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.

A **whitelist** is a list of entities to be *accepted* for service.

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)

<table>
<thead>
<tr>
<th>whitelist</th>
<th>StdIn</th>
<th>StdOut</th>
</tr>
</thead>
<tbody>
<tr>
<td>alice@home bob@office carl@beach dave@boat</td>
<td>bob@office carl@beach marvin@spam bob@office mallory@spam dave@boat eve@airport alice@home ...</td>
<td>✓ ✓ ✓ ✓ ✓ ✓...</td>
</tr>
</tbody>
</table>
Search client: Whitelist filter

```java
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
Hey, Alice. I think I'm going to start an Internet company.

Me too. I'm thinking about having 1 thousand customers next month and 1 million next year.

We're hoping to grow even faster than that.

Good luck! BTW, you're going to need a whitelist filter.

Yes, I know. I'm going to a hackathon to knock it out.

I'm going to take a few CS courses first.
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 $-1$.

```java
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!

<table>
<thead>
<tr>
<th>$i$</th>
<th>$a[i]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>alice</td>
</tr>
<tr>
<td>1</td>
<td>bob</td>
</tr>
<tr>
<td>2</td>
<td>carlos</td>
</tr>
<tr>
<td>3</td>
<td>carol</td>
</tr>
<tr>
<td>4</td>
<td>craig</td>
</tr>
<tr>
<td>5</td>
<td>dave</td>
</tr>
<tr>
<td>6</td>
<td>erin</td>
</tr>
<tr>
<td>7</td>
<td>eve</td>
</tr>
<tr>
<td>8</td>
<td>frank</td>
</tr>
<tr>
<td>9</td>
<td>mallory</td>
</tr>
<tr>
<td>10</td>
<td>oscar</td>
</tr>
<tr>
<td>11</td>
<td>peggy</td>
</tr>
<tr>
<td>12</td>
<td>trent</td>
</tr>
<tr>
<td>13</td>
<td>walter</td>
</tr>
<tr>
<td>14</td>
<td>wendy</td>
</tr>
</tbody>
</table>

@#$%#@%##!
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.

```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;
}
```

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: $N^2$. 

<table>
<thead>
<tr>
<th>whitelist</th>
<th>dobqi</th>
<th>lnuqv</th>
</tr>
</thead>
<tbody>
<tr>
<td>xwnzb</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dqwak</td>
<td></td>
<td>czpwx</td>
</tr>
<tr>
<td>lnuqv</td>
<td></td>
<td>czpwx</td>
</tr>
<tr>
<td>czpwx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bshla</td>
<td></td>
<td>dqwak</td>
</tr>
<tr>
<td>idh1d</td>
<td></td>
<td>idh1d</td>
</tr>
<tr>
<td>utfyw</td>
<td></td>
<td>dobqi</td>
</tr>
<tr>
<td>hafah</td>
<td></td>
<td>dobqi</td>
</tr>
<tr>
<td>tsirv</td>
<td></td>
<td>tsirv</td>
</tr>
<tr>
<td></td>
<td>dobqi</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>dqwak</td>
</tr>
<tr>
<td></td>
<td>dobqi</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>idh1d</td>
</tr>
<tr>
<td></td>
<td>dobqi</td>
<td>dqwak</td>
</tr>
<tr>
<td></td>
<td></td>
<td>dobqi</td>
</tr>
<tr>
<td></td>
<td></td>
<td>lnuqv</td>
</tr>
<tr>
<td></td>
<td>xwnzb</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>idh1d</td>
</tr>
<tr>
<td></td>
<td>bshla</td>
<td></td>
</tr>
<tr>
<td></td>
<td>xwnzb</td>
<td></td>
</tr>
</tbody>
</table>
Random representative inputs for searching and sorting

Generate N random strings of length L from a given alphabet

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

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

% java Generator 1 60 actg
tctatagggtcgtttcgaagctacacaaaaagtagttgattgacaacgattgacaaca
Test client for sequential search

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

```java
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");
    }
}
```

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

generate 10,000 ten-letter words (lowercase)
print time for 100,000 searches
random successful search (no output)
a-z = abcdefghijklmnopqrstuvwxyz
Empirical tests of sequential search

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

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

Does NOT scale.

% 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

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

Hmmm. That doesn't seem too good.

more than 10.5 hours

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

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

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

no need to calculate $a$ & $b$.

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

= 10 a-z | java TestSS
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

- 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.

```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;
}
```

oscar?

Match found. Return 10
Binary search arithmetic

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

Search in \( a[lo,hi) \)

\[
\begin{align*}
lo & \quad lo \\
mid & \quad mid \\
hi & \quad hi
\end{align*}
\]

\[
\begin{align*}
lo & \quad lo \\
mid & \quad mid \\
hi & \quad hi
\end{align*}
\]

Lower half: \( a[lo,mid) \)

Upper half: \( a[mid+1,hi) \)

Tricky! Needs study...
Binary search: Java implementation

```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;
}
```

Still, this was easier than I thought!
Recursion trace for binary search

```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;
}
```

```
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;
```
Mathematical analysis of binary search

**Exact analysis for search miss for** \( N = 2^n - 1 \)

- Note that \( n = \lceil \log(N+1) \rceil \sim \lg N \).
- 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 \( 2^{n-2} - 1 \).
- \( \ldots \)
- Subarray size for \( n \)th call is \( 1 \).
- Total \# compares (one per call): \( n \sim \lg N \).

**Proposition.** Binary search uses \( \sim \lg N \) compares for a search miss.

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

**Proposition.** Binary search uses \( \sim \lg N \) compares for a random search hit.

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

![Binary search tree diagram](image)
Empirical tests of binary search

Whitelist filter scenario

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

<table>
<thead>
<tr>
<th>$N$</th>
<th>$T_N$ (seconds)</th>
<th>$T_N/T_{N/2}$</th>
<th>transactions per second</th>
</tr>
</thead>
<tbody>
<tr>
<td>100,000</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>200,000</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>400,000</td>
<td>6</td>
<td>2</td>
<td>67,000</td>
</tr>
<tr>
<td>800,000</td>
<td>14</td>
<td>2.35</td>
<td>57,000</td>
</tr>
<tr>
<td>1,600,000</td>
<td>33</td>
<td>2.33</td>
<td>48,000</td>
</tr>
<tr>
<td>10.28 million</td>
<td>264</td>
<td>2</td>
<td>48,000</td>
</tr>
</tbody>
</table>

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

Will scale.

Great! But how do I get the list into sorted order at the beginning?

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

nearly 50,000 transactions per second, and holding
11. Sorting and Searching

- A typical client
- Binary search
- Insertion sort
- Mergesort
- Longest repeated substring
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]
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.
## Insertion sort trace

<table>
<thead>
<tr>
<th></th>
<th>wendy</th>
<th>alice</th>
<th>alice</th>
<th>alice</th>
<th>alice</th>
<th>alice</th>
<th>alice</th>
<th>alice</th>
<th>alice</th>
<th>alice</th>
<th>alice</th>
<th>alice</th>
<th>alice</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>alice</td>
<td>wendy</td>
<td>dave</td>
<td>dave</td>
<td>carlos</td>
<td>carlos</td>
<td>carlos</td>
<td>carlos</td>
<td>carlos</td>
<td>carlos</td>
<td>carlos</td>
<td>carlos</td>
<td>carlos</td>
</tr>
<tr>
<td>2</td>
<td>dave</td>
<td>dave</td>
<td>wendy</td>
<td>walter</td>
<td>dave</td>
<td>carol</td>
<td>carol</td>
<td>carol</td>
<td>carol</td>
<td>carol</td>
<td>carol</td>
<td>carol</td>
<td>carol</td>
</tr>
<tr>
<td>3</td>
<td>walter</td>
<td>walter</td>
<td>walter</td>
<td>wendy</td>
<td>dave</td>
<td>dave</td>
<td>dave</td>
<td>dave</td>
<td>dave</td>
<td>dave</td>
<td>carol</td>
<td>carol</td>
<td>carol</td>
</tr>
<tr>
<td>4</td>
<td>carlos</td>
<td>carlos</td>
<td>carlos</td>
<td>carlos</td>
<td>wendy</td>
<td>walter</td>
<td>ernin</td>
<td>ernin</td>
<td>ernin</td>
<td>ernin</td>
<td>ernin</td>
<td>ernin</td>
<td>dave</td>
</tr>
<tr>
<td>5</td>
<td>carol</td>
<td>carol</td>
<td>carol</td>
<td>carol</td>
<td>carol</td>
<td>wendy</td>
<td>walter</td>
<td>ernin</td>
<td>ernin</td>
<td>ernin</td>
<td>ernin</td>
<td>ernin</td>
<td>dave</td>
</tr>
<tr>
<td>6</td>
<td>ernin</td>
<td>ernin</td>
<td>ernin</td>
<td>ernin</td>
<td>ernin</td>
<td>wendy</td>
<td>walter</td>
<td>ernin</td>
<td>ernin</td>
<td>ernin</td>
<td>ernin</td>
<td>ernin</td>
<td>ernin</td>
</tr>
<tr>
<td>7</td>
<td>oscar</td>
<td>oscar</td>
<td>oscar</td>
<td>oscar</td>
<td>oscar</td>
<td>oscar</td>
<td>oscar</td>
<td>oscar</td>
<td>oscar</td>
<td>oscar</td>
<td>oscar</td>
<td>oscar</td>
<td>oscar</td>
</tr>
<tr>
<td>8</td>
<td>peggy</td>
<td>peggy</td>
<td>peggy</td>
<td>peggy</td>
<td>peggy</td>
<td>wendy</td>
<td>walter</td>
<td>trudy</td>
<td>peggy</td>
<td>peggy</td>
<td>peggy</td>
<td>peggy</td>
<td>peggy</td>
</tr>
<tr>
<td>9</td>
<td>trudy</td>
<td>trudy</td>
<td>trudy</td>
<td>trudy</td>
<td>trudy</td>
<td>trudy</td>
<td>trudy</td>
<td>trudy</td>
<td>trudy</td>
<td>trudy</td>
<td>trudy</td>
<td>trent</td>
<td>trent</td>
</tr>
<tr>
<td>10</td>
<td>eve</td>
<td>eve</td>
<td>eve</td>
<td>eve</td>
<td>eve</td>
<td>eve</td>
<td>eve</td>
<td>eve</td>
<td>eve</td>
<td>eve</td>
<td>wendy</td>
<td>walter</td>
<td>trent</td>
</tr>
<tr>
<td>11</td>
<td>trent</td>
<td>trent</td>
<td>trent</td>
<td>trent</td>
<td>trent</td>
<td>trent</td>
<td>trent</td>
<td>trent</td>
<td>trent</td>
<td>trent</td>
<td>trent</td>
<td>wendy</td>
<td>walter</td>
</tr>
<tr>
<td>12</td>
<td>bob</td>
<td>bob</td>
<td>bob</td>
<td>bob</td>
<td>bob</td>
<td>bob</td>
<td>bob</td>
<td>bob</td>
<td>bob</td>
<td>bob</td>
<td>wendy</td>
<td>walter</td>
<td>walter</td>
</tr>
<tr>
<td>13</td>
<td>craig</td>
<td>craig</td>
<td>craig</td>
<td>craig</td>
<td>craig</td>
<td>craig</td>
<td>craig</td>
<td>craig</td>
<td>craig</td>
<td>craig</td>
<td>craig</td>
<td>wendy</td>
<td>walter</td>
</tr>
<tr>
<td>14</td>
<td>frank</td>
<td>frank</td>
<td>frank</td>
<td>frank</td>
<td>frank</td>
<td>frank</td>
<td>frank</td>
<td>frank</td>
<td>frank</td>
<td>frank</td>
<td>frank</td>
<td>frank</td>
<td>wendy</td>
</tr>
<tr>
<td>15</td>
<td>victor</td>
<td>victor</td>
<td>victor</td>
<td>victor</td>
<td>victor</td>
<td>victor</td>
<td>victor</td>
<td>victor</td>
<td>victor</td>
<td>victor</td>
<td>victor</td>
<td>victor</td>
<td>victor</td>
</tr>
</tbody>
</table>

25
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]);
    }
}
Empirical tests of insertion sort

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

<table>
<thead>
<tr>
<th>$N$</th>
<th>$T_N$ (seconds)</th>
<th>$T_N/T_N/2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>20,000</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>40,000</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>80,000</td>
<td>35</td>
<td>9</td>
</tr>
<tr>
<td>160,000</td>
<td>225</td>
<td>6.4</td>
</tr>
<tr>
<td>320,000</td>
<td>1019</td>
<td>4.5</td>
</tr>
<tr>
<td>...</td>
<td>1.28 million</td>
<td>4</td>
</tr>
</tbody>
</table>

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

$\frac{T_N}{T_N/2} = \frac{1}{2}$

Do you have anything better?

And $4 \times 64 / 24 = 10+$ days to sort 10 million? Sounds bad.
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.

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

<table>
<thead>
<tr>
<th>computer</th>
<th>instructions per second</th>
<th>words of memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDP-9</td>
<td>tens of thousands</td>
<td>tens of thousands</td>
</tr>
<tr>
<td>VAX 11-780</td>
<td>millions</td>
<td>millions</td>
</tr>
<tr>
<td>CRAY 1</td>
<td>tens of millions</td>
<td>tens of millions</td>
</tr>
<tr>
<td>MacBook Air</td>
<td>billions</td>
<td>billions</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>order of growth</th>
<th>scales?</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>✓</td>
</tr>
<tr>
<td>$N \log N$</td>
<td>✓</td>
</tr>
<tr>
<td>$N^2$</td>
<td>✗</td>
</tr>
<tr>
<td>$N^3$</td>
<td>✗</td>
</tr>
</tbody>
</table>

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.

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.

**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.
Abstract inplace merge

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

```java
private static String[] aux;

public static void merge(String[] a, int lo, int mid, int hi)
{
    // Merge a[lo, mid) with a[mid, hi) into aux[0, hi-lo).
    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.

```java
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);
    }
    ...
}
```

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

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

Mergesort

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

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

Cost model. Count data moves.

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 \( 2^{n-2} \).
- ...\n- \( 2^n \) 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.

<table>
<thead>
<tr>
<th>$N$</th>
<th>$T_N$ (seconds)</th>
<th>$T_N/T_{N/2}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 million</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2 million</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4 million</td>
<td>5</td>
<td>2.5</td>
</tr>
<tr>
<td>8 million</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>16 million</td>
<td>20</td>
<td>2.5</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.02 billion</td>
<td>1280</td>
<td>2</td>
</tr>
</tbody>
</table>

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

WILL scale

% 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

OK! Let’s get started...
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. \( a a c a a a g t t t t a c a a a g g t t t a c a a a g c t a g c \)

Example 3 (first 100 digits of \( \pi \)).

\[ 3 . 1 4 1 5 9 2 6 5 3 5 8 9 7 9 3 2 3 8 4 \]

\[ 6 2 6 4 3 3 8 3 2 7 9 5 0 2 8 8 4 1 9 7 \]

\[ 1 6 9 3 9 9 3 7 5 1 0 5 8 2 0 9 7 4 9 4 \]

\[ 4 5 9 2 3 0 7 8 1 6 4 0 6 2 8 6 2 0 8 9 \]

\[ 9 8 6 2 8 0 3 4 8 2 5 3 4 2 1 1 7 0 6 9 \]
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.

Example 2: Cryptography
- Find LRS.
- Check for “known” message header information.
- Break code.

Example 3: DNA
- Find LRS
- Look somewhere else for causal mechanisms
- Ex. Chromosome 11 has 7.1 million nucleotides
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.

Implementation (easy)

```java
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);
}
```
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));
    }
}
LRS: An efficient solution that uses sorting

1. Form suffix strings

<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>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a a g</td>
<td>c a a g</td>
<td>t t t a c a a g c</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>
<tr>
<td>0</td>
<td>a a c</td>
<td>c a a g</td>
<td>t t t a c a a g c</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>
<tr>
<td>1</td>
<td>a c a</td>
<td>a g t</td>
<td>t t t a c a a g c</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>
<tr>
<td>2</td>
<td>c a a</td>
<td>g t t</td>
<td>t t a c a a g c</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>
<tr>
<td>3</td>
<td>a a g</td>
<td>t t t</td>
<td>a c a a g c</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>
<tr>
<td>4</td>
<td>a g t</td>
<td>t t t</td>
<td>a c a a g c</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>
<tr>
<td>5</td>
<td>g t t</td>
<td>t t a</td>
<td>c a a g c</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>
<tr>
<td>6</td>
<td>t t t</td>
<td>t a c</td>
<td>a a g c</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>
<tr>
<td>7</td>
<td>t t a</td>
<td>c a a</td>
<td>g c</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>
<tr>
<td>8</td>
<td>t a c</td>
<td>a a g</td>
<td>c</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>
<tr>
<td>9</td>
<td>a c a</td>
<td>a g c</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>
<tr>
<td>10</td>
<td>c a a</td>
<td>g c</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>
<tr>
<td>11</td>
<td>a a g</td>
<td>c</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>
<tr>
<td>12</td>
<td>a g c</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>
<tr>
<td>13</td>
<td>g c</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>
<tr>
<td>14</td>
<td>c</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>

2. Sort suffix strings

<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>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a a c</td>
<td>c a a g</td>
<td>t t t a c a a g c</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>
<tr>
<td>0</td>
<td>a a c</td>
<td>a a g</td>
<td>t t t a c a a g c</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>
<tr>
<td>1</td>
<td>a c a</td>
<td>a g t</td>
<td>t t t a c a a g c</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>
<tr>
<td>2</td>
<td>c a a</td>
<td>g t t</td>
<td>t t a c a a g c</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>
<tr>
<td>3</td>
<td>a a g</td>
<td>t t t</td>
<td>a c a a g c</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>
<tr>
<td>4</td>
<td>a g t</td>
<td>t t t</td>
<td>a c a a g c</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>
<tr>
<td>5</td>
<td>g t t</td>
<td>t t a</td>
<td>c a a g c</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>
<tr>
<td>6</td>
<td>t t t</td>
<td>t a c</td>
<td>a a g c</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>
<tr>
<td>7</td>
<td>t t a</td>
<td>c a a</td>
<td>g c</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>
<tr>
<td>8</td>
<td>t a c</td>
<td>a a g</td>
<td>c</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>
<tr>
<td>9</td>
<td>a c a</td>
<td>a g c</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>
<tr>
<td>10</td>
<td>c a a</td>
<td>g c</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>
<tr>
<td>11</td>
<td>a a g</td>
<td>c</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>
<tr>
<td>12</td>
<td>a g c</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>
<tr>
<td>13</td>
<td>g c</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>
<tr>
<td>14</td>
<td>c</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>

3. 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;
}

Model

- Alphabet: actg.
- \( N \)-character random strings.

<table>
<thead>
<tr>
<th>Doubling</th>
<th>( N )</th>
<th>( T_N )</th>
<th>( T_N/T_{N/2} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2,000,000</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4,000,000</td>
<td>7</td>
<td>2.3</td>
<td></td>
</tr>
<tr>
<td>8,000,000</td>
<td>16</td>
<td>2.3</td>
<td></td>
</tr>
<tr>
<td>16,000,000</td>
<td>39</td>
<td>2.4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x10</th>
<th>( N )</th>
<th>( T_N )</th>
<th>( T_N/T_{N/10} )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1,000,000</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10,000,000</td>
<td>21</td>
<td>10</td>
</tr>
</tbody>
</table>

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
% 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.

   ```java
   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.

Hey, Bob. Our IPO is next week!

I think I'll take a few CS courses.
Image sources

https://www.bewitched.com/match
11. Sorting and Searching