CS 112 Introduction to Programming  
(Spring 2012)  
Lecture #25: Sorting and Searching  
Zhong Shao  
Department of Computer Science  
Yale University  
Office: 314 Watson  
http://flint.cs.yale.edu/cs112  

Acknowledgements: some slides used in this class are taken directly or adapted from those accompanying the textbook: Introduction to Programming in Java: An Interdisciplinary Approach by Robert Sedgewick and Kevin Wayne (Copyright 2002-2010).

Sequential Search

Sequential search. Scan through array, looking for key.  
• Search hit: return array index.  
• Search miss: return -1.

```java
public static int search(String key, String[] a) {  
    int N = a.length;  
    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 sorted list of strings from a whitelist file, then print out all strings from standard input not in the whitelist.

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

Searching Challenge 1

Q. A credit card company needs to whitelist 10 million customer account numbers, processing 10,000 transactions per second.  

Using sequential search, what kind of computer is needed?

A. Toaster.  
B. Cell phone.  
C. Your laptop.  
D. Supercomputer.  
E. Google server farm.
Binary Search

Main idea.
- Sort the array (stay tuned).
- Play "20 questions" to determine index with a given key.

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.

Ex.

<table>
<thead>
<tr>
<th>interval</th>
<th>size</th>
<th>Q</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>64</td>
<td>&lt;</td>
<td>false</td>
</tr>
<tr>
<td>64</td>
<td>32</td>
<td>&lt;</td>
<td>true</td>
</tr>
<tr>
<td>32</td>
<td>16</td>
<td>&lt;</td>
<td>false</td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>&lt;</td>
<td>false</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>&lt;</td>
<td>true</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>&lt;</td>
<td>false</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>=</td>
<td>true</td>
</tr>
</tbody>
</table>

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 - 1);
}

public static int search(String key, String[] a, int lo, int hi) {
    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()`
Binary Search: Mathematical Analysis

**Analysis.** To binary search in an array of size $N$: do one compare, 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.** $\log_2 N$.

<table>
<thead>
<tr>
<th>Number</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td>32</td>
<td>16</td>
</tr>
<tr>
<td>64</td>
<td>8</td>
</tr>
<tr>
<td>128</td>
<td>4</td>
</tr>
<tr>
<td>256</td>
<td>2</td>
</tr>
<tr>
<td>512</td>
<td>1</td>
</tr>
<tr>
<td>1024</td>
<td>1</td>
</tr>
<tr>
<td>2048</td>
<td>2</td>
</tr>
<tr>
<td>4096</td>
<td>4</td>
</tr>
<tr>
<td>8192</td>
<td>8</td>
</tr>
<tr>
<td>16384</td>
<td>16</td>
</tr>
<tr>
<td>32768</td>
<td>32</td>
</tr>
<tr>
<td>65536</td>
<td>64</td>
</tr>
<tr>
<td>131072</td>
<td>128</td>
</tr>
<tr>
<td>262144</td>
<td>256</td>
</tr>
</tbody>
</table>

Searching Challenge 2

**Q.** A credit card company needs to whitelist 10 million customer account numbers, processing 10,000 transactions per second. Using binary search, what kind of computer is needed?

**A.** Toaster.

**B.** Cell phone.

**C.** Your laptop.

**D.** Supercomputer.

**E.** Google server farm.

Sorting

**Sorting problem.** Rearrange $N$ items in ascending order.

**Applications.** Statistics, databases, data compression, bioinformatics, computer graphics, scientific computing, (too numerous to list), ...

<table>
<thead>
<tr>
<th>Hanley</th>
<th>Hanley</th>
</tr>
</thead>
<tbody>
<tr>
<td>Henson</td>
<td>Henson</td>
</tr>
<tr>
<td>Hong</td>
<td>Hong</td>
</tr>
<tr>
<td>Hayes</td>
<td>Hayes</td>
</tr>
<tr>
<td>Haskell</td>
<td>Haskell</td>
</tr>
<tr>
<td>Hanley</td>
<td>Hanley</td>
</tr>
<tr>
<td>Henson</td>
<td>Henson</td>
</tr>
</tbody>
</table>
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>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
</tr>
</tbody>
</table>

Inserting a[6] into position by exchanging with larger entries to its left

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++) {
      if (a[i-1].compareTo(a[i]) > 0) {
        exch(a, i-1, i);
      } 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: Empirical Analysis

**Observation.** Number of compares depends on input family.

- Descending: \( -N^2/2 \)
- Random: \( -N^2/4 \)
- Ascending: \( -N \)

![Graph showing comparisons for different input sizes and orderings.

Insertion Sort: Mathematical Analysis

**Worst case.** [descending]

- Iteration \( i \) requires \( i \) comparisons.
- Total = \( (0 + 1 + 2 + \ldots + N-1) \sim N^2/2 \) compares.

**Average case.** [random]

- Iteration \( i \) requires \( i/2 \) comparisons on average.
- Total = \( (0 + 1 + 2 + \ldots + N-1)/2 \sim N^2/4 \) compares.

Sorting Challenge 1

**Q.** A credit card company sorts 10 million customer account numbers, for use with binary search.

Using **insertion sort**, what kind of computer is needed?

**A.** Toaster  
**B.** Cell phone  
**C.** Your laptop  
**D.** Supercomputer  
**E.** Google server farm

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>Supercomputer</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!

*Lesson.* Need linear (or linearithmic) algorithm to keep pace with Moore’s law.

“Software inefficiency can always outpace Moore’s Law. Moore’s Law isn’t a match for our bad coding.” – Jaron Lanier

Mergesort

Mergesort algorithm.

- 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.

String[] aux = new String[N];

// merge into auxiliary array
int k = lo, j = mid;
for (int k = 0; k < M; 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 < M; k++) {
    a[lo + k] = aux[k];
}

Mergesort: Java Implementation

public class Merge {
    public static void sort(String[] a) {
        sort(a, 0, a.length);
    }
    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);
        // merge sorted halves (see previous slide)
    }
}

Trace of the merge of the sorted left half with the sorted right half
Mergesort: Mathematical Analysis

Analysis. To mergesort array of size $N$, mergesort two subarrays of size $N/2$, and merge them together using $\leq N$ compares. We assume $N$ is a power of 2.

Mathematical analysis.

<table>
<thead>
<tr>
<th></th>
<th>analysis</th>
<th>comparisons</th>
</tr>
</thead>
<tbody>
<tr>
<td>worst</td>
<td>$N \log_2 N$</td>
<td></td>
</tr>
<tr>
<td>average</td>
<td>$N \log_2 N$</td>
<td></td>
</tr>
<tr>
<td>best</td>
<td>$\frac{1}{2} N \log_2 N$</td>
<td></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>

Sorting Challenge 2

Q. A credit card company sorts 10 million customer account numbers, for use with binary search. Using mergesort, what kind of computer is needed?

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

Sorting Challenge 3

Q. What’s the fastest way to sort 1 million 32-bit integers?
Mergesort: Lesson

Lesson. Great algorithms can be more powerful than supercomputers.

<table>
<thead>
<tr>
<th>Computer</th>
<th>Compared 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, data compression, ...

Redundancy Detector

LRS Application: The Shape of a Song

Music is characterized by its repetitive structure.

Mary Had a Little Lamb

Like a Prayer

http://www.bewitched.com
Longest Repeated Substring: 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.

Mathematical analysis.
- All pairs: \(0 + 1 + 2 + \cdots + N-1 \sim \frac{N^2}{2}\) calls on LCP.
- Way too slow for long strings.

Longest Repeated Substring: A Sorting Solution

Sort suffixes to bring repeated substrings together

form suffixes

sort suffixes to bring repeated substrings together

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(“acaagtttacaagc”, “acaagc”) = “acaagc”.
  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;
```

OOP Context for Strings

String representation.
- A String is an address and a length.
- Characters can be shared among strings.
- substring() computes address and length.

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

Consequences.
- substring() is constant-time operation (instead of linear).
- Creating suffixes takes linear space (instead of quadratic).
- Running time of LRS is dominated by the string sort.
Sorting Challenge 4

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

Which one is more likely to find a cure for cancer?

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

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>12 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 II</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.

Summary

Binary search. Efficient algorithm to search a sorted array.

Mergesort. Efficient algorithm to sort an array.

Applications. Many many applications are enabled by fast sorting and searching.

Extra Slides
Searching a Sorted Array

Searching a sorted array. Given a sorted array, determine the index associated with a given key.

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.

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

Java library implementation: Arrays.binarySearch()
### Insertion Sort: Prediction and Verification

**Experimental hypothesis.** # comparisons ~ \(N^2/4\).

**Prediction.** 400 million comparisons for \(N = 40,000\).

**Observations.**

<table>
<thead>
<tr>
<th>(N)</th>
<th>Comparisons</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>40,000</td>
<td>401.3 million</td>
<td>5.595 sec</td>
</tr>
<tr>
<td>40,000</td>
<td>399.7 million</td>
<td>5.573 sec</td>
</tr>
<tr>
<td>40,000</td>
<td>401.6 million</td>
<td>5.648 sec</td>
</tr>
<tr>
<td>40,000</td>
<td>400.0 million</td>
<td>5.632 sec</td>
</tr>
</tbody>
</table>

Agrees.

**Prediction.** 10 billion comparisons for \(N = 200,000\).

**Observation.**

<table>
<thead>
<tr>
<th>(N)</th>
<th>Comparisons</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>200,000</td>
<td>9,997 billion</td>
<td>145 seconds</td>
</tr>
</tbody>
</table>

Agrees.

### Insertion Sort: Mathematical Analysis

**Mathematical analysis.**

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Comparisons</th>
<th>Stddev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Worst</td>
<td>(N^2/2)</td>
<td>-</td>
</tr>
<tr>
<td>Average</td>
<td>(N^2/4)</td>
<td>1.66 (N^2)</td>
</tr>
<tr>
<td>Best</td>
<td>(N)</td>
<td>-</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>40,000</td>
<td>401.3 million</td>
<td>400 million</td>
</tr>
<tr>
<td>200,000</td>
<td>9,9997 billion</td>
<td>10,000 billion</td>
</tr>
</tbody>
</table>

### Mergesort: Preliminary Hypothesis

**Experimental hypothesis.** Number of comparisons ~ 20\(N\).

### Mergesort: Prediction and Verification

**Experimental hypothesis.** Number of comparisons ~ 20\(N\).

**Prediction.** 80 million comparisons for \(N = 4\) million.

**Observations.**

<table>
<thead>
<tr>
<th>(N)</th>
<th>Comparisons</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 million</td>
<td>82.7 million</td>
<td>3.13 sec</td>
</tr>
<tr>
<td>4 million</td>
<td>82.7 million</td>
<td>3.25 sec</td>
</tr>
<tr>
<td>4 million</td>
<td>82.7 million</td>
<td>3.22 sec</td>
</tr>
</tbody>
</table>

Agrees.

**Prediction.** 400 million comparisons for \(N = 20\) million.

**Observations.**

<table>
<thead>
<tr>
<th>(N)</th>
<th>Comparisons</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 million</td>
<td>460 million</td>
<td>17.9 sec</td>
</tr>
<tr>
<td>50 million</td>
<td>1216 million</td>
<td>45.9 sec</td>
</tr>
</tbody>
</table>

Not quite.