</tr>
</thead>
<tbody>
<tr>
<td>sort left</td>
<td>and had him was you his the but</td>
</tr>
<tr>
<td>sort right</td>
<td>and had him was but his the you</td>
</tr>
<tr>
<td>merge</td>
<td>and but had him his the was you</td>
</tr>
</tbody>
</table>

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

How to merge efficiently? Use an auxiliary array.

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

Merging

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

Merge.

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

Merge.

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

Merge.

A G L O R H I M S T
Merging

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

Merging

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

Mergesort: Java Implementation

```java
def merge(String[] a, int lo, int hi)
    int N = hi - lo;
    int mid = lo + N/2;
    sort(a, lo, mid);
    sort(a, mid, hi);
    // Merge sorted halves (see previous slide).
    int[] aux = new int[N];
    for (int k = 0; k < N; k++)
        aux[k] = a[lo + k];
    for (int k = 0; k < N; k++)
        a[lo + k] = aux[k];
```

How to merge efficiently? Use an auxiliary array.
Mergesort: Empirical Analysis

**Experimental hypothesis.** Number of comparisons $\approx 20N$.

![Graph showing comparisons vs. input size]

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>

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 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>

Not quite.

Mergesort: Mathematical Analysis

**Analysis.** To mergesort array of size $N$, mergesort two subarrays of size $N/2$, and merge them together using $\leq N$ 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>$\frac{1}{2} N \log_2 N$</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>
**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**

**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 - 128</td>
<td>128</td>
<td>&lt; 64?</td>
<td>no</td>
</tr>
<tr>
<td>64 - 128</td>
<td>64</td>
<td>&lt; 96?</td>
<td>yes</td>
</tr>
<tr>
<td>32 - 64</td>
<td>32</td>
<td>&lt; 80?</td>
<td>yes</td>
</tr>
<tr>
<td>16 - 32</td>
<td>16</td>
<td>&lt; 72?</td>
<td>no</td>
</tr>
<tr>
<td>8 - 16</td>
<td>8</td>
<td>&lt; 76?</td>
<td>no</td>
</tr>
<tr>
<td>4 - 8</td>
<td>4</td>
<td>&lt; 78?</td>
<td>yes</td>
</tr>
<tr>
<td>2 - 4</td>
<td>2</td>
<td>&lt; 77?</td>
<td>no</td>
</tr>
<tr>
<td>1 - 2</td>
<td>1</td>
<td>= 77</td>
<td></td>
</tr>
</tbody>
</table>

**Idea:**
- Sort the array (stay tuned)
- 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.
Binary Search

Binary search. Given a key and sorted array \( a[] \), find index \( i \) such that \( a[i] = \text{key} \), or report that no such index exists.

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

Ex. Binary search for 33.
Binary Search

Given a key and sorted array $a[]$, find index $i$ such that $a[i] = key$, or report that no such index exists.

Invariant. Algorithm maintains $a[lo] \leq key \leq a[hi-1]$.

Ex. Binary search for 33.
Binary Search

**Binary search.** Given a key and sorted array $a[]$, find index $i$ such that $a[i] = key$, or report that no such index exists.

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

**Ex.** Binary search for 33.

![Binary search example](image)

**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.** $\log_2 N$.

![Binary search: Mathematical Analysis](image)

**Java library implementation:** Arrays.binarySearch()

**Binary Search: Java Implementation**

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

```java
public static int search(String key, String[] a) {
    return search(key, a, 0, a.length);
}

public static int search(String key, String[] a, int lo, int hi) {
    if (hi <= lo) return -1;
    int mid = lo + (hi - lo) / 2;
    int cmp = a[mid].compareTo(key);
    if (cmp > 0) return search(key, a, lo, mid);
    else if (cmp < 0) return search(key, a, mid+1, hi);
    else return mid;
}
```

**Order of Growth Classifications**

![Orders of growth (log-log plot)](image)

<table>
<thead>
<tr>
<th>order of growth</th>
<th>function</th>
<th>factor for doubling hypothesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>constant</td>
<td>$1$</td>
<td>$1$</td>
</tr>
<tr>
<td>logarithmic</td>
<td>$\log N$</td>
<td>$1$</td>
</tr>
<tr>
<td>linear</td>
<td>$N$</td>
<td>$2$</td>
</tr>
<tr>
<td>linearithmic</td>
<td>$N \log N$</td>
<td>$2$</td>
</tr>
<tr>
<td>quadratic</td>
<td>$N^2$</td>
<td>$4$</td>
</tr>
<tr>
<td>cubic</td>
<td>$N^3$</td>
<td>$8$</td>
</tr>
<tr>
<td>exponential</td>
<td>$2^N$</td>
<td>$2^N$</td>
</tr>
</tbody>
</table>

Commonly encountered growth functions
**Order of Growth Classifications**

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

```java
public static void g(int N) {
    if (N == 0) return;
    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 \log N

\( \lg N = \log_2 N \)

\( \text{while} \ (N > 1) \{
    N = N / 2;
    ...
\} \)

\( N \log N \)

\( \text{for (int i = 0; i < N; i++)}
    ...
\)

N

\( N^2 \)

\( \text{for (int i = 0; i < N; i++)}
    \text{for (int j = 0; j < N; j++)}
    ...
\)

\( 2^N \)

**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>