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.
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.

Andrew Appel
PU '81

![Graph showing time vs. number of bodies with quadratic, linearithmic, and linear trends.](Galaxies_NGC_1297_and_IC_2163.png)
Algorithmic Successes

Discrete Fourier transform.

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

![Graph showing time vs. number of samples]

Freidrich Gauss
1805
Algorithmic Successes

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.

![Graph](image)

\( time \)

\( size \) →

1K 2K 4K 8K

(number of samples)

- quadratic \( (N^2) \)
- linearithmic \( (N \log N) \)
- linear
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 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>0</td>
<td>1</td>
<td>2</td>
</tr>
<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>
<tr>
<td></td>
<td></td>
<td>and had him his the the was you but</td>
</tr>
</tbody>
</table>

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

**Insertion sort.**

- Brute-force sorting solution.
- Move left-to-right through array.
- Exchange next element 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></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td></td>
<td>was</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>had</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>had</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>and</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>and</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>and</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>and</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>and</td>
</tr>
</tbody>
</table>

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

**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><strong>Value</strong></td>
<td>2.78</td>
<td>7.42</td>
<td>0.56</td>
<td>1.12</td>
<td>1.17</td>
<td>0.32</td>
<td>6.21</td>
<td>4.42</td>
<td>3.14</td>
<td>7.71</td>
</tr>
</tbody>
</table>

**Iteration 0:** step 0.
Insertion Sort

**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>0.32</td>
<td>0.56</td>
<td>1.12</td>
<td>1.17</td>
<td>2.78</td>
<td>3.14</td>
<td>4.42</td>
<td>6.21</td>
<td>7.42</td>
<td>7.71</td>
</tr>
</tbody>
</table>

Iteration 10: **DONE.**
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;
    }
}
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>
Data analysis. Plot # comparisons vs. input size on log-log scale.

Hypothesis. # comparisons grows quadratically with input size $\sim N^2/4$. 
Observation. Number of compares depends on input family.

- **Descending**: $\sim \frac{N^2}{2}$.
- **Random**: $\sim \frac{N^2}{4}$.
- **Ascending**: $\sim N$. 

![Graph showing comparisons (millions) vs input size](image-url)
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 = $(0 + 1 + 2 + ... + N-1) \sim N^2 / 2$ compares.

- $E$ - $F$ - $G$ - $H$ - $I$ - $J$ - $D$ - $C$ - $B$ - $A$

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

- $A$ - $C$ - $D$ - $F$ - $H$ - $J$ - $E$ - $B$ - $I$ - $G$
**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 $.

http://en.wikipedia.org/wiki/Moore's_law
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

First Draft of a Report on the EDVAC

John von Neumann
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

<table>
<thead>
<tr>
<th></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>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>E</td>
<td>R</td>
<td>G</td>
<td>E</td>
<td>S</td>
<td>O</td>
<td>R</td>
<td>T</td>
<td>E</td>
<td>X</td>
<td>A</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>M</td>
<td>R</td>
<td>G</td>
<td>E</td>
<td>S</td>
<td>O</td>
<td>R</td>
<td>T</td>
<td>E</td>
<td>X</td>
<td>A</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>M</td>
<td>G</td>
<td>R</td>
<td>E</td>
<td>S</td>
<td>O</td>
<td>R</td>
<td>T</td>
<td>E</td>
<td>X</td>
<td>A</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>G</td>
<td>M</td>
<td>R</td>
<td>E</td>
<td>S</td>
<td>O</td>
<td>R</td>
<td>T</td>
<td>E</td>
<td>X</td>
<td>A</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>G</td>
<td>M</td>
<td>R</td>
<td>E</td>
<td>O</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>E</td>
<td>X</td>
<td>A</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>G</td>
<td>M</td>
<td>R</td>
<td>E</td>
<td>O</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>E</td>
<td>X</td>
<td>A</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>G</td>
<td>M</td>
<td>R</td>
<td>E</td>
<td>O</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>E</td>
<td>X</td>
<td>A</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>E</td>
<td>G</td>
<td>M</td>
<td>O</td>
<td>R</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>E</td>
<td>X</td>
<td>A</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>E</td>
<td>G</td>
<td>M</td>
<td>O</td>
<td>R</td>
<td>R</td>
<td>S</td>
<td>E</td>
<td>T</td>
<td>E</td>
<td>X</td>
<td>A</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>E</td>
</tr>
<tr>
<td>E</td>
<td>E</td>
<td>G</td>
<td>M</td>
<td>O</td>
<td>R</td>
<td>R</td>
<td>S</td>
<td>E</td>
<td>T</td>
<td>E</td>
<td>A</td>
<td>X</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>E</td>
</tr>
<tr>
<td>E</td>
<td>E</td>
<td>G</td>
<td>M</td>
<td>O</td>
<td>R</td>
<td>R</td>
<td>S</td>
<td>A</td>
<td>E</td>
<td>T</td>
<td>X</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>E</td>
<td>G</td>
<td>M</td>
<td>O</td>
<td>R</td>
<td>R</td>
<td>S</td>
<td>A</td>
<td>E</td>
<td>T</td>
<td>X</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>E</td>
<td>G</td>
<td>M</td>
<td>O</td>
<td>R</td>
<td>R</td>
<td>S</td>
<td>A</td>
<td>E</td>
<td>T</td>
<td>X</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>E</td>
<td>G</td>
<td>M</td>
<td>O</td>
<td>R</td>
<td>R</td>
<td>S</td>
<td>A</td>
<td>E</td>
<td>T</td>
<td>X</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>G</td>
<td>L</td>
<td>M</td>
<td>M</td>
<td>O</td>
<td>P</td>
<td>R</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>X</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></th>
<th></th>
<th></th>
<th>aux[k]</th>
<th></th>
<th></th>
<th></th>
<th>a</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>j</td>
<td>k</td>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></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>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>1</td>
<td>5</td>
<td>2</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>2</td>
<td>5</td>
<td>3</td>
<td>him</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>3</td>
<td>6</td>
<td>5</td>
<td>the</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>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>

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

Merge.

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

Merge.
• Keep track of smallest element in each sorted half.
• Choose smaller of two elements.
• Repeat until done.
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) aux[k] = a[j++];
    else aux[k] = a[i++];
}
// Copy back.
for (int k = 0; k < N; k++)
a[lo + k] = aux[k];
```
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: Java Implementation
Experimental hypothesis. Number of comparisons $\approx 20N$. 

![Comparison Graph]
Mergesort: Prediction and Verification

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

**Prediction.** 80 million comparisons for $N = 4$ million.

<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>

**Observations.**

Agrees.

**Prediction.** 400 million comparisons for $N = 20$ million.

<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>

**Observations.**

Not quite.
### Analysis

To mergesort an 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*
Mergesort: Mathematical Analysis

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>$10^7$</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>[64, 96]</td>
<td>32</td>
<td>&lt; 80?</td>
<td>yes</td>
</tr>
<tr>
<td>[64, 80]</td>
<td>16</td>
<td>&lt; 72?</td>
<td>no</td>
</tr>
<tr>
<td>[72, 80]</td>
<td>8</td>
<td>&lt; 76?</td>
<td>no</td>
</tr>
<tr>
<td>[76, 80]</td>
<td>4</td>
<td>&lt; 78?</td>
<td>yes</td>
</tr>
<tr>
<td>[76, 78]</td>
<td>2</td>
<td>&lt; 77?</td>
<td>no</td>
</tr>
<tr>
<td>[77]</td>
<td>1</td>
<td>= 77</td>
<td></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.
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.

<table>
<thead>
<tr>
<th>6</th>
<th>13</th>
<th>14</th>
<th>25</th>
<th>33</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>
</tr>
</tbody>
</table>

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

```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);   // String compare: text p. 523
    if (cmp > 0) return search(key, a, lo, mid);
    else if (cmp < 0) return search(key, a, mid+1, hi);
    else return mid;
}
```

**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.** $\log_2 N$.

\[
\begin{align*}
1 \\
2 & \rightarrow 1 \\
4 & \rightarrow 2 \rightarrow 1 \\
8 & \rightarrow 4 \rightarrow 2 \rightarrow 1 \\
16 & \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1 \\
32 & \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1 \\
64 & \rightarrow 32 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1 \\
128 & \rightarrow 64 \rightarrow 32 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1 \\
256 & \rightarrow 128 \rightarrow 64 \rightarrow 32 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1 \\
512 & \rightarrow 256 \rightarrow 128 \rightarrow 64 \rightarrow 32 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1 \\
1024 & \rightarrow 512 \rightarrow 256 \rightarrow 128 \rightarrow 64 \rightarrow 32 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1
\end{align*}
\]
Order of Growth Classifications

**Orders of growth (log-log plot)**

- **constant**
  - Function: $1$
  - Factor for doubling hypothesis: $1$
- **logarithmic**
  - Function: $\log N$
  - Factor for doubling hypothesis: $1$
- **linear**
  - Function: $N$
  - Factor for doubling hypothesis: $2$
- **linearithmic**
  - Function: $N \log N$
  - Factor for doubling hypothesis: $2$
- **quadratic**
  - Function: $N^2$
  - Factor for doubling hypothesis: $4$
- **cubic**
  - Function: $N^3$
  - Factor for doubling hypothesis: $8$
- **exponential**
  - Function: $2^N$
  - Factor for doubling hypothesis: $2^N$

*Commonly encountered growth functions*
Order of Growth Classifications

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

\[
\begin{align*}
\text{while (N > 1)} & \{ \\
N & = N / 2; \\
& \ldots \\
\} \\
\text{l} \gamma N & = \log_2 N
\end{align*}
\]

\[
\begin{align*}
\text{for (int i = 0; i < N; i++)} & \ldots \\
N & \\
\text{for (int i = 0; i < N; i++) for (int j = 0; j < N; j++)} & \ldots \\
N^2
\end{align*}
\]

\[
\begin{align*}
\text{public static void g(int N)} & \{ \\
& \text{if (N == 0) return;} \\
& g(N/2); \\
& g(N/2); \\
& \text{for (int i = 0; i < N; i++)} \\
& \quad \ldots \\
& \} \\
N \lg N
\end{align*}
\]

\[
\begin{align*}
\text{public static void f(int N)} & \{ \\
& \text{if (N == 0) return;} \\
& f(N-1); \\
& f(N-1); \\
& \} \\
2^N
\end{align*}
\]
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>