4.2 Sorting and Searching

Binary Search

### Twenty Questions

**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</td>
<td>128</td>
<td></td>
<td>false</td>
</tr>
<tr>
<td>0</td>
<td>64</td>
<td></td>
<td>128</td>
</tr>
<tr>
<td>0</td>
<td>64</td>
<td></td>
<td>64</td>
</tr>
<tr>
<td>0</td>
<td>64</td>
<td></td>
<td>96</td>
</tr>
<tr>
<td>1</td>
<td>64</td>
<td></td>
<td>80</td>
</tr>
<tr>
<td>1</td>
<td>64</td>
<td></td>
<td>72</td>
</tr>
<tr>
<td>1</td>
<td>72</td>
<td></td>
<td>80</td>
</tr>
<tr>
<td>1</td>
<td>72</td>
<td></td>
<td>90</td>
</tr>
<tr>
<td>0</td>
<td>72</td>
<td></td>
<td>78</td>
</tr>
<tr>
<td>0</td>
<td>72</td>
<td></td>
<td>77</td>
</tr>
<tr>
<td>1</td>
<td>72</td>
<td></td>
<td>77</td>
</tr>
<tr>
<td>1</td>
<td>77</td>
<td></td>
<td>77</td>
</tr>
</tbody>
</table>

### 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.
**Binary Search: Java Implementation**

**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) {
    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()

**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 $N/2$.

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

**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
```

**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 s = in.readAll();
    String[] words = s.split("\s+\n");
    while (!StdIn.isEmpty()) {
        String key = StdIn.readString();
        if (search(key, words) == -1) StdOut.println(key);
    }
}
```

**Sorting**

more test.txt
boboffice
carl1014
marvinsob
boboffice
marvinsob
davepope
evepope
alicepope

% more whitelist.txt
alicepope
boboffice
carl1014
marvinsob
boboffice
marvinsob
davepope
evepope

% java BinarySearch whitelist.txt = test.txt
carl1014
marvinsob
boboffice
marvinsob
boboffice
davepope
evepope
Case Study: Sorting

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

**Applications.** Statistics, databases, data compression, bioinformatics, computer graphics, scientific computing, ...

---

**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>1</td>
<td>had</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>was</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>had</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>had</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>and</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>and</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>had</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>and</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>1</td>
<td>0</td>
<td>had</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>had</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>and</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>and</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>and</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>and</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>and</td>
</tr>
</tbody>
</table>

Inserting $a[1]$ through $a[N-1]$ into position (insertion sort)
**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: Empirical Analysis**

Data analysis. Plot # comparisons vs. input size on log-log scale.

Hypothesis. # comparisons grows quadratically with input size ~ $N^2/4$.

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

**Insertion Sort: Mathematical Analysis**

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

Average case. [random]
- Iteration $i$ requires $i/2$ comparisons on average.
- Total = $(0 + 1 + 2 + ... + N-1) / 2$ ~ $N^2/4$ compares.
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 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.

Moore’s Law

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

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

Mergesort

First Draft of a Report on the EDVAC

John von Neumann
Mergesort

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

How to merge efficiently? Use an auxiliary array.

```
// 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;
        // recursively sort left and right halves
        int mid = lo + N/2;
        sort(a, lo, mid);
        sort(a, mid, hi);
        // merge (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 \( \approx N \) comparisons.

we assume \( N \) is a power of 2

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: Lesson

Lesson. Great algorithms can be more powerful than supercomputers.
Longest Repeated Substring: A Sorting Solution

**Sort suffixes to bring repeated substrings together**

- Form suffixes
- Compute longest prefix between adjacent suffixes

**Applications:** Bioinformatics, Burrows-Wheeler transform, ...

---

Redundancy Detector

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

```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);
}
```

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

```java
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.** Search only adjacent suffixes.

```java
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;
}
```

**LCP.** Find the longest string that is a prefix of both s and t.

Ex. lcp("aacaagttaac", "aacaagc") = "aacaag".
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.

Summary

**Binary search.** Efficient algorithm to search a sorted array.

**Mergesort.** Efficient algorithm to sort an array.

**Applications.** Many many applications involve sorting and searching.

---

Possible memory representation of a string.

- \( s = "\text{aacaagttaaagc}"; \)

```
D0  D1  D2  D3  D4  D5  D6  D7  D8  D9  DA  DB  DC  DD  DE
a  a  c  a  a  g  t  t  t  a  c  a  a  g  c
```

- \( t = s . \text{substring}(5, 15); \)

```
B0  B1
D5  10
```

- No characters are copied when you invoke `substring()` method!

**Consequences.**

- Calling `substring()` takes constant time (instead of linear).
- Creating suffixes takes linear space (instead of quadratic).