11. Sorting and Searching

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)

<table>
<thead>
<tr>
<th>whitelist</th>
<th>StdIn</th>
<th>StdOut</th>
</tr>
</thead>
<tbody>
<tr>
<td>alice@home</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>carl@beach</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>bob@office</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>marvin@spam</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>bob@office</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>mallory@spam</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>dave@boat</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>eveairport</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>alice@home</td>
<td>✓</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) {
            for (int i = 0; i < args.length; i++) {
                String[] words = in.readAllStrings();
                while (!std.in.isEmpty()) {
                    String key = std.in.readString();
                    if (search(key, words) == -1) {
                        std.out.println(key);
                    }
                }
            }
        }
    }

    public static String search(String key, String[] keys) {
        if (key == null) {
            return null;
        }
        int n = keys.length;
        for (int i = 0; i < n; i++) {
            String word = keys[i];
            if (word.equals(key)) {
                return word;
            }
        }
        return null;
    }
}
```

% more whit4.txt
alice@home
bob@office
carl@beach
dave@boat

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

% java WhiteFilter whit4.txt < test.txt
bob@office
carl@beach
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.

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

**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].equals(key)) 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.
Random representative inputs for searching and sorting

Generate N random strings of length L from a given alphabet

```java
class Generator {
    public static String randomString(int L, String alphabet) {
        char[] a = new char[L];
        for (int i = 0; i < L; i++) {
            int t = StdRandom.uniform(alphabet.length());
            a[i] = alphabet.charAt(t);
        }
        return new String(a);
    }
    public static void main(String[] args) {
        // Generate random strings...
    }
}
```

Test client for sequential search

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

```java
class TestSS {
    public static int search(String key, String[] words) {
        for (int i = 0; i < words.length; i++) {
            if (words[i].compareTo(key) == 0) return i;
        }
        return -1;
    }
    public static void main(String[] args) {
        String[] words =...;
        int length =...;
        double start =...;
        for (int i = 0; i < length; i++) {
            String key =...;
            double now =...;
            StdOut.println(key);
        }
    }
}
```

Empirical tests of sequential search

<table>
<thead>
<tr>
<th>N</th>
<th>T_a (seconds)</th>
<th>T_u/T_a2</th>
<th>Transactions per second</th>
</tr>
</thead>
<tbody>
<tr>
<td>10,000</td>
<td>3</td>
<td>3,333</td>
<td>10,000</td>
</tr>
<tr>
<td>20,000</td>
<td>9</td>
<td>2,222</td>
<td>20,000</td>
</tr>
<tr>
<td>40,000</td>
<td>35</td>
<td>1,143</td>
<td>40,000</td>
</tr>
<tr>
<td>80,000</td>
<td>149</td>
<td>536</td>
<td>80,000</td>
</tr>
<tr>
<td></td>
<td>1.28 million</td>
<td>38,500</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>= 10 a-z</td>
<td></td>
<td>more than 10.5 hours</td>
</tr>
</tbody>
</table>

The running time of our program is T_a = a N^2.

<table>
<thead>
<tr>
<th>N</th>
<th>T_a (seconds)</th>
<th>T_u/T_a2</th>
<th>Transactions per second</th>
</tr>
</thead>
<tbody>
<tr>
<td>10,000</td>
<td>3</td>
<td>3,333</td>
<td>10,000</td>
</tr>
<tr>
<td>20,000</td>
<td>9</td>
<td>2,222</td>
<td>20,000</td>
</tr>
<tr>
<td>40,000</td>
<td>35</td>
<td>1,143</td>
<td>40,000</td>
</tr>
<tr>
<td>80,000</td>
<td>149</td>
<td>536</td>
<td>80,000</td>
</tr>
<tr>
<td></td>
<td>1.28 million</td>
<td>38,500</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>= 10 a-z</td>
<td></td>
<td>more than 10.5 hours</td>
</tr>
</tbody>
</table>

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

Hmmm. That doesn't seem too good.

Validates hypothesis that order of growth is N^2. Does NOT scale.
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;
}
```

**Binary search arithmetic**

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

<table>
<thead>
<tr>
<th>Search in (a[lo,hi])</th>
<th>mid = (lo + (hi-lo)/2)</th>
<th>Lower half: (a[lo,mid])</th>
<th>Upper half: (a[mid+1,hi])</th>
</tr>
</thead>
<tbody>
<tr>
<td>(lo)</td>
<td>(lo)</td>
<td>(lo)</td>
<td>(lo)</td>
</tr>
<tr>
<td>(hi)</td>
<td>(hi)</td>
<td>(hi)</td>
<td>(hi)</td>
</tr>
<tr>
<td>(mid)</td>
<td>(mid)</td>
<td>(mid)</td>
<td>(mid)</td>
</tr>
<tr>
<td>(lo)</td>
<td>(lo)</td>
<td>(lo)</td>
<td>(lo)</td>
</tr>
<tr>
<td>(hi)</td>
<td>(hi)</td>
<td>(hi)</td>
<td>(hi)</td>
</tr>
</tbody>
</table>

**Binary search: Java implementation**

```java
public static int search(String key, String[] a) {
  return search(key, a, 0, a.length - 1);
}

public static int search(String key, String[] a, int lo, int hi) {
  if (hi <= lo) return -1;
  int mid = (lo + hi) / 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;
}
```
**Reursion 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;
}
```

**Mathematical analysis of binary search**

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

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

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

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

**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/2} )</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>
<td></td>
</tr>
<tr>
<td>200,000</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>400,000</td>
<td>6</td>
<td>2</td>
<td>3.5</td>
<td>67,000</td>
</tr>
<tr>
<td>800,000</td>
<td>14</td>
<td>2.35</td>
<td></td>
<td>57,000</td>
</tr>
<tr>
<td>1,600,000</td>
<td>33</td>
<td>2.33</td>
<td></td>
<td>48,000</td>
</tr>
<tr>
<td>10.28 million</td>
<td>264</td>
<td>2</td>
<td></td>
<td>48,000</td>
</tr>
</tbody>
</table>

Validates hypothesis that order of growth is \( N \log N \).
11. Sorting and Searching

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

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

Sorting: Rearrange N items to put them in ascending order

Insertion sort algorithm

- 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.
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>$T_N/T_{102}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>20,000</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>40,000</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>80,000</td>
<td>15</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>...</td>
<td>...</td>
</tr>
<tr>
<td>1.28 million</td>
<td>14400</td>
<td>4</td>
</tr>
</tbody>
</table>

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>

And $4x4.624 = 17$ days to sort 1 million! Sounds bad.

Do you have anything better?
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.

11. Sorting and Searching

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

**Mergesort algorithm**

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

**John von Neumann**
1903–1957

- 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
int i = lo, j = mid, k = lo;  // Copy a[lo..mid)
for (int k = lo; k < N; k++) {
  if (i < mid) aux[k] = a[i++];
  else if (j < hi) aux[k] = a[j++];
  else aux[k] = a[mid];
}
// Copy back into a[lo, hi)
for (int k = lo; k < N; k++)
a[lo + k] = aux[k];
```

Mergesort

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

```java
public class MergeSort {
    public static void mergeSort(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);
    }
    public static void merge(String[] a, int lo, int mid, int hi) {
        // Sort a[lo, hi).
        int N = hi - lo;
        if (N <= 1) return;
        // Copy a[lo, mid)
        for (int k = lo; k < N; k++)
            aux[k] = a[k];
        // Copy back into a[lo, hi)
        for (int k = lo; k < N; k++)
a[lo + k] = aux[k];
    }
    public static void sort(String[] a) {
        mergeSort(a, 0, a.length);
    }
}
```

Mergesort trace

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

**Cost model. Count data moves.**

<table>
<thead>
<tr>
<th># of times a string moves from one array to another</th>
</tr>
</thead>
</table>
| 1 subarray of size 2
| 2 subarrays of size 2^2
| 4 subarrays of size 2^3
| ... |
| 2^N subarrays of size 1 |
| Total # data moves: 2N log 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>$T_N/T_8/2$</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>1.02 billion</td>
<td>1280</td>
<td>2</td>
</tr>
</tbody>
</table>

% java Generator 1000000 ... 1 second
% 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 $N \log N$

OK! Let’s get started...

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.
```
acagtttacagc
```

Example 2.
```
acagtttacagttttacagc
cgttagc
```

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

Example. \texttt{aacaagtttaacaagc}
\texttt{aacaagtttaacaagc}

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++)
                if (s.substring(i, j + 1).equals(s.substring(j, j + 1)))
                    lrs += s.substring(i, j + 1);
        return lrs;
    }
    public static void main(String[] args) {
        String str = StdIn.readString();
        StdOut.println(lrs(str));
    }
}
```

Analysis
- \( \approx N/2 \) calls on \texttt{lcp}.
- Obviously does not scale.
LRS: Efficient solution that uses sorting

1. Form suffix strings
   - acaagttacaagc
   - acaagttacaagc
   - acaagttacaagc
   - acaagttacaagc
   - acaagttacaagc
   - acaagttacaagc
   - acaagttacaagc
   - acaagttacaagc
   - acaagttacaagc
   - acaagttacaagc
   - acaagttacaagc

2. Sort suffix strings
   - acacgttttacaagc
   - acacgttttacaagc
   - acacgttttacaagc
   - acacgttttacaagc
   - acacgttttacaagc
   - acacgttttacaagc
   - acacgttttacaagc
   - acacgttttacaagc
   - acacgttttacaagc
   - acacgttttacaagc
   - acacgttttacaagc

3. Find longest LCP among adjacent entries.


Model
- Alphabet: actg
- N-character random strings

<table>
<thead>
<tr>
<th>Doubling</th>
<th>(N)</th>
<th>(T_0)</th>
<th>(T_0/T_0/2)</th>
<th>(T_0/T_0/10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2,000,000</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4,000,000</td>
<td>7</td>
<td>2.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8,000,000</td>
<td>16</td>
<td>2.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16,000,000</td>
<td>39</td>
<td>2.4</td>
<td></td>
<td></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

<table>
<thead>
<tr>
<th>Doubling</th>
<th>(N)</th>
<th>(T_0)</th>
<th>(T_0/T_0/2)</th>
<th>(T_0/T_0/10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,000,000</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10,000,000</td>
<td>21</td>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

LRS: Suffix array implementation

```java
public static String Trs(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 Trs = "";
    for (int i = 0; i < N-1; i++)
        if (x[strlen(suffices[i])] > strlen(suffices[i+1]))
            Trs = Trs + x[i];
    return Trs;
}
```

Analysis
- \(N\) calls on substring()
- \(N\) calls on 1cp()
- Potentially scales.
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)

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.
11. Sorting and Searching