4.2 Sorting and Searching

Sequential Search: Java Implementation

Scan through array, looking for key.
• search hit: return array index
• search miss: 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;
}

Search Client: Exception Filter

A credit card company needs to whitelist 10 million customer accounts, processing 1000 transactions per second.
Using sequential search, what kind of computer is needed?
A. Toaster
B. Cellphone
C. Your laptop
D. Supercomputer
E. Google server farm
A credit card company needs to whitelist 10 million customer accounts, processing 1000 transactions per second. Using sequential search, what kind of computer is needed?

A. Toaster  
B. Cellphone  
C. Your laptop  
D. Supercomputer  
E. Google server farm  

D. or E.
• BOE rule of thumb for any computer:  
  N bytes in memory, ~N memory accesses per second.  
• sequential search touches about half the memory  
• 2 transactions per second, 500 seconds for 1000 transactions  
• fix 1: Increase memory (and speed) by factor of 1000 (supercomputer)  
• fix 2: Increase number of processors by factor of 1000 (server farm)  
• fix 3: Use a better algorithm (stay tuned)

Intuition. Find a hidden integer.

<table>
<thead>
<tr>
<th>interval</th>
<th>size</th>
<th>Q</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 128</td>
<td>128</td>
<td>&lt; 64</td>
<td>false</td>
</tr>
<tr>
<td>0 - 64</td>
<td>64</td>
<td>&lt; 96</td>
<td>true</td>
</tr>
<tr>
<td>0 - 64</td>
<td>32</td>
<td>&lt; 80</td>
<td>true</td>
</tr>
<tr>
<td>0 - 64</td>
<td>16</td>
<td>&lt; 72</td>
<td>false</td>
</tr>
<tr>
<td>0 - 64</td>
<td>8</td>
<td>&lt; 76</td>
<td>false</td>
</tr>
<tr>
<td>0 - 64</td>
<td>4</td>
<td>&lt; 78</td>
<td>true</td>
</tr>
<tr>
<td>0 - 64</td>
<td>2</td>
<td>&lt; 77</td>
<td>false</td>
</tr>
<tr>
<td>0 - 2</td>
<td>1</td>
<td>= 77</td>
<td></td>
</tr>
</tbody>
</table>

Ex. Dictionary, phone book, book index, credit card numbers, ...

Binary search.  
• Examine the middle key.  
• If it matches, return its index.  
• Otherwise, search either the left or right half.
Invariant. Algorithm maintains $a[lo] \leq key \leq a[hi-1]$.

```java
public static int search(String key, String[] a)
{
    return search(key, a, 0, a.length);
}
public static int search(String key, String[] a, int lo, int hi)
{
    if (hi <= lo) return -1;
    int mid = lo + (hi - lo) / 2;
    int cmp = a[mid].compareTo(key);
    if (cmp > 0) return search(key, a, lo, mid);
    else if (cmp < 0) return search(key, a, mid + 1, hi);
    else return mid;
}
```

Java library implementation: `Arrays.binarySearch()`

Analysis. To binary search in an array of size $N$: do one comparison, then binary search in an array of size $N/2$.

$$N \rightarrow N/2 \rightarrow N/4 \rightarrow N/8 \rightarrow \cdots \rightarrow 1$$

Q. How many times can you divide a number by 2 until you reach 1?

A. $\log_2 N$.

TEQ on Searching 2

A credit card company needs to whitelist 10 million customer accounts, processing 1 thousand transactions per second.

Using binary search, what kind of computer is needed?

A. Toaster
B. Cellphone
C. Your laptop
D. Supercomputer
E. Google server farm
Q. What’s the fastest way to sort 1 million 32-bit integers?

Insertion Sort

• Brute-force sorting solution.
• Move left-to-right through array.
• Exchange next element with larger elements to its left, one-by-one.

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>6</td>
<td>and had him his was you the but</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>and had him his was the you but</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>and had him his the was you but</td>
</tr>
</tbody>
</table>

Inserting a[6] into position by exchanging with larger entries to its left
**Insertion sort.**
- Brute-force sorting solution.
- Move left-to-right through array.
- Exchange next element with larger elements to its left, one-by-one.

**Insertion Sort**

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>was had him and you his the but</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>had was him and you his the but</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>had him was and you his the but</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>and had him was you his the but</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>and had him was you his the but</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>and had him his was you the but</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>and had him his the was you but</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>and but had him his the was you and but had him his the was you</td>
</tr>
</tbody>
</table>

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

**Insertion Sort: Empirical Analysis**

**Observation.** Number of comparisons depends on input family.
- Descending: ~ $N^2/2$.
- Ascending: ~ $N$.

**Insertion Sort: Java Implementation**

```java
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 swap = a[i];
        a[i] = a[j];
        a[j] = swap;
    }
}
```

**Insertion Sort: Mathematical Analysis**

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

**Average case.** [random]
- Iteration $i$ requires $i/2$ comparisons on average.
- Total $= (0 + 1 + 2 + ... + N-1)/2 \sim N^2/4$ compares
Hypothesis: Running time is $\sim a N^b$ seconds

Initial experiments:

<table>
<thead>
<tr>
<th>N</th>
<th>Comparisons</th>
<th>Time</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>5,000</td>
<td>6.2 million</td>
<td>0.13 sec</td>
<td></td>
</tr>
<tr>
<td>10,000</td>
<td>25 million</td>
<td>0.43 sec</td>
<td>3.3</td>
</tr>
<tr>
<td>20,000</td>
<td>99 million</td>
<td>1.5 sec</td>
<td>3.5</td>
</tr>
<tr>
<td>40,000</td>
<td>400 million</td>
<td>5.6 sec</td>
<td>3.7</td>
</tr>
<tr>
<td>80,000</td>
<td>1,600 million</td>
<td>23 sec</td>
<td>4.1</td>
</tr>
</tbody>
</table>

Doubling hypothesis:
- $b = \lg 4 = 2$, so running time is $\sim a N^2$
- checks with math analysis
- $a \approx \frac{23}{80000} \approx 3.5 \times 10^{-9}$

Refined hypothesis: Running time is $\approx 3.5 \times 10^{-9} N^2$ seconds

Refined hypothesis: Running time is $\approx 3.5 \times 10^{-9} N^2$ seconds

Prediction: Running time for $N = 200,000$
should be $3.5 \times 10^{-9} \times 4 \times 10^{10} \approx 140$ seconds

Observation:

<table>
<thead>
<tr>
<th>N</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>200,000</td>
<td>145 seconds</td>
</tr>
</tbody>
</table>

Observation matches prediction and validates refined hypothesis.

A credit card company uses insertion sort to sort 10 million customer account numbers, for use in whitelisting with binary search. What kind of computer is needed?

A. Toaster
B. Cellphone
C. Your laptop
D. Supercomputer
E. Google server farm

TEQ on Sorting 1

Lesson. Supercomputer can’t rescue a bad algorithm.

<table>
<thead>
<tr>
<th>Computer</th>
<th>Comparisons Per-Second</th>
<th>Thousand</th>
<th>Million</th>
<th>Billion</th>
</tr>
</thead>
<tbody>
<tr>
<td>laptop</td>
<td>$10^7$</td>
<td>instant</td>
<td>1 day</td>
<td>3 centuries</td>
</tr>
<tr>
<td>super</td>
<td>$10^{12}$</td>
<td>instant</td>
<td>1 second</td>
<td>2 weeks</td>
</tr>
</tbody>
</table>
Moore’s Law

Moore’s law. Transistor density on a chip doubles every 2 years.

Variants. Memory, disk space, bandwidth, computing power per $.

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.
• Divide array into two halves.
• Recursively sort each half.
• Merge two halves to make sorted whole.
Mergesort: Example

Merging Combine two pre-sorted lists into a sorted whole.

How to merge efficiently? Use an auxiliary array.

```java
String[] aux = new String[N];
// Merge into auxiliary array.
int i = lo, j = mid;
for (int k = 0; k < N; k++)
{
    if (i == mid) aux[k] = a[j++];
    else if (j == hi) aux[k] = a[i++];
    else if (a[j].compareTo(a[i]) < 0) aux[k] = a[j++];
    else aux[k] = a[i++];
}
// Copy back.
for (int k = 0; k < N; k++)
a[lo + k] = aux[k];
```

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;

        int mid = lo + N/2;
        sort(a, lo, mid);
        sort(a, mid, hi);

        // Recursively sort left and right halves.
        // 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 $N$ comparisons.

Mathematical analysis.

$$T(N) = T(N/2) + T(N/2) + N$$

Validation. Theory agrees with observations.

<table>
<thead>
<tr>
<th>$N$</th>
<th>actual</th>
<th>predicted</th>
</tr>
</thead>
<tbody>
<tr>
<td>10,000</td>
<td>120 thousand</td>
<td>133 thousand</td>
</tr>
<tr>
<td>20 million</td>
<td>460 million</td>
<td>485 million</td>
</tr>
<tr>
<td>50 million</td>
<td>1,216 million</td>
<td>1,279 million</td>
</tr>
</tbody>
</table>

Mergesort: Scientific Analysis

Hypothesis. Running time is a $N \log N$ seconds.

Initial experiments:

<table>
<thead>
<tr>
<th>$N$</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 million</td>
<td>3.13 sec</td>
</tr>
<tr>
<td>4 million</td>
<td>3.25 sec</td>
</tr>
<tr>
<td>4 million</td>
<td>3.22 sec</td>
</tr>
</tbody>
</table>

Refined hypothesis. Running time is $2.5 \times 10^{-7} N \log N$ seconds.

Prediction: Running time for $N = 20,000,000$

should be about $2.5 \times 10^{-8} \times 2 \times 10^{7} \times 35 \approx 17.5$ seconds

Observation:

<table>
<thead>
<tr>
<th>$N$</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 million</td>
<td>17.5 sec</td>
</tr>
</tbody>
</table>

Observation matches prediction and validates refined hypothesis.

TEQ on Sorting 2

A credit card company uses mergesort to sort 10 million customer account numbers, for use in whitelisting with binary search. What kind of computer is needed?

A. Toaster
B. Cellphone
C. Your laptop
D. Supercomputer
E. Google server farm
Mergesort: Lesson

Lesson. Great algorithms can be more powerful than supercomputers.

<table>
<thead>
<tr>
<th>Computer</th>
<th>Comparisons Per Second</th>
<th>Insertion</th>
<th>Mergesort</th>
</tr>
</thead>
<tbody>
<tr>
<td>laptop</td>
<td>$10^7$</td>
<td>3 centuries</td>
<td>3 hours</td>
</tr>
<tr>
<td>super</td>
<td>$10^{12}$</td>
<td>2 weeks</td>
<td>instant</td>
</tr>
</tbody>
</table>

$N = 1$ billion

Longest Repeated Substring

Longest repeated substring. Given a string, find the longest substring that appears at least twice.

Brute force.
- Try all indices $i$ and $j$ for start of possible match.
- Compute longest common prefix for each pair (quadratic+).

Applications. Bioinformatics, cryptography, ...

Redundancy Detector

Music is characterized by its repetitive structure

Mary Had a Little Lamb

Fur Elise

Source: http://www.bewitched.com/match/
LRS applications: patterns in sequences

Repeated sequences in real-world data are causal.

Ex. 1. Digits of pi
• Q. are they “random”?
• A. No, but we can’t tell the difference
• Ex. Length of LRS in first 10 million digits is 14

Ex. 2. Cryptography
• Find LRS
• Check for “known” message header identifying place, date, person, etc.
• Break code

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

Longest Repeated Substring: A Sorting Solution

1. Form suffixes
2. Sort suffixes to bring repeated substrings together
3. Compute longest prefix between adjacent suffixes

Suffix sorting implementation.

```java
int N = s.length();
String[] suffixes = new String[N];
for (int i = 0; i < N; i++)
    suffixes[i] = s.substring(i, N);
Arrays.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;
}
```

Longest Repeated Substring: Java Implementation

Longest common prefix: \( lcp(s, t) \).
• longest string that is a prefix of both \( s \) and \( t \)
• Ex: \( lcp(\text{“aacaagtttacaagc”}, \text{“acaag”}) = \text{“acaag”} \).
• easy to implement (you could write this one).

Longest repeated substring. Search only adjacent suffixes.

Brute-force solution

Longest repeated substring. Given a string, find the longest substring that appears at least twice.

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

Brute force.
• Try all indices \( i \) and \( j \) for start of possible match.
• Compute longest common prefix (LCP) for each pair

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

Analysis.
• all pairs: \( 1 + 2 + ... + N \sim N^2/2 \) calls on LCP
• too slow for long strings
Java substring operation

Memory representation of strings.

- A String is an address and a length.
- Characters can be shared among strings.
- substring() computes address, length (instead of copying chars).

```java
s = "aacaagtttacaagc";
t = s.substring(5, 15);
```

Consequences.

- substring() is a constant-time operation (instead of linear).
- Creating suffixes takes linear space (instead of quadratic).
- Running time of LRS is dominated by the string sort.

Longest Repeated Substring: Empirical Analysis

<table>
<thead>
<tr>
<th>Input File</th>
<th>Characters</th>
<th>Brute</th>
<th>Suffix Sort</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>LRS.java</td>
<td>2,162</td>
<td>0.6 sec</td>
<td>0.14 sec</td>
<td>73</td>
</tr>
<tr>
<td>Amendments</td>
<td>18,369</td>
<td>37 sec</td>
<td>0.25 sec</td>
<td>216</td>
</tr>
<tr>
<td>Aesop's Fables</td>
<td>191,945</td>
<td>3958 sec</td>
<td>1.0 sec</td>
<td>58</td>
</tr>
<tr>
<td>Moby Dick</td>
<td>1.2 million</td>
<td>43 hours†</td>
<td>7.6 sec</td>
<td>79</td>
</tr>
<tr>
<td>Bible</td>
<td>4.0 million</td>
<td>20 days†</td>
<td>34 sec</td>
<td>11</td>
</tr>
<tr>
<td>Chromosome 11</td>
<td>7.1 million</td>
<td>2 months†</td>
<td>61 sec</td>
<td>12,567</td>
</tr>
<tr>
<td>Pi</td>
<td>10 million</td>
<td>4 months†</td>
<td>84 sec</td>
<td>14</td>
</tr>
</tbody>
</table>

† estimated

Lesson. Sorting to the rescue; enables new research.

Many, many, many other things enabled by fast sort and search!

Q. Four researchers A, B, C and D are looking for long repeated subsequences in a genome with over 1 billion characters.

- A. has a grad student do it.
- B. uses brute force (check all pairs) solution.
- C. uses sorting solution with insertion sort.
- D. uses sorting solution with mergesort.

Which one is more likely to find a cancer cure?

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.