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

Exception filter. Read a list of strings from a whitelist file, then print out all strings from standard input not in the whitelist.

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

Search Challenge 1

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
Search Challenge 1

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:
  
  \[ N \text{ bytes in memory, } \sim N \text{ memory accesses per second.} \]

• 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)

need enough memory for 10M accounts

Binary Search

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 - 32</td>
<td>32</td>
<td>&lt; 96?</td>
<td>true</td>
</tr>
<tr>
<td>0 - 16</td>
<td>16</td>
<td>&lt; 72?</td>
<td>false</td>
</tr>
<tr>
<td>0 - 8</td>
<td>8</td>
<td>&lt; 76?</td>
<td>false</td>
</tr>
<tr>
<td>0 - 4</td>
<td>4</td>
<td>&lt; 78?</td>
<td>true</td>
</tr>
<tr>
<td>0 - 2</td>
<td>2</td>
<td>&lt; 77?</td>
<td>false</td>
</tr>
<tr>
<td>0 - 1</td>
<td>1</td>
<td>= 77</td>
<td>true</td>
</tr>
</tbody>
</table>

Twenty Questions

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

Idea:
• Sort the array (stay tuned)
• Play "20 questions" to determine the index associated with a given key.

Binary search.
• Examine the middle key.
• If it matches, return its index.
• Otherwise, search either the left or right half.

Binary search in an array (one step)
**Binary Search: Java Implementation**

**Invariant.** Algorithm maintains $a[lo] \leq \text{key} \leq a[hi-1]$.

```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()`

**Search Challenge 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

**Binary Search: Mathematical Analysis**

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

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

Q. How many times can you divide a number by 2 until you reach 1?  
A. $\log_2 N$.

```
1  
2 \rightarrow 1  
4 \rightarrow 2 \rightarrow 1  
8 \rightarrow 4 \rightarrow 2 \rightarrow 1  
16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1  
32 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1  
64 \rightarrow 32 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1  
128 \rightarrow 64 \rightarrow 32 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1  
256 \rightarrow 128 \rightarrow 64 \rightarrow 32 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1  
512 \rightarrow 256 \rightarrow 128 \rightarrow 64 \rightarrow 32 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1  
1024 \rightarrow 512 \rightarrow 256 \rightarrow 128 \rightarrow 64 \rightarrow 32 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1
```

• back-of-envelope rule of thumb for any computer:  
  $M$ bytes in memory, $\sim M$ memory accesses per second.  
  $\lg M$ accesses per transaction  
  $M/\lg M$ transactions per second  
  $(1000 \lg M / M)$ seconds for $1000$ transactions  
  Ex: $M = 128$ MB, $\lg M \sim 27$: $0.0002$ seconds for $1000$ transactions
Insertion Sort

Insertion Sort

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

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>and</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>had</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>him</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>his</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.

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</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</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: \( \sim \frac{N^2}{2} \).
• Random: \( \sim \frac{N^2}{4} \).
• Ascending: \( \sim N \).

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

Insertion Sort: Mathematical Analysis

Worst case. [descending]
• Iteration \( i \) requires \( i \) comparisons.
• Total \( \sum (0 + 1 + 2 + \ldots + N-1) = \frac{N^2}{2} \) compares.

Average case. [random]
• Iteration \( i \) requires \( \frac{i}{2} \) comparisons on average.
• Total \( \sum (0 + 1 + 2 + \ldots + N-1)/2 = \frac{N^2}{4} \) compares
Insertion Sort: Scientific Analysis

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 s</td>
<td></td>
</tr>
<tr>
<td>10,000</td>
<td>25 million</td>
<td>0.43 s</td>
<td>3.3</td>
</tr>
<tr>
<td>20,000</td>
<td>99 million</td>
<td>1.5 s</td>
<td>3.5</td>
</tr>
<tr>
<td>40,000</td>
<td>400 million</td>
<td>5.6 s</td>
<td>3.7</td>
</tr>
<tr>
<td>80,000</td>
<td>1600 million</td>
<td>23 s</td>
<td>4.1</td>
</tr>
</tbody>
</table>

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

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 s</td>
</tr>
</tbody>
</table>

Observation matches prediction and validates refined hypothesis.

Sort Challenge 1

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

D. or E.
- on your laptop: Running time for \( N = 10^7 \) should be \( 3.5 \times 10^{-9} \times 10^{14} = 350000 \) seconds \( \approx 4 \) days
- fix 1: supercomputer (easy, but expensive)
- fix 2: parallel sort on server farm (also expensive, and more challenging)
- fix 3: Use a better algorithm (stay tuned)
**Insertion Sort: Lesson**

**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

```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];
```

Merging

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

How to merge efficiently? Use an auxiliary array.

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>k</th>
<th>aux[k]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>0</td>
<td>and had him was but his the you</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>1</td>
<td>but and had him was but his the you</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>2</td>
<td>had and had him was but his the you</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>3</td>
<td>his and had him was but his the you</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>4</td>
<td>his and had him was but his the you</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>5</td>
<td>the and had him was but his the you</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>6</td>
<td>was and had him was but his the you</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>7</td>
<td>you and had him was but his the you</td>
</tr>
</tbody>
</table>
public class Merge
{
    public static void sort(String[] a)
    {
        sort(a, 0, a.length);
    }
    // Sort a[lo, hi).
    public static void sort(String[] a, int lo, int hi)
    {
        int N = hi - lo;
        if (N <= 1) return;
        // Recursively sort left and right halves.
        int mid = lo + N/2;
        sort(a, lo, mid);
        sort(a, mid, hi);
        // Merge sorted halves (see previous slide).
    }
}

Mergesort:  Mathematical Analysis

Mathematical analysis.

<table>
<thead>
<tr>
<th>analysis</th>
<th>comparisons</th>
</tr>
</thead>
<tbody>
<tr>
<td>worst</td>
<td>N log₂ N</td>
</tr>
<tr>
<td>average</td>
<td>N log₂ N</td>
</tr>
<tr>
<td>best</td>
<td>1/2 N log₂ N</td>
</tr>
</tbody>
</table>

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 lg N seconds

Initial experiments:

\[ a \approx \frac{3.2}{(4 \times 10^6 \times 32)} = 2.5 \times 10^{-8} \]

Refined hypothesis. Running time is \( 2.5 \times 10^{-7} \) N lg 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.
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

ANY of the above (!) (well, maybe not the toaster).

- cellphone: less than a minute
- laptop: several seconds

Longest Repeated Substring

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. 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, ...

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

Music is characterized by its repetitive structure

Brute-force solution

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 (LCP) for each pair

Analysis.
• all pairs: $1 + 2 + \ldots + N \sim N^2/2$ calls on LCP
• too slow for long strings
Longest Repeated Substring: A Sorting Solution

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

Java substring operation

Memory representation of strings.
```
s = "aacaagtttacaagc";
```

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

```
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: Java Implementation

Suffix sorting implementation.
```
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);
```

Longest common prefix: lcp(s, t).
• longest string that is a prefix of both s and t
• Ex: lcp("aacaagtttac", "aacaagc") = "aacaagc"
• easy to implement (you could write this one).

Longest repeated substring. Search only adjacent 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;
}
```

Sort Challenge 3

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?
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?

A. NO, need to be able to program to do science nowadays
B. C. NO, not in this century!
D. Fast sort enables progress

Note: LINEAR-time algorithm for LRS is known (see COS 226)

<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!

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.