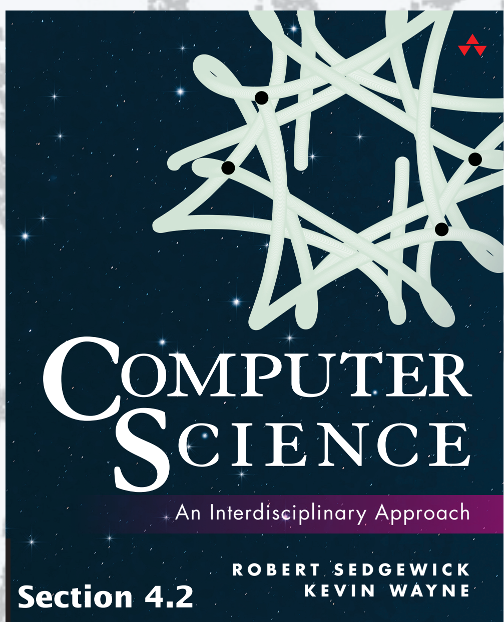



http://introcs.cs.princeton.edu

# 11. Sorting and Searching

# 11. Searching and Sorting

- A typical client
- Binary search
- Insertion sort
- Mergesort
- Longest repeated substring

# A typical client: Whitelist filter

A blacklist is a list of entities to be *rejected* for service. ← Examples: Overdrawn account, Spammers

A whitelist is a list of entities to be *accepted* for service. ← Examples: Account in good standing, Friends and relatives

Whitelist filter

- Read a list of strings from a *whitelist* file.
- Read strings from StdIn and write to StdOut only those in the whitelist.

Example. Email spam filter
(message contents omitted)

**whitelist**

```
alice@home
bob@office
carl@beach
dave@boat
```

**StdIn**

```
bob@office      ✓
carl@beach      ✓
marvin@spam
bob@office      ✓
bob@office      ✓
mallory@spam
dave@boat       ✓
eve@airport
alice@home      ✓
...
```

**StdOut**

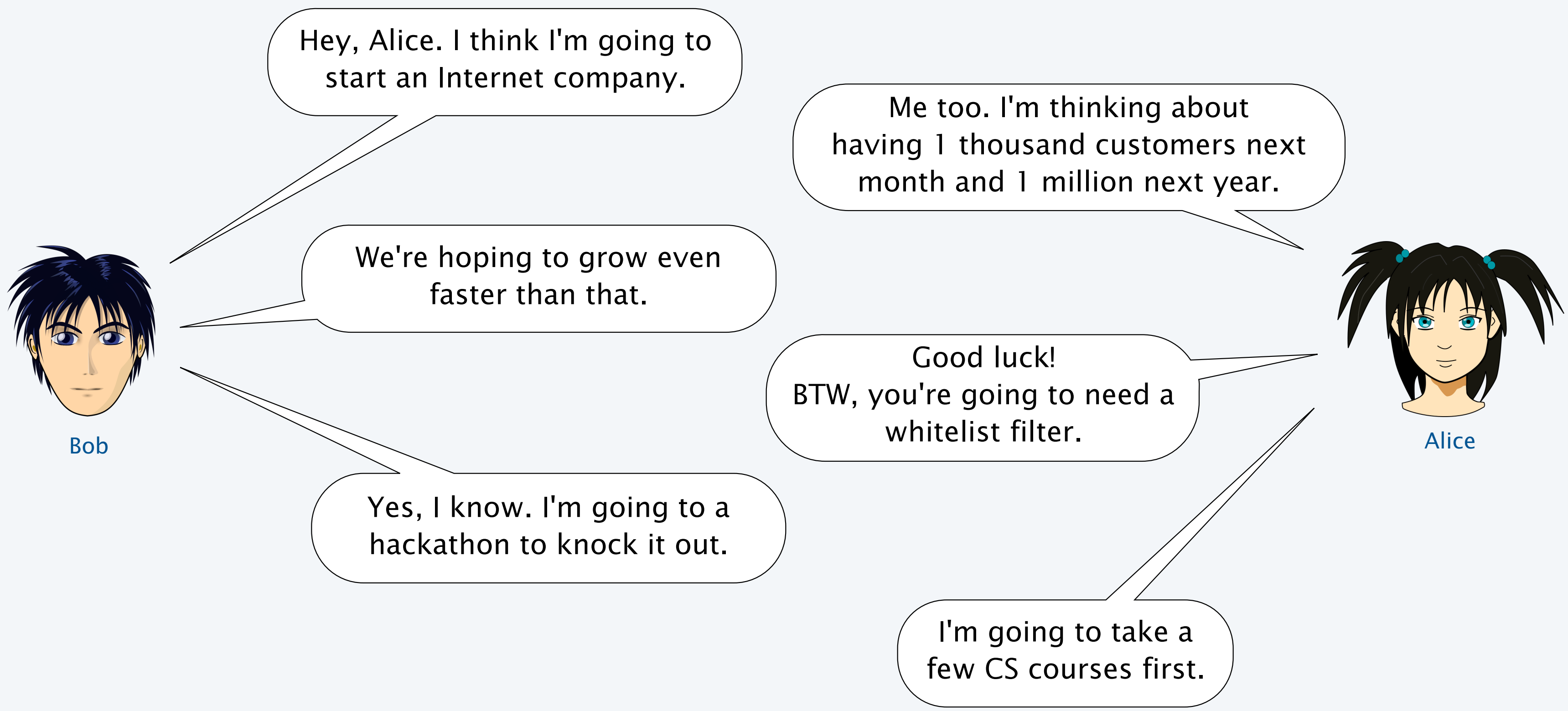
```
bob@office
carl@beach
bob@office
bob@office
dave@boat
alice@home
...
```

## Search client: Whitelist filter

```
public class WhiteFilter
{
   public static int search(String key, String[] a)
   // Search method (stay tuned).

   public static void main(String[] args)
   {
      In in = new In(args[0]);
      String[] words = in.readAllStrings();

      while (!StdIn.isEmpty())
      {
         String key = StdIn.readString();
         if (search(key, words) != -1)
            StdOut.println(key);
      }
   }
}
```

```
% more white4.txt
alice@home
bob@office
carl@beach
dave@boat

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

% java WhiteFilter white4.txt < test.txt
bob@office
carl@beach
bob@office
bob@office
dave@boat
alice@home
```

# Alice and Bob



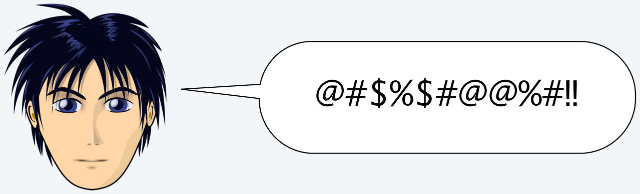
## Strawman implementation: Sequential search (first try)

### Sequential search

- Check each array entry 0, 1, 2, 3, ...
  for match with search string.
- If match found, return index of matching string.
- If not, return −1.

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

✘ Compares references, not strings!




| *i* | a[i] |
|---|---|
| 0 | alice |
| 1 | bob |
| 2 | carlos |
| 3 | carol |
| 4 | craig |
| 5 | dave |
| 6 | erin |
| 7 | eve |
| 8 | frank |
| 9 | mallory |
| 10 | oscar |
| 11 | peggy |
| 12 | trent |
| 13 | walter |
| 14 | wendy |

oscar?

## Strawman implementation: Sequential search

### Sequential search

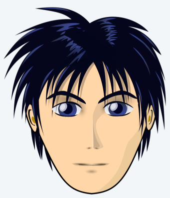
- Check each array entry 0, 1, 2, 3, ...
  for match with search string.
- If match found, return index of matching string.
- If not, return −1.

```
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;
}
```

| *i* | a[i] |
|---|---|
| 0 | alice |
| 1 | bob |
| 2 | carlos |
| 3 | carol |
| 4 | craig |
| 5 | dave |
| 6 | erin |
| 7 | eve |
| 8 | frank |
| 9 | mallory |
| 10 | oscar |
| 11 | peggy |
| 12 | trent |
| 13 | walter |
| 14 | wendy |

oscar?

Match found.
Return 10



Still, this was even easier than I thought!

# Mathematical analysis of whitelist filter using sequential search

## Model

- $N$ strings on the whitelist.
- $cN$ transactions for constant $c$.
- String length not long.

## Analysis

- A random search *hit* checks *about half* of the $N$ strings on the whitelist, on average.
- A random search *miss* checks *all* of the $N$ strings on the whitelist, on average.
- Expected order of growth of running time: $N^2$.

| whitelist |
|---|
| dobqi |
| xwnzb |
| dqwak |
| lnuqv |
| czpwx |
| bshla |
| idhld |
| utfyw |
| hafah |
| tsirv |

| transactions |
|---|
| xwnzb |
| lnuqv |
| lnuqv |
| czpwx |
| czpwx |
| dqwak |
| idhld |
| dobqi |
| dobqi |
| tsirv |
| dqwak |
| dobqi |
| idhld |
| dqwak |
| dobqi |
| lnuqv |
| xwnzb |
| idhld |
| bshla |
| xwnzb |

# Random representative inputs for searching and sorting

**Generate N random strings of length L from a given alphabet**

```
public class Generator
{
   public static String randomString(int L, String alpha)
   {
      char[] a = new char[L];
      for (int i = 0; i < L; i++)
      {
         int t = StdRandom.uniform(alpha.length());
         a[i] = alpha.charAt(t);
      }
      return new String(a);
   }
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      int L = Integer.parseInt(args[1]);
      String alpha = args[2];
      for (int i = 0; i < N; i++)
          StdOut.println(randomString(L, alpha));
   }
}
```

```
% java Generator 1 60 actg
tctatagggtcgtttgcgaagcctacacaaaagtagttgttggacaacgattgacaaaca
```

```
% java Generator 10 3 abc
bab
bab
bbb
cac
aba
abb
bab
ccb
cbc
bab
```

good chance of duplicates

```
% java Generator 15 8 0123456789
62855405
83179069
79061047
27258805
54441080
76592141
95956542
19442316
75032539
10528640
42496398
34226197
10320073
80072566
87979201
```

not much chance of duplicates

# Test client for sequential search

**Print time required for 10N searches in a whitelist of length N**

```
public class TestSS
{
   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;
   }
   public static void main(String[] args)
   {
      String[] words = StdIn.readAllStrings();
      int N = words.length;
      double start = System.currentTimeMillis()/1000.0;
      for (int i = 0; i < 10*N; i++)
      {
         String key = words[StdRandom.uniform(N)];
         if (search(key, words) == -1)
            StdOut.println(key);
       }
      double now = System.currentTimeMillis()/1000.0;
      StdOut.println(Math.round(now-start) + " seconds");
   }
}
```

random successful search (no output)

a-z = abcdefghijklmnopqrstuvwxyz

```
% java Generator 10000 10 a-z | java TestSS
3 seconds
```

generate 10,000 ten-letter words (lowercase)

print time for 100,000 searches

# Empirical tests of sequential search

| $N$ | $T_N$ (*seconds*) | $T_N/T_{N/2}$ | *transactions per second* |
|---|---|---|---|
| 10,000 | 3 | | 3,333 |
| 20,000 | 9 | | 2,222 |
| 40,000 | 35 | 3.9 | 1,143 |
| 80,000 | 149 | 4.3 | 536 |
| … | | | |
| 1.28 million | 38,500 | 4 | 34 |

more than 10.5 hours

1.28 million transactions at a rate of 34 per second and dropping

```
% java Generator 10000 ...
3 seconds
% java Generator 20000 ...
9 seconds
% java Generator 40000 ...
35 seconds
% java Generator 80000 ...
149 seconds
```

```
... = 10 a-z | java TestSS
```

## Whitelist filter scenario

- Whitelist of size $N$.
- $10N$ transactions.

**Doubling method**

Hypothesis. The running time of my program is $T_N \sim a\,N^b$.

Consequence. As $N$ increases, $T_N/T_{N/2}$ approaches $2^b$.

no need to calculate $a$ (!)

Proof: $\frac{a(2N)^b}{aN^b} = 2^b$

Validates hypothesis that order of growth is $N^2$. ← Does NOT scale.

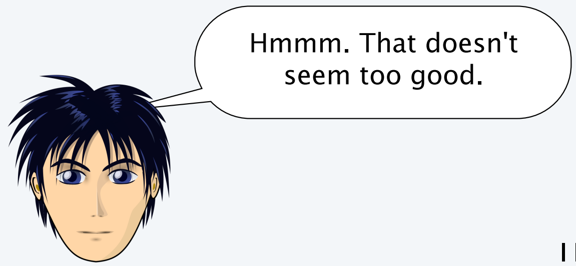


*Image sources*

https://openclipart.org/detail/25617/astrid-graeber-adult-by-anonymous-25617

https://openclipart.org/detail/169320/girl-head-by-jza

# 11. Sorting and Searching

- A typical client
- **Binary search**
- Insertion sort
- Mergesort
- Longest repeated substring

# Binary search

## Binary search

- Keep the array in sorted order (stay tuned).
- Examine the middle key.
- If it matches, return its index.
- If it is larger, search the half with lower indices.
- If it is smaller, search the half with upper indices.

```
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;
}
```

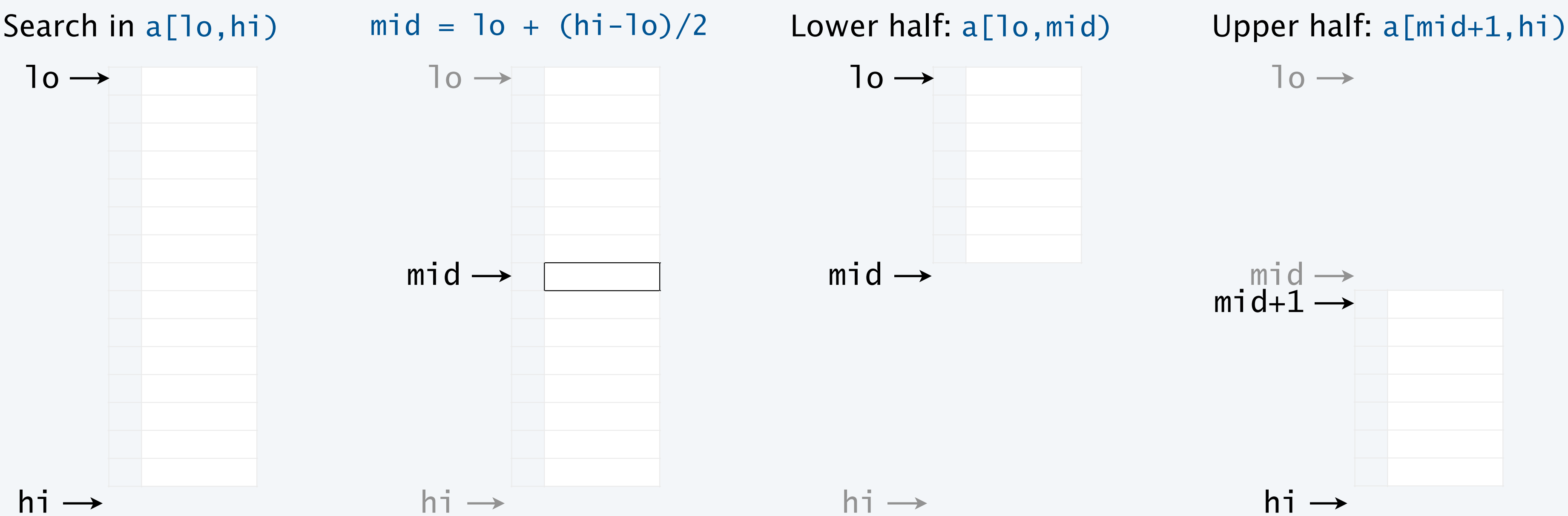
Match found.
Return 10

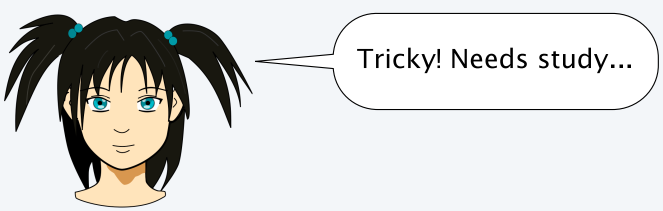
| *i* | a[i] |
|---|---|
| 0 | alice |
| 1 | bob |
| 2 | carlos |
| 3 | carol |
| 4 | craig |
| 5 | dave |
| 6 | erin |
| 7 | eve |
| 8 | frank |
| 9 | mallory |
| 10 | oscar |
| 11 | peggy |
| 12 | trent |
| 13 | walter |
| 14 | wendy |

oscar?

# Binary search arithmetic

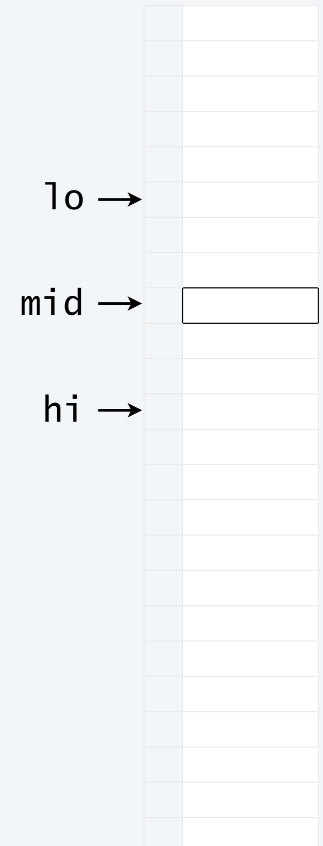
**Notation.** `a[lo,hi)` means `a[lo]`, `a[lo+1]` ... `a[hi-1]` (does not include `a[hi]`).

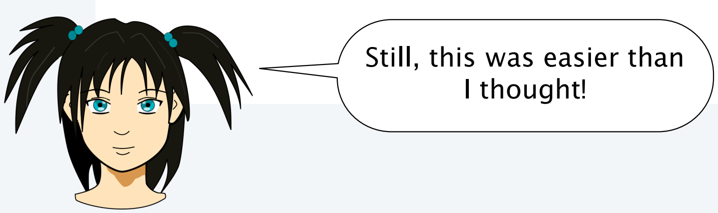


## Binary search: Java implementation

```
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;
}
```



# Recursion trace for binary search

```
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;
}
```

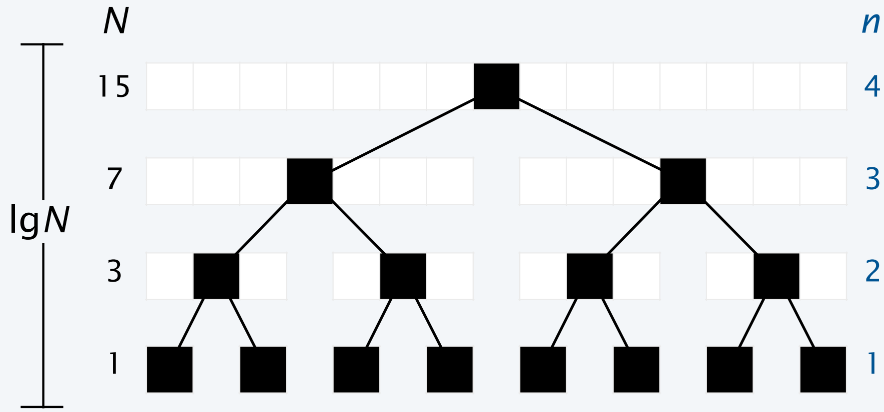
```
search("oscar")
   return            10

   search("oscar", a, 0, 15)
      mid = 7;
      > "eve"
      return          10

   search("oscar", a, 8, 15)
      mid = 11;
      < "peggy"
      return          10

   search("oscar", a, 8, 11)
      mid = 9;
      > "mallory"
      return          10

   search("oscar", a, 10, 11)
      mid = 10;
      == "oscar"
      return 10;
```

| | |
|---|---|
| 10 | oscar |

# Mathematical analysis of binary search

**Exact analysis for search miss for $N = 2^n - 1$**

- Note that $n = \lg(N+1) \sim \lg N$.
- Subarray size for 1st call is $2^n - 1$.
- Subarray size for 2nd call is $2^{n-1} - 1$.
- Subarray size for 3rd call is $2^{n-2} - 1$.
- …
- Subarray size for $n$th call is 1.
- Total # compares (one per call): $n \sim \lg N$.




Every search miss is a top-to-bottom path in this tree.

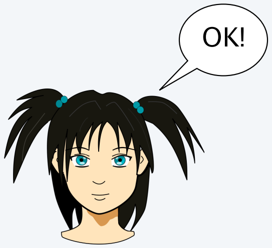
**Proposition.** Binary search uses ~lg $N$ compares for a search miss.

**Proof.** An (easy) exercise in discrete math.

**Proposition.** Binary search uses ~lg $N$ compares for a random search hit.

**Proof.** A slightly more difficult exercise in discrete math.


Interested in details? Take a course in algorithms.



# Empirical tests of binary search

**Whitelist filter scenario**

- Whitelist of size $N$.
- $10N$ transactions.

| $N$ | $T_N$ (*seconds*) | $T_N / T_{N/2}$ | *transactions per second* |
|---|---|---|---|
| 100,000 | 1 | | |
| 200,000 | 3 | | |
| 400,000 | 6 | 2 | 67,000 |
| 800,000 | 14 | 2.35 | 57,000 |
| 1,600,000 | 33 | 2.33 | 48,000 |
| | | | |
| 10.28 million | 264 | 2 | 48,000 |

```
% java Generator 100000 ...
1 seconds
% java Generator 200000 ...
3 seconds
% java Generator 400000 ...
6 seconds
% java Generator 800000 ...
14 seconds
% java Generator 1600000 ...
33 seconds
```

```
... = 10 a-z | java TestBS
a-z = abcdefghijklmnopqrstuvwxyz
```

nearly 50,000 transactions per second, and holding

Validates hypothesis that order of growth is $N \log N$.

Will scale.

# COMPUTER SCIENCE
SEDGEWICK / WAYNE

PART I: PROGRAMMING IN JAVA

CS.13.B.SearchSort.BinarySearch

COMPUTER SCIENCE
SEDGEWICK / WAYNE

PART II: ALGORITHMS, THEORY, AND MACHINES

# 11. Sorting and Searching

- A typical client
- Binary search
- **Insertion sort**
- Mergesort
- Longest repeated substring

CS.11.C.SearchSort.Insertion

# Sorting: Rearrange N items to put them in ascending order

## Applications

- Binary search
- Statistics
- Databases
- Data compression
- Bioinformatics
- Computer graphics
- Scientific computing
- ...
- [Too numerous to list]

| | |
|---|---|
| 0 | wendy |
| 1 | alice |
| 2 | dave |
| 3 | walter |
| 4 | carlos |
| 5 | carol |
| 6 | erin |
| 7 | oscar |
| 8 | peggy |
| 9 | trudy |
| 10 | eve |
| 11 | trent |
| 12 | bob |
| 13 | craig |
| 14 | frank |
| 15 | victor |

→

| | |
|---|---|
| 0 | alice |
| 1 | bob |
| 2 | carlos |
| 3 | carol |
| 4 | craig |
| 5 | dave |
| 6 | erin |
| 7 | eve |
| 8 | frank |
| 9 | oscar |
| 10 | peggy |
| 11 | trent |
| 12 | trudy |
| 13 | victor |
| 14 | walter |
| 15 | wendy |

## Pop quiz 0 on sorting

Q. What's the most efficient way to sort 1 million 32-bit integers?

# Insertion sort algorithm

## Insertion sort

- Move down through the array.
- Each item *bubbles up* above the larger ones above it.
- Everything above the current item is in order.
- Everything below the current item is untouched.

Like bubble sort, but not bubble sort.
We don't teach bubble sort any more because this is simpler and faster.

| | |
|---|---|
| 0 | wendy |
| 1 | alice |
| 2 | dave |
| 3 | walter |
| 4 | carlos |
| 5 | carol |
| 6 | erin |
| 7 | oscar |
| 8 | peggy |
| 9 | trudy |
| 10 | eve |
| 11 | trent |
| 12 | bob |
| 13 | craig |
| 14 | frank |
| 15 | victor |

# Insertion sort trace

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | wendy | alice | alice | alice | alice | alice | alice | alice | alice | alice | alice | alice | alice | alice | alice | alice |
| 1 | alice | wendy | dave | dave | carlos | carlos | carlos | carlos | carlos | carlos | carlos | carlos | bob | bob | bob | bob |
| 2 | dave | dave | wendy | walter | dave | carol | carol | carol | carol | carol | carol | carol | carlos | carlos | carlos | carlos |
| 3 | walter | walter | walter | wendy | walter | dave | dave | dave | dave | dave | dave | dave | carol | carol | carol | carol |
| 4 | carlos | carlos | carlos | carlos | wendy | walter | erin | erin | erin | erin | erin | erin | dave | craig | craig | craig |
| 5 | carol | carol | carol | carol | carol | wendy | walter | oscar | oscar | oscar | eve | eve | erin | dave | dave | dave |
| 6 | erin | erin | erin | erin | erin | erin | wendy | walter | peggy | peggy | oscar | oscar | eve | erin | erin | erin |
| 7 | oscar | oscar | oscar | oscar | oscar | oscar | oscar | wendy | walter | trudy | peggy | peggy | oscar | eve | eve | eve |
| 8 | peggy | peggy | peggy | peggy | peggy | peggy | peggy | peggy | wendy | walter | trudy | trent | peggy | oscar | frank | frank |
| 9 | trudy | trudy | trudy | trudy | trudy | trudy | trudy | trudy | trudy | wendy | walter | trudy | trent | peggy | oscar | oscar |
| 10 | eve | eve | eve | eve | eve | eve | eve | eve | eve | eve | wendy | walter | trudy | trent | peggy | peggy |
| 11 | trent | trent | trent | trent | trent | trent | trent | trent | trent | trent | trent | wendy | walter | trudy | trent | trent |
| 12 | bob | bob | bob | bob | bob | bob | bob | bob | bob | bob | bob | bob | wendy | walter | trudy | trudy |
| 13 | craig | craig | craig | craig | craig | craig | craig | craig | craig | craig | craig | craig | craig | wendy | walter | victor |
| 14 | frank | frank | frank | frank | frank | frank | frank | frank | frank | frank | frank | frank | frank | frank | wendy | walter |
| 15 | victor | victor | victor | victor | victor | victor | victor | victor | victor | victor | victor | victor | victor | victor | victor | wendy |

# Insertion sort: Java implementation

```
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].compareTo(a[j]) > 0)
               exch(a, j-1, j);
            else break;
   }

   private static void exch(String[] a, int i, int j)
   {  String t = a[i]; a[i] = a[j]; a[j] = t;  }

   public static void main(String[] args)
   {
      String[] a = StdIn.readAllStrings();
      sort(a);
      for (int i = 0; i < a.length; i++)
         StdOut.println(a[i]);
   }
}
```

```
% more names16.txt
wendy
alice
dave
walter
carlos
carol
erin
oscar
peggy
trudy
eve
trent
bob
craig
frank
victor
```

```
% java Insertion < names16.txt
alice
bob
carlos
carol
craig
dave
erin
eve
frank
oscar
peggy
trent
trudy
victor
walter
wendy
```

# Empirical tests of insertion sort

Sort random strings
- Array of length $N$.
- 10-character strings.

| $N$ | $T_N$ (*seconds*) | $T_N / T_{N/2}$ |
|---|---|---|
| 20,000 | 1 | |
| 40,000 | 4 | |
| 80,000 | 35 | 9 |
| 160,000 | 225 | 6.4 |
| 320,000 | 1019 | 4.5 |
| … | | |
| 1.28 million | 14400 | 4 |

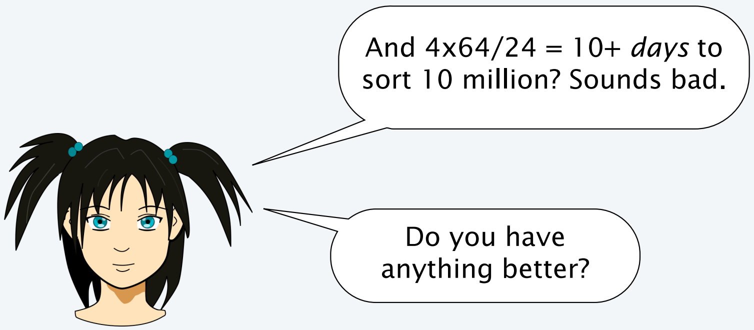
← 4 hours

```
% java Generator 20000 ...
1 seconds
% java Generator 40000 ...
4 seconds
% java Generator 80000 ...
35 seconds
% java Generator 160000 ...
225 seconds
% java Generator 320000 ...
1019 seconds
```

```
... = 10 a-z | java Insertion
a-z = abcdefghijklmnopqrstuvwxyz
```

Confirms hypothesis that order of growth is $N^2$.

↑ will NOT scale



# A rule of thumb

Moore's law. The number of transistors in an integrated circuit doubles about every 2 years.

Implications

- Memory size doubles every two years.
- Processor speed doubles every two years.

Gordon Moore
Founder of Intel
1929 –

Sedgewick's rule of thumb. It takes *a few seconds* to access every word in a computer.

| *computer* | *instructions per second* | *words of memory* |
|---|---|---|
| PDP-9 | tens of thousands | tens of thousands |
| VAX 11-780 | millions | millions |
| CRAY 1 | tens of millions | tens of millions |
| MacBook Air | billions | billions |

# Scalability

An algorithm *scales* if its running time doubles when the problem size doubles.

2x faster computer with 2x memory using an alg that scales?

- Can solve problems we're solving now in half the time.
- Can solve a 2x-sized problem in the *same* time it took to solve an x-sized problem.
- Progress.

2x faster computer with 2x memory using quadratic alg?

- Can solve problems we're solving now in half the time.
- Takes *twice* as long solve a 2x-sized problem as it took to solve an x-sized problem.
- Frustration.

| *order of growth* | *scales?* |
|---|---|
| $N$ | ✓ |
| $N \log N$ | ✓ |
| $N^2$ | ✗ |
| $N^3$ | ✗ |

Bottom line. Need algorithms that scale to keep pace with Moore's law.

*Image sources*

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

# 11. Sorting and Searching

- A typical client
- Binary search
- Insertion sort
- **Mergesort**
- Longest repeated substring

# Mergesort algorithm

## Mergesort

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

| Divide | Sort halves | Merge |
|---|---|---|
| wendy | alice | |
| alice | carlos | |
| dave | carol | |
| walter | dave | |
| carlos | erin | |
| carol | oscar | |
| erin | walter | |
| oscar | wendy | |
| peggy | bob | |
| trudy | craig | |
| eve | eve | |
| trent | frank | |
| bob | peggy | |
| craig | trent | |
| frank | trudy | |
| victor | victor | |

John von Neumann
1903–1957

## John von Neumann

- Pioneered computing (stay tuned).
- Early focus on numerical calculations.
- Invented mergesort as a test to see how his machine would measure up on other tasks.

# Merge: Java implementation

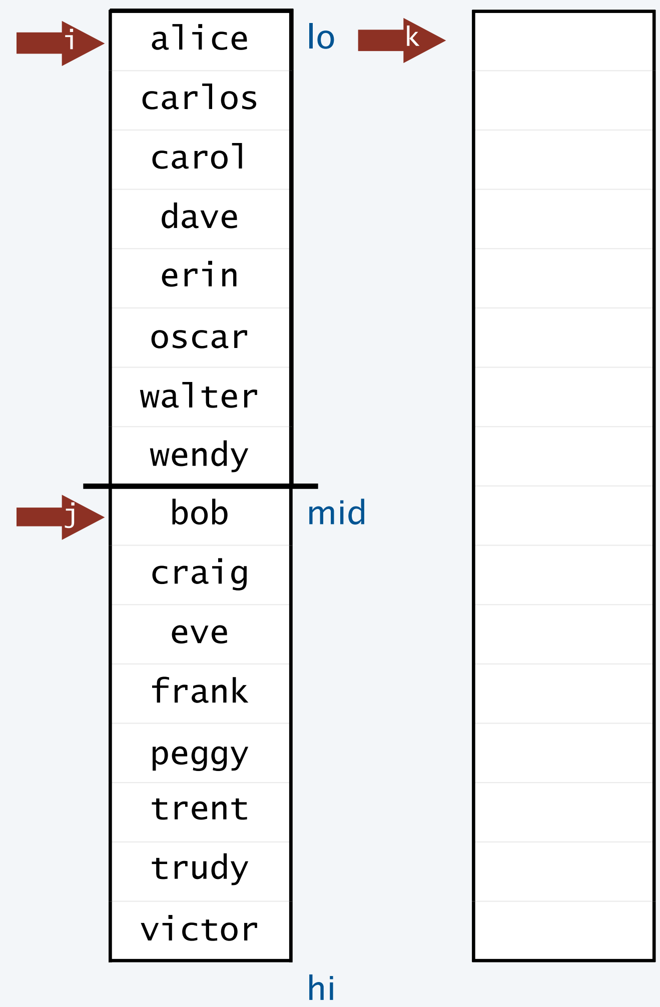
### Abstract inplace merge

- Merge a[lo, mid) with a[mid, hi).
- Use auxiliary array for result.
- Copy back when merge is complete.

t

[0,

```
   int i = lo, j = mid, N = hi - lo;
   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 into a[lo, hi)
   for (int k = 0; k < N; k++)
       a[lo + k] = aux[k];
}
```



# Mergesort: Java implementation

## Mergesort

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

```
public class Merge
{
   private static String[] aux;
   public static void merge(String[] a, int lo, int mid, int hi)
   { // See previous slide.  }
   public static void sort(String[] a)
   {
      aux = new String[a.length];  // Allocate just once!
      sort(a, 0, a.length);
   }
   public static void sort(String[] a, int lo, int hi)
   {  // Sort a[lo, hi).
      int N = hi - lo;
      if (N <= 1) return;
      int mid = lo + N/2;
      sort(a, lo, mid);
      sort(a, mid, hi);
      merge(a, lo, mid, hi);
   }
   ...
```

← same test client as for Insertion

```
% more names16.txt
wendy
alice
dave
walter
carlos
carol
erin
oscar
peggy
trudy
eve
trent
bob
craig
frank
victor
```

```
% java Merge < names16.txt
alice
bob
carlos
carol
craig
dave
erin
eve
frank
oscar
peggy
trent
trudy
victor
walter
wendy
```

# Mergesort trace

**Mergesort**

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

| | | | | |
|---|---|---|---|---|
| alice | alice | alice | alice | wendy |
| bob | carlos | dave | wendy | alice |
| carlos | carol | walter | dave | dave |
| carol | dave | wendy | walter | walter |
| craig | erin | carlos | carlos | carlos |
| dave | oscar | carol | carol | carol |
| erin | walter | erin | erin | erin |
| eve | wendy | oscar | oscar | oscar |
| frank | bob | eve | peggy | peggy |
| oscar | craig | peggy | trudy | trudy |
| peggy | eve | trent | eve | eve |
| trent | frank | trudy | trent | trent |
| trudy | peggy | bob | bob | bob |
| victor | trent | craig | craig | craig |
| walter | trudy | frank | frank | frank |
| wendy | victor | victor | victor | victor |

# Mergesort analysis

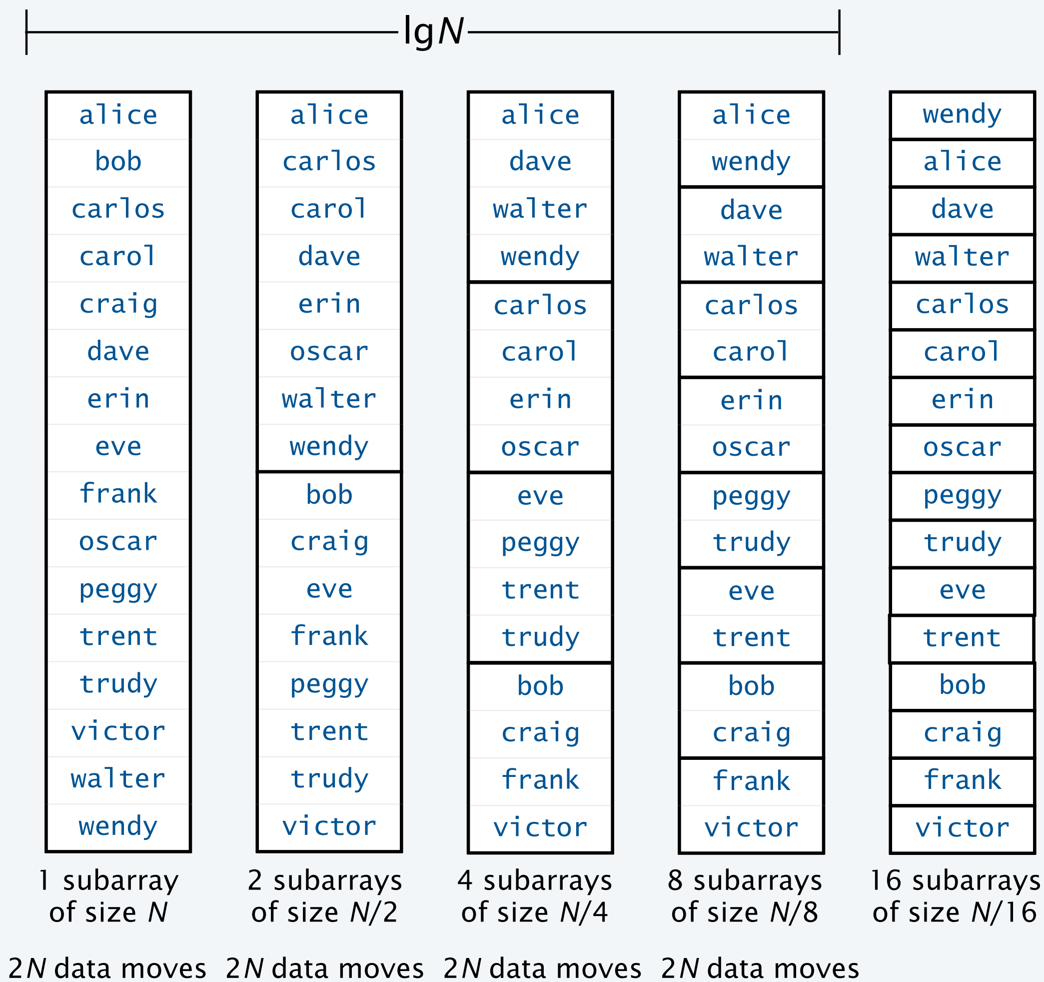
**Cost model.** Count *data moves*.

↑ # of times a string moves from one array to another

**Exact analysis for $N = 2^n$.**

- Note that $n = \lg N$.
- 1 subarray of size $2^n$.
- 2 subarrays of size $2^{n-1}$.
- 4 subarrays of size $2^{n-2}$.
- …
- $2^n$ subarrays of size 1.
- Total # data moves: $2N \lg N$.


Interested in details? Take a course in algorithms.



# Empirical tests of mergesort

Sort random strings

- Array of length *N*.
- 10-character strings.

| $N$ | $T_N$ (*seconds*) | $T_N / T_{N/2}$ |
|---|---|---|
| 1 million | 1 | |
| 2 million | 2 | |
| 4 million | 5 | 2.5 |
| 8 million | 10 | 2 |
| 16 million | 20 | 2.5 |
| … | | |
| 1.02 billion | 1280 | 2 |

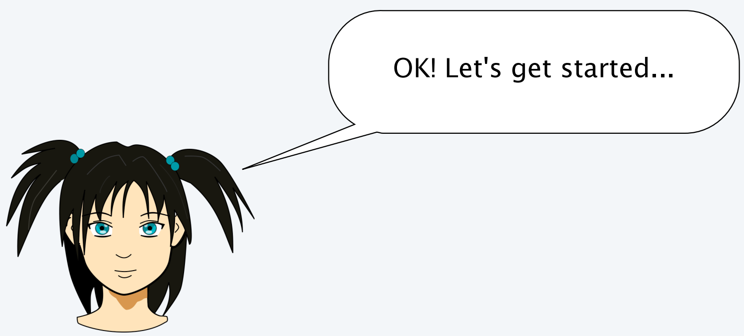
20 minutes

```
% java Generator 1000000 ...
1 seconds
% java Generator 2000000 ...
2 seconds
% java Generator 4000000 ...
5 seconds
% java Generator 8000000 ...
10 seconds
% java Generator 16000000 ...
20 seconds
```

```
... = 10 a-z | java Merge
a-z = abcdefghijklmnopqrstuvwxyz
```

Confirms hypothesis that order of growth is $N \log N$

WILL scale



# COMPUTER SCIENCE
SEDGEWICK / WAYNE

PART I: PROGRAMMING IN JAVA

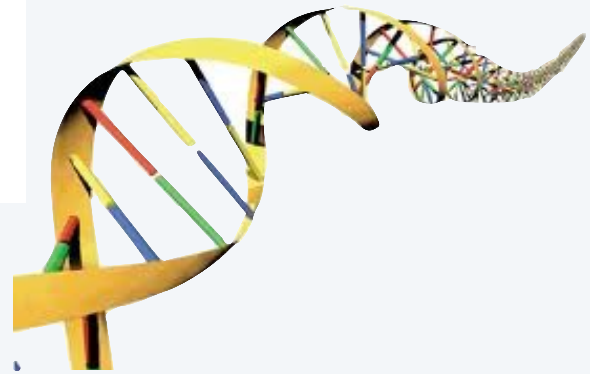
CS.13.D.SearchSort.Mergesort

# 11. Sorting and Searching

- A typical client
- Binary search
- Insertion sort
- Mergesort
- **Longest repeated substring**

## Detecting repeats in a string

**Longest repeated substring**
- Given: A string s.
- Task: Find the longest substring in s that appears at least twice.



Example 1.

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| a | a | c | a | a | g | t | t | t | a | c | a | a | g | c |

Example 2.

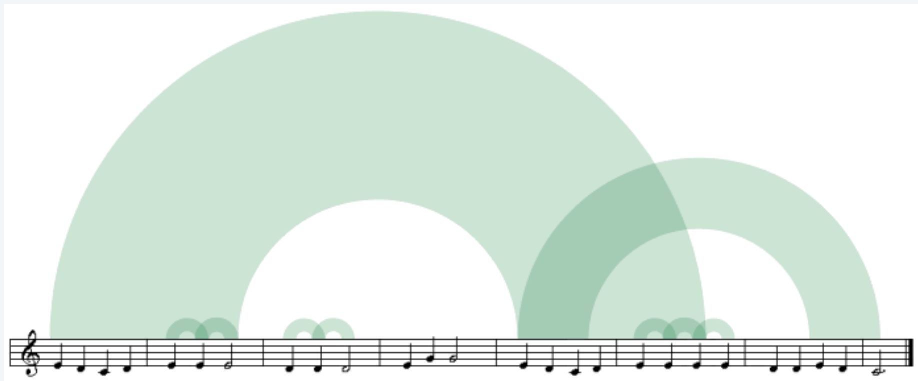
| | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| a | a | c | a | a | g | t | t | t | a | c | a | a | g | t | t | t | a | c | a | a | g | c | t | a | g | c |

Example 3 (first 100 digits of π).

| | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 3 | . | 1 | 4 | 1 | 5 | 9 | 2 | 6 | 5 | 3 | 5 | 8 | 9 | 7 | 9 | 3 | 2 | 3 | 8 | 4 |
| | 6 | 2 | 6 | 4 | 3 | 3 | 8 | 3 | 2 | 7 | 9 | 5 | 0 | 2 | 8 | 8 | 4 | 1 | 9 | 7 |
| | 1 | 6 | 9 | 3 | 9 | 9 | 3 | 7 | 5 | 1 | 0 | 5 | 8 | 2 | 0 | 9 | 7 | 4 | 9 | 4 |
| | 4 | 5 | 9 | 2 | 3 | 0 | 7 | 8 | 1 | 6 | 4 | 0 | 6 | 2 | 8 | 6 | 2 | 0 | 8 | 9 |
| | 9 | 8 | 6 | 2 | 8 | 0 | 3 | 4 | 8 | 2 | 5 | 3 | 4 | 2 | 1 | 1 | 7 | 0 | 6 | 9 |

# LRS example: repetitive structure in music

**Mary had a little lamb**



**Für Elise**

# LRS applications

Analysts seek repeated sequences in real-world data because they are causal.

### Example 1: Digits of π

- Q. Are they “random” ?
- A. No, but we can’t tell the difference.
- Ex. Length of LRS in first 10 million digits is 14.

```
3.141592653589793238462643383279502884
197169399375105820974944592307816406286
```

### Example 2: Cryptography

- Find LRS.
- Check for “known” message header information.
- Break code.

```
11001001001111011011100101101011100110
00101111110100100001001101001011110011
00100111111101110000010101100010000111
01010011010000111100100110011101111111
01010000010000100010100101010001100000
10111100010010011010110111100011010011
01110011110101111001000100111010101110
10000010100100010001101010101110000000
10110000001001110001011101101001010110 0
```

### Example 3: DNA

- Find LRS
- Look somewhere else for causal mechanisms
- Ex. Chromosome 11 has 7.1 million nucleotides

```
tgactaatccagtatccagggcaaattaggttacccac
gtgattacgagaggttccgccgctaatcgggtgcgtcc
gaaacgtatgccctcttctgctcgatgtgattggccgg
cctgtgtcatgccggcacttaaacgatcaaatagtgaa
aatcaaaatcgccggtctgtgagcctagcggatgcaag
atgggcgtacatgcccagcccaccttcggaccgagctg
cgcgtagggccgtagtgctaaagtctgagaatacccca
gtcgttcgttgaggcgcacgtctatgcataatttatgg
aggtcagtgctcttcagaggttgcagtttactctattc
```

# Warmup: Longest common prefix

**Longest common prefix**

- Given: Two strings string s and t.
- Task: Find the longest substring that appears at the beginning of both

Example.

```
a a c a a g t t t a c a a g c
a a c a a g t t t a c a a g t t t a c a a g c t a g c
```

**Implementation (easy)**

```
private static String lcp(String s, String t)
{
   int N = Math.min(s.length(), t.length());
   for (int i = 0; i < N; i++)
      if (s.charAt(i) != t.charAt(i))
         return s.substring(0, i);
   return s.substring(0, N);
}
```

# LRS: Brute-force implementation

```
public class LRS
{
   public static String lcp(String s)
   { // See previous slide. }

   public static String lrs(String s)
   {
      int N = s.length();
      String lrs = "";
      for (int i = 0; i < N; i++)
         for (int j = i+1; j < N; j++)
         {
            String x = lcp(s.substring(i, N), s.substring(j, N));
            if (x.length() > lrs.length()) lrs = x;
         }
      return lrs;
   }
   public static void main(String[] args)
   {
      String s= StdIn.readAll();
      StdOut.println(lrs(s));
   }
}
```

```
% more tiny.txt
aacaagtttacaagc

% java LRSbrute
acaag
```

## Analysis

- $\sim N^2/2$ calls on `lcp()`.
- Obviously does not scale.

# LRS: An efficient solution that uses sorting

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| a | a | c | a | a | g | t | t | t | a | c | a | a | g | c |

**1. Form suffix strings**

| | |
|---|---|
| 0 | a a c a a g t t t a c a a g c |
| 1 | a c a a g t t t a c a a g c |
| 2 | c a a g t t t a c a a g c |
| 3 | a a g t t t a c a a g c |
| 4 | a g t t t a c a a g c |
| 5 | g t t t a c a a g c |
| 6 | t t t a c a a g c |
| 7 | t t a c a a g c |
| 8 | t a c a a g c |
| 9 | a c a a g c |
| 10 | c a a g c |
| 11 | a a g c |
| 12 | a g c |
| 13 | g c |
| 14 | c |

**2. Sort suffix strings**

| | |
|---|---|
| 0 | a a c a a g t t t a c a a g c |
| 11 | a a g c |
| 3 | a a g t t t a c a a g c |
| 9 | a c a a g c |
| 1 | a c a a g t t t a c a a g c |
| 12 | a g c |
| 4 | a g t t t a c a a g c |
| 14 | c |
| 10 | c a a g c |
| 2 | c a a g t t t a c a a g c |
| 13 | g c |
| 5 | g t t t a c a a g c |
| 8 | t a c a a g c |
| 7 | t t a c a a g c |
| 6 | t t t a c a a g c |

**3. Find longest LCP among adjacent entries.**

# LRS: Suffix array implementation

**Form suffix strings**

**Sort suffix strings**

**Find longest LCP among adjacent entries.**

```
public static String lrs(String s)
{
   int N = s.length();
   String[] suffixes = new String[N];
   for (int i = 0; i < N; i++)
      suffixes[i] = s.substring(i, N);

   Merge.sort(suffixes);

   String lrs = "";
   for (int i = 0; i < N-1; i++)
   {
      String x = lcp(suffixes[i], suffixes[i+1]);
      if (x.length() > lrs.length()) lrs = x;
   }
   return lrs;
}
```

```
% more tiny.txt
aacaagtttacaagc

% java LRS
acaag
```

## Analysis

- *N* calls on `substring()`.
- *N* calls on `lcp()`.
- Potentially scales.

# LRS: Empirical analysis (1995-2012)

**Model**

- Alphabet: actg.
- *N*-character random strings.

```
% java Generator 1 1000000 actg | java LRS
2 seconds
% java Generator 1 10000000 actg | java LRS
21 seconds
```

**Doubling**

| $N$ | $T_N$ | $T_N / T_{N/2}$ |
|---|---|---|
| 2,000,000 | 3 | |
| 4,000,000 | 7 | 2.3 |
| 8,000,000 | 16 | 2.3 |
| 16,000,000 | 39 | 2.4 |

**x10**

| $N$ | $T_N$ | $T_N / T_{N/10}$ |
|---|---|---|
| 1,000,000 | 2 | |
| 10,000,000 | 21 | 10 |

Confirms hypothesis that the order of growth is $N \log N$ (for the sort).

Bottom line. Scales with the size of the input and enables new research and development.

## LRS: Empirical analysis (since 2012)

**Model**

- Alphabet: actg.
- *N*-character random strings.

```
% java Generator 1 10000 actg | java LRS
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
        at java.util.Arrays.copyOfRange(Arrays.java:3664)
        at java.lang.String.<init>(String.java:201)
        at java.lang.String.substring(String.java:1956)
        at LRS.LRS(LRS.java:17)
        at LRS.main(LRS.java:33)
```

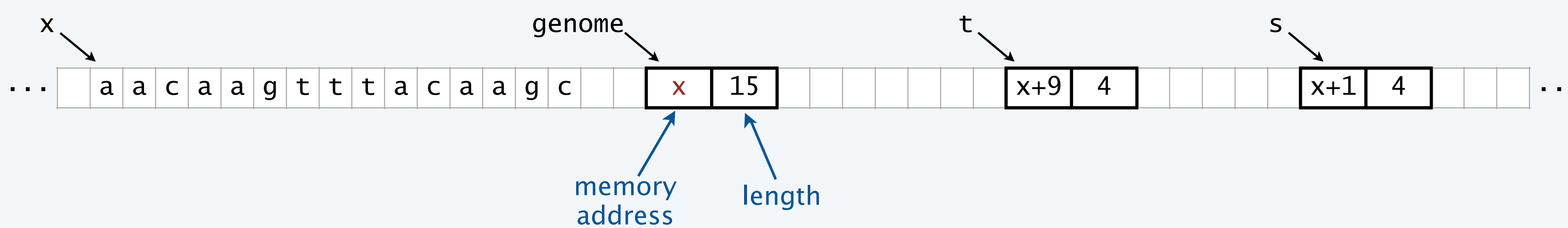
?

Change in the system *breaks a working program* (not good).

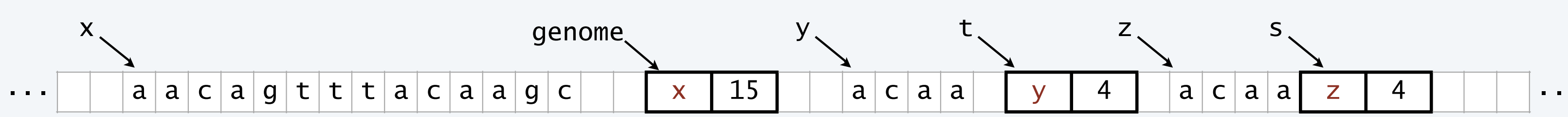
## Explanation: Two alternatives for implementing substrings

1. Refer to original string (1995-2102).
   - No need to copy characters.
   - *Constant* time and space.

```
String genome = "aacaagtttacaagc";
String s = genome.substring(1, 5);
String t = genome.substring(9, 13);
```



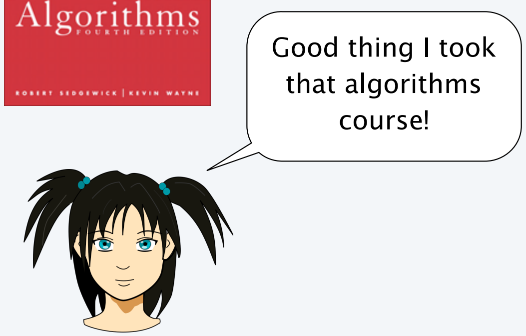

2. Copy the characters to make a new string (since 2012).
   - Allows potential to free up memory when the original string is no longer needed.
   - *Linear* time and space (in the length of the substring).



## Fixing the LRS implementation

Implement our own constant-time suffix operation.

- Imitate old `substring()` implementation.
- Need `compareTo()` to enable sort.
- (Details in *Algorithms*)


```
% java Generator 1 1000000 actg | java LRSfixed
2 seconds
% java Generator 1 10000000 actg | java LRSfixed
21 seconds
```

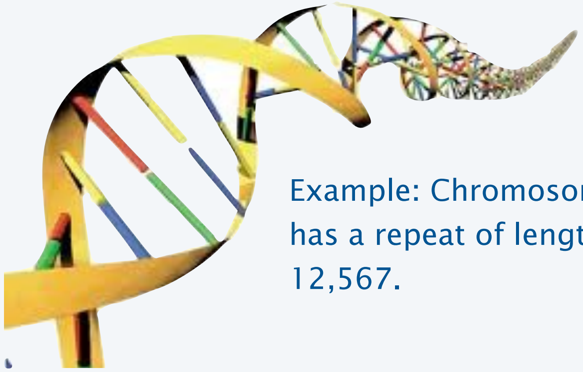
Lesson. Trust the *algorithm*, not the system.

Bottom line. New research and development can continue.

# Final note on LRS implementation

**Long repeats**

- More precise analysis reveals that running time is *quadratic* in the length of the longest repeat.
- Model has no long repeats.
- Real data may have long repeats.
- Linear time algorithm (guarantee) is known.



Example: Chromosome 11 has a repeat of length 12,567.

## Summary

Binary search. Efficient algorithm to search a sorted array.

Mergesort. Efficient algorithm to sort an array.

Applications. Many, many, many things are enabled by fast sort and search.

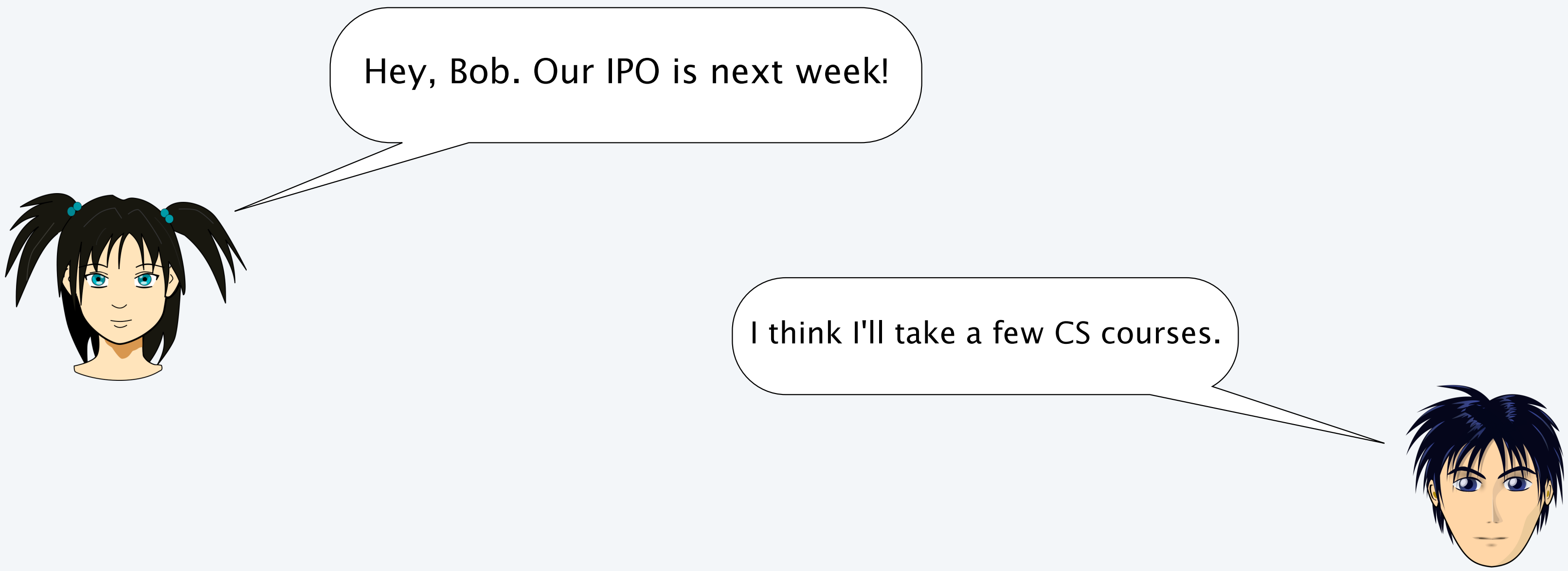
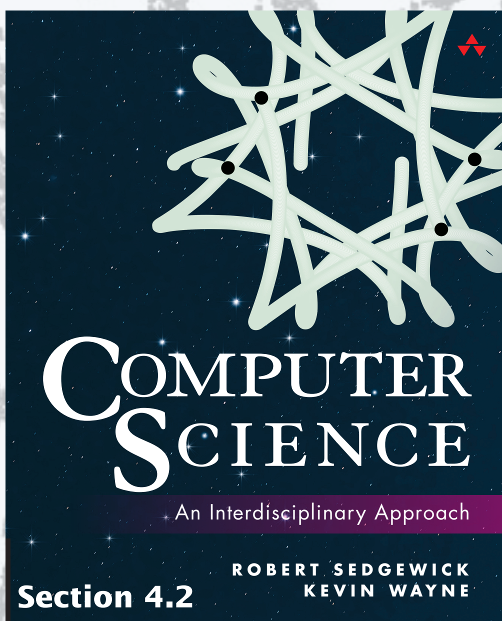


*Image sources*

https://www.bewitched.com/match

COMPUTER SCIENCE
SEDGEWICK / WAYNE

PART II: ALGORITHMS, THEORY, AND MACHINES




http://introcs.cs.princeton.edu

# 11. Sorting and Searching