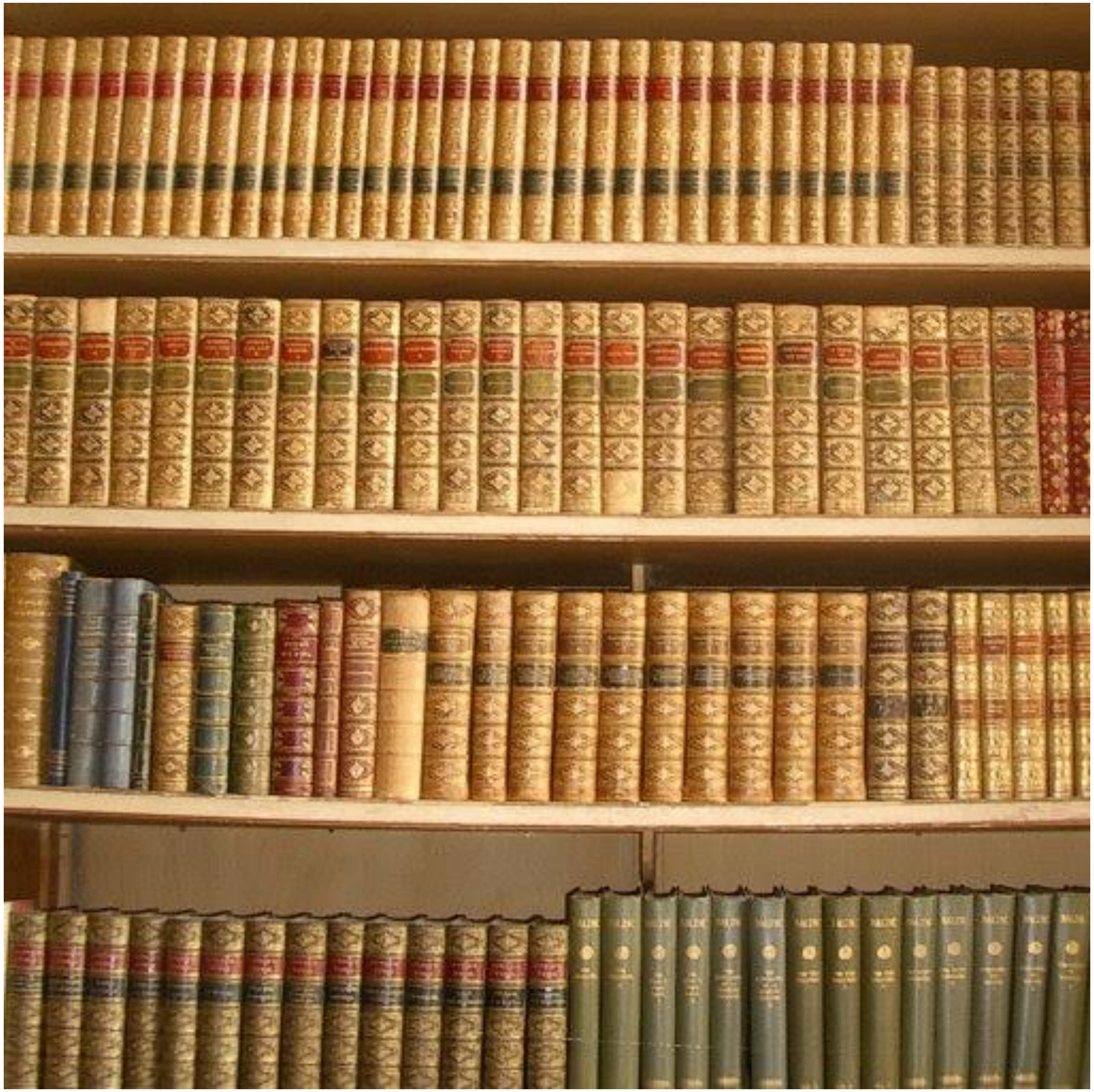
Searching and sorting



Sequential search

- Sequential search
  - Scan through array, looking for key.
  - Search hit: return array index.
  - Search miss: return -1.

```java
public static int search(String key, String[] a)
{
   for (int i = 0; i < a.length; i++)
      if (a[i].compareTo(key) == 0)
         return i;
   return -1;
}
```

Search client, exception filter

- Exception filter
  - Read sorted list of strings from a whitelist file
  - Print strings from standard input not in whitelist

```java
public static void main(String [] args)
{
   In in = new In(args[0]);
   String s = in.readAll();
   String[] words = s.split("\\s+");
   while (!StdIn.isEmpty())
   {
      String key = StdIn.readString();
      if (search(key, words) == -1)
         System.out.println(key);
   }
}
```

```
% more test.txt
bob@office
carl@beach
marvin@spam
bob@office
bob@office
mallory@spam
dave@boat
eve@airport
alice@home
```

```
% more whitelist.txt
alice@home
bob@office
carl@beach
dave@boat
```

```
% java Whitelist whitelist.txt < test.txt
marvin@spam
mallory@spam
eve@airport
```

Searching challenge 1

- Problem: A credit card company needs to whitelist 10 million customer account numbers, processing 10,000 transactions per second

- Question: Using *sequential search*, what kind of computer is needed?

  A. Toaster.

  B. Cell phone.

  C. Your laptop.

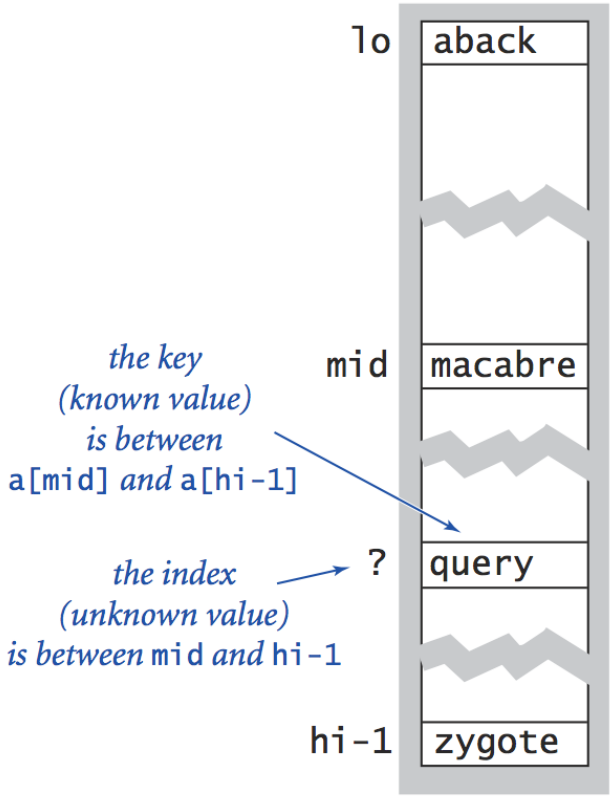
  [✓ D. Supercomputer.]

  [✓ E. Google server farm.]

Binary search

- Main idea
  - Sort the array (stay tuned)
  - Play "20 questions" to determine index with a given key
  - Examples: 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 in a sorted array (one step)

Binary search: Java implementation

- Invariant
  - Algorithm maintains: `a[lo]` ≤ `key` ≤ `a[hi-1]`
- Java library implementation: `Arrays.binarySearch()`

```
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;
}
```

"I was amazed: given ample time, only about ten percent of professional programmers were able to get this small program right. But they aren't the only ones to find this task difficult: in the history in Section 6.2.1 of his Sorting and Searching, Knuth points out that while the first binary search was published in 1946, the first published binary search without bugs did not appear until 1962."

– Jon Bentley, Programming Pearls

Binary search: mathematical analysis

- Analysis, binary search array of size N
  - Do one compare
  - Then binary search in an array of size N/2
  - N → N/2 → N/4 → N/8 → … → 1
- Question: How many times can you divide a number by 2 until you reach 1?
- Answer: $\log_2 N$

1
2 → 1
4 → 2 → 1
8 → 4 → 2 → 1
16 → 8 → 4 → 2 → 1
32 → 16 → 8 → 4 → 2 → 1
64 → 32 → 16 → 8 → 4 → 2 → 1
128 → 64 → 32 → 16 → 8 → 4 → 2 → 1
256 → 128 → 64 → 32 → 16 → 8 → 4 → 2 → 1
512 → 256 → 128 → 64 → 32 → 16 → 8 → 4 → 2 → 1
1024 → 512 → 256 → 128 → 64 → 32 → 16 → 8 → 4 → 2 → 1

Searching challenge 2

- Problem: A credit card company needs to whitelist 10 million customer account numbers, processing 10,000 transactions per second

- Question: Using *binary search*, what kind of computer is needed?

A. Toaster.

[✓ B. Cell phone.]

[✓ C. Your laptop.]

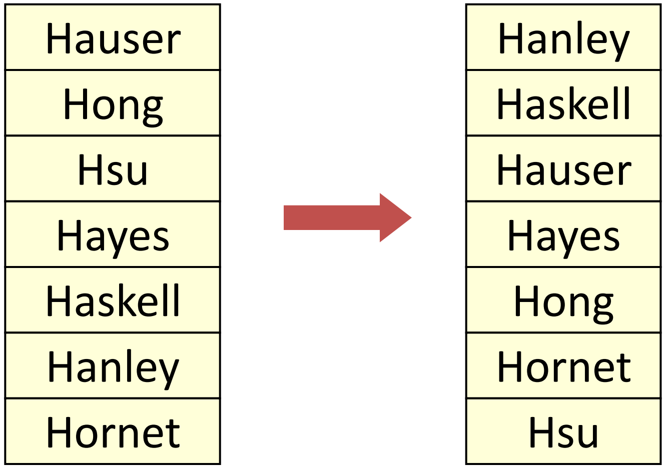
[✓ D. Supercomputer.]

[✓ E. Google server farm.]

But binary search requires a *sorted* list!

Sorting

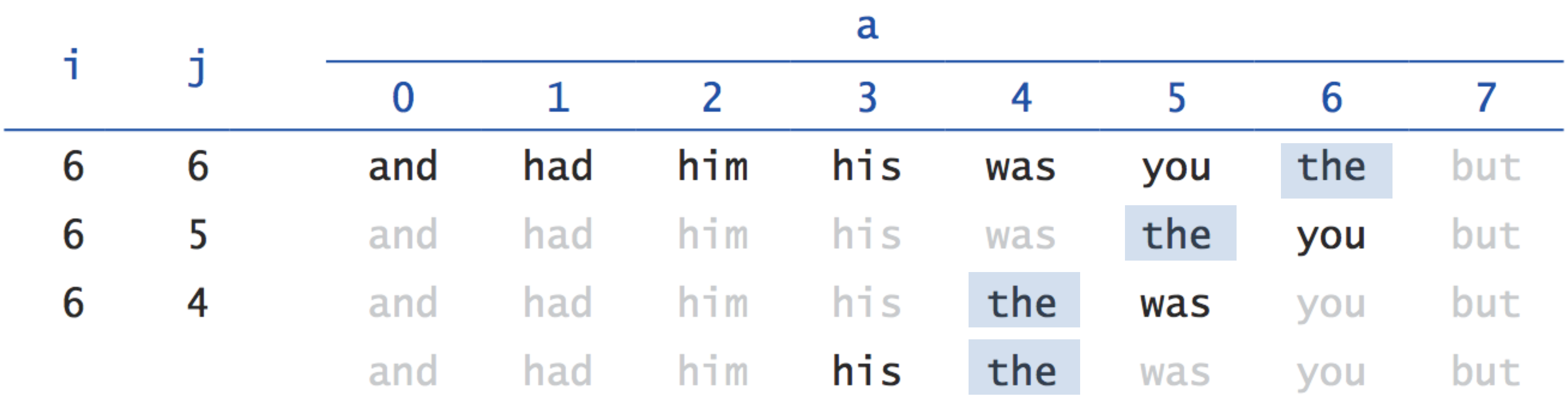
- Sorting problem
  - Rearrange N items in ascending order
- Applications
  - Statistics, databases, data compression, bioinformatics, computer graphics, scientific computing, ...



Hauser	Hanley
Hong	Haskell
Hsu	Hauser
Hayes	Hayes
Haskell	Hong
Hanley	Hornet
Hornet	Hsu

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

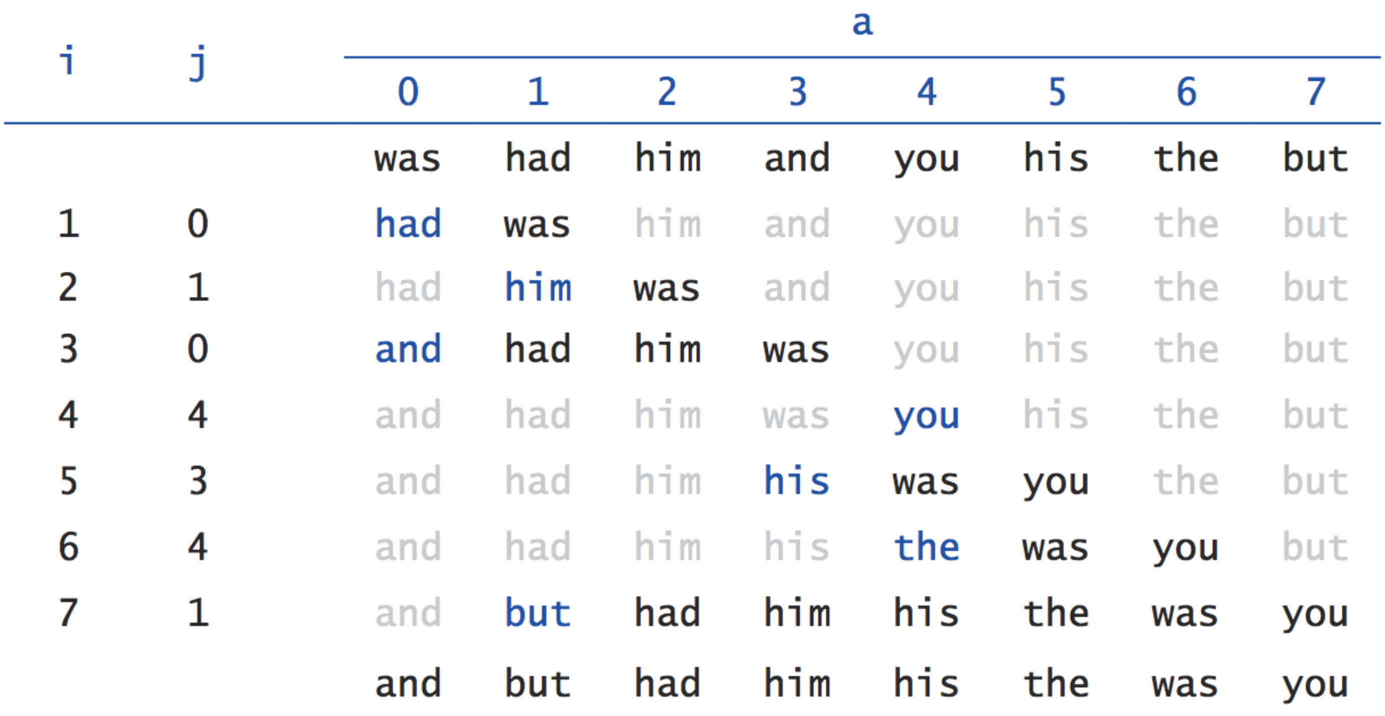


		a							
i	j	0	1	2	3	4	5	6	7
6	6	and	had	him	his	was	you	the	but
6	5	and	had	him	his	was	the	you	but
6	4	and	had	him	his	the	was	you	but
		and	had	him	his	the	was	you	but

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



i	j	a							
		0	1	2	3	4	5	6	7
		was	had	him	and	you	his	the	but
1	0	had	was	him	and	you	his	the	but
2	1	had	him	was	and	you	his	the	but
3	0	and	had	him	was	you	his	the	but
4	4	and	had	him	was	you	his	the	but
5	3	and	had	him	his	was	you	the	but
6	4	and	had	him	his	the	was	you	but
7	1	and	but	had	him	his	the	was	you
		and	but	had	him	his	the	was	you

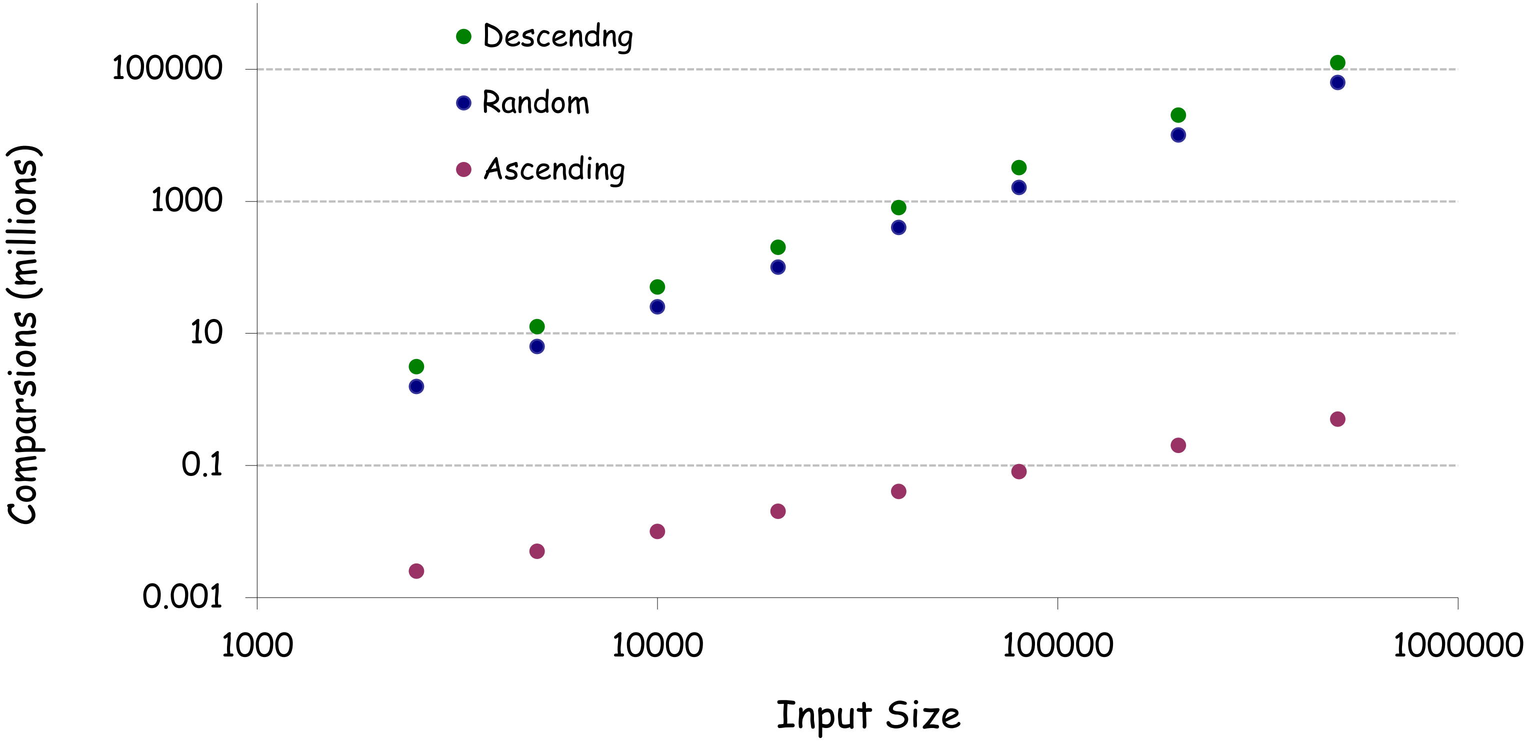
Inserting a[1] *through* a[N-1] *into position (insertion sort)*

Insertion sort: Java implementation

```java
public class Insertion
{
    public static void sort(String[] a)
    {
        for (int i = 1; i < a.length; i++)
            for (int j = i; j > 0; j--)
                if (a[j-1].compareTo(a[j]) > 0)
                    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;
    }
}
```

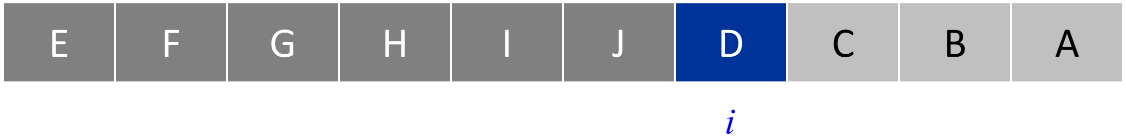
Insertion sort: empirical analysis

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

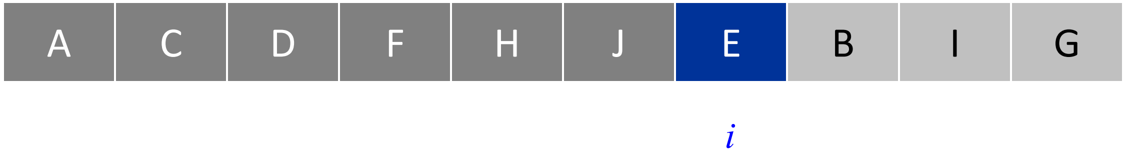


Insertion sort: mathematical analysis

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



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



Sorting challenge 1

- Problem: A credit card company sorts 10 million customer account numbers, for use with binary search.

- Question: Using *insertion sort*, what kind of computer is needed?

  A. Toaster.

  B. Cell phone.

  C. Your laptop.

  [✓ D.] Supercomputer.

  [✓ E.] Google server farm.

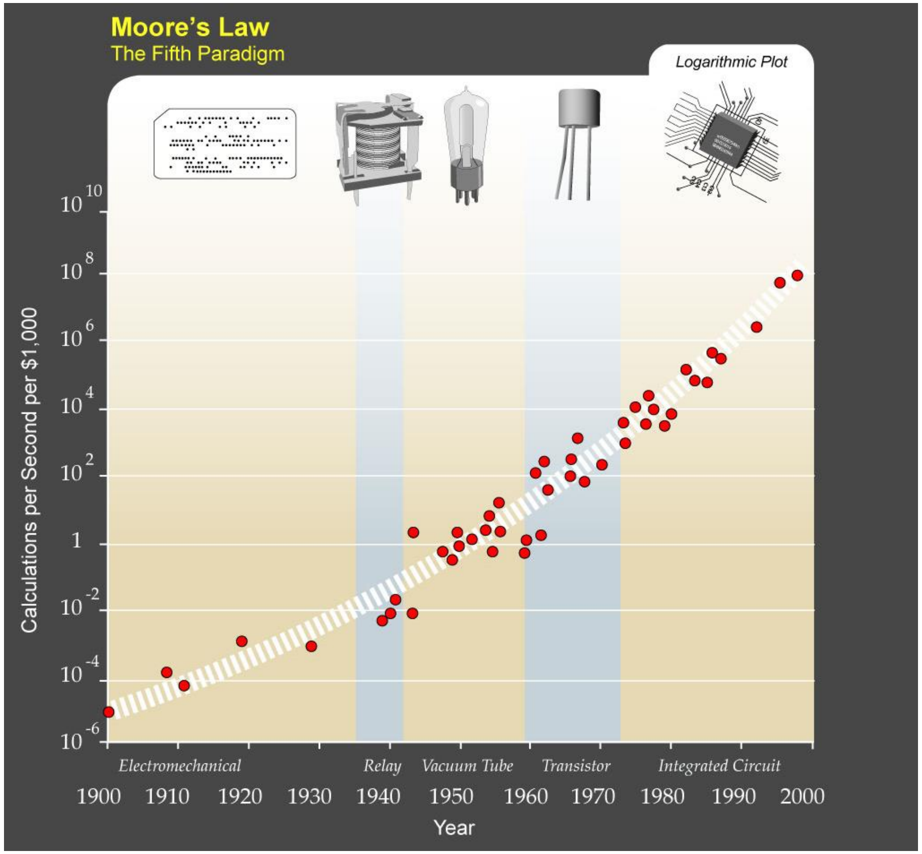
Insertion sort: lesson

- Lesson:
  - Even a supercomputer can't rescue a bad algorithm

Computer	Comparisons per second	Thousand	Million	Billion
laptop	10^7	instant	1 day	3 centuries
super	10^{12}	instant	1 second	2 weeks

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

Moore's law and algorithms

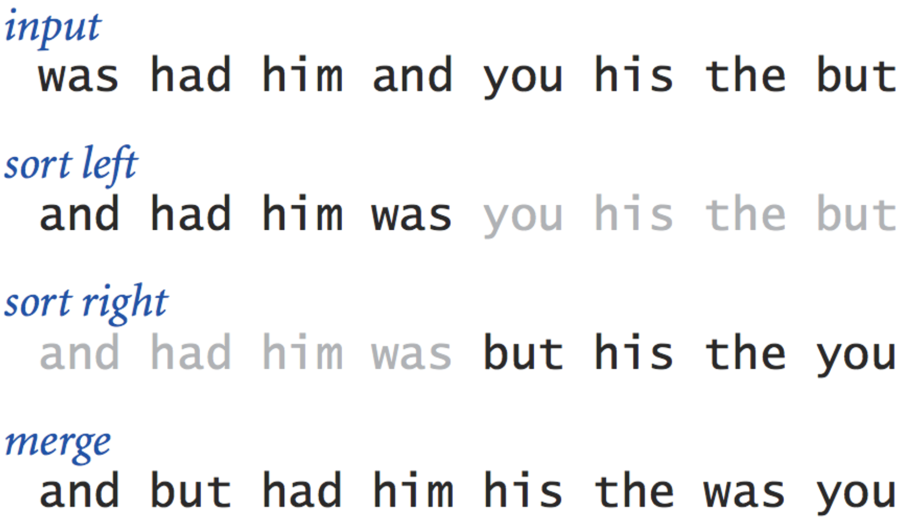
- Quadratic algorithms do not scale with technology
  - New computer may be 10x as fast.
  - But, has 10x as much memory so problem may be 10x bigger
  - With quadratic algorithm, takes 10x as long!

> *"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 algorithm
  - 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

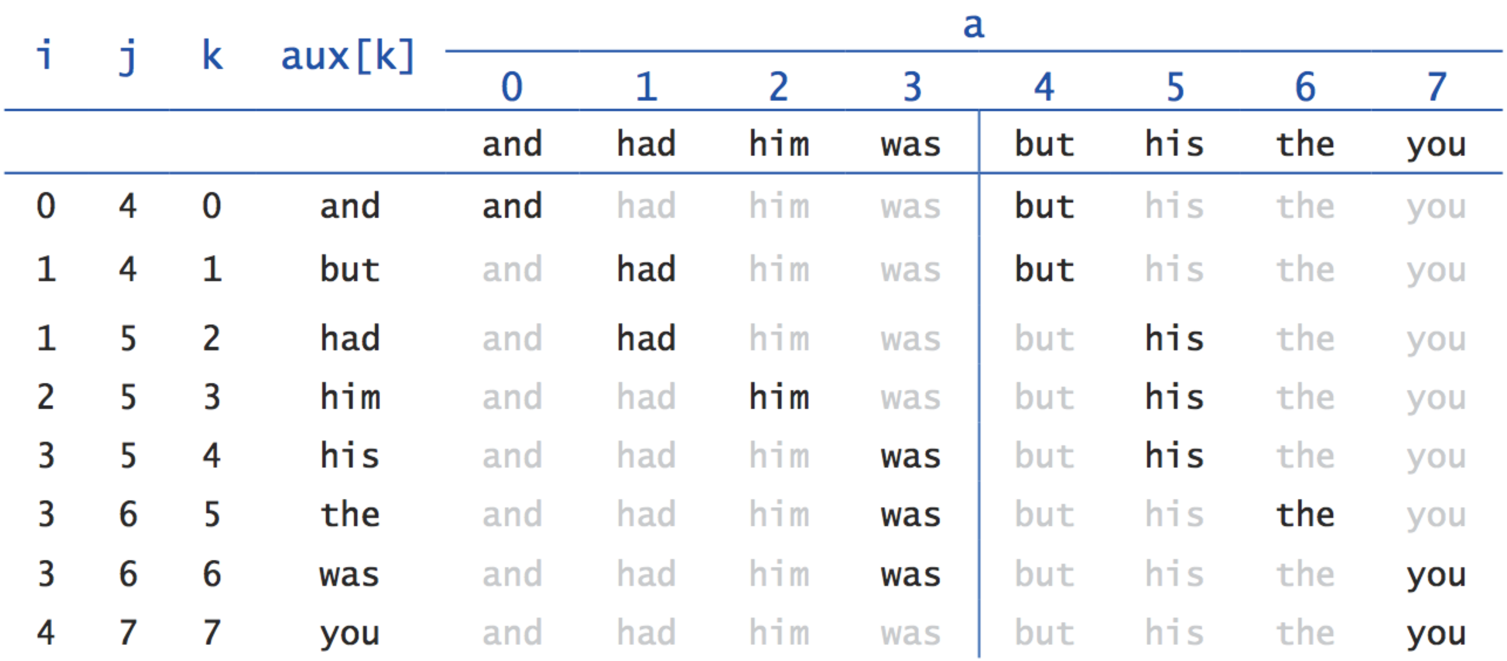
```
and had him was but his the you
```

merge

```
and but had him his the was you
```

Merging

- Merging
  - Combine two pre-sorted lists into a sorted whole.
- How to merge efficiently?
  - Use an auxiliary array



i	j	k	aux[k]	a							
				0	1	2	3	4	5	6	7
				and	had	him	was	but	his	the	you
0	4	0	and	and	had	him	was	but	his	the	you
1	4	1	but	and	had	him	was	but	his	the	you
1	5	2	had	and	had	him	was	but	his	the	you
2	5	3	him	and	had	him	was	but	his	the	you
3	5	4	his	and	had	him	was	but	his	the	you
3	6	5	the	and	had	him	was	but	his	the	you
3	6	6	was	and	had	him	was	but	his	the	you
4	7	7	you	and	had	him	was	but	his	the	you

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

Merging

- 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;
int 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];
```

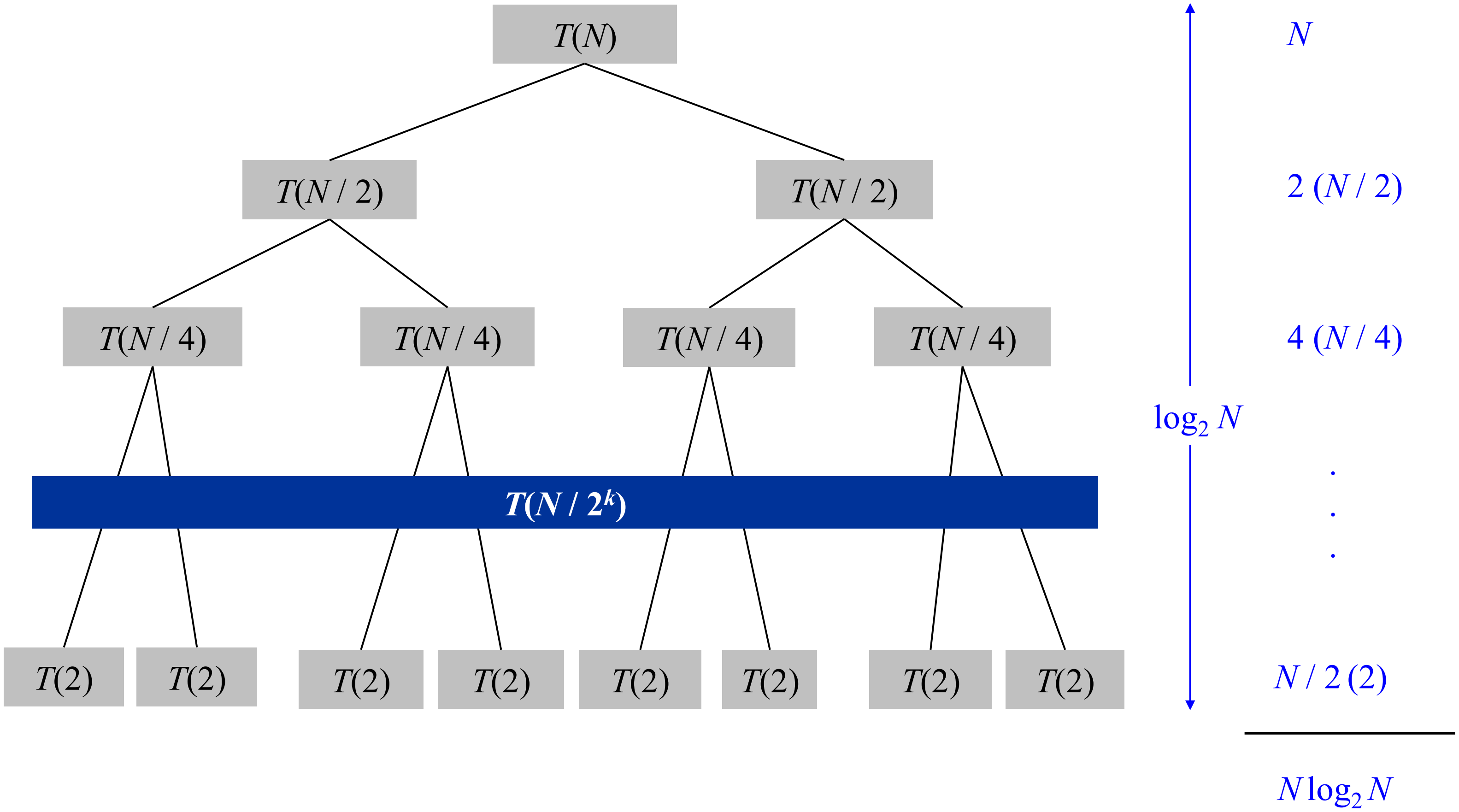
Mergesort: Java implementation

```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);

        String[] aux = new String[N];
        // merge sorted halves (see previous slide)
    }
}
```

Mergesort: mathematical analysis

- Analysis
  - To mergesort array of size N, mergesort two subarrays of size $N / 2$, and merge them together using $\leq N$ compares

Assume *N* is a power of 2



Mergesort: mathematical analysis

- Mathematical analysis

analysis	comparisons
worst	$N \log_2 N$
average	$N \log_2 N$
best	$1/2\ N \log_2 N$

N	actual	predicted
10,000	120 thousand	133 thousand
20 million	460 million	485 million
50 million	1,216 million	1,279 million

- Validation, theory agrees with observations

Sorting challenge 2

- Problem: A credit card company sorts 10 million customer account numbers, for use with binary search.

- Question: Using ***mergesort***, what kind of computer is needed?

A. Toaster.

[✓ B. Cell phone.]

[✓ C. Your laptop.]

[✓ D. Supercomputer.]

[✓ E. Google server farm.]

Sorting challenge 3

- Question: What's the fastest way to sort 1 million 32-bit integers?

http://www.youtube.com/watch?v=k4RRi_ntQc8

Mergesort: lesson

- Lesson
  - Great algorithms can be more powerful than supercomputers
  - How long to sort 1 billion things?

Computer	Compares per second	Insertion	Mergesort
laptop	10^7	3 centuries	3 hours
super	10^{12}	2 weeks	instant

N = 1 billion

Summary

- Binary search
  - Efficient algorithm to search a sorted array
- Mergesort
  - Efficient algorithm to sort an array
- Applications
  - Many many applications are enabled by fast sorting and searching