In the previous episodes

- O-notation

- **Data structure implementations:**
  - Lists, Queues, Heaps, Trees, Maps, Graphs

- **Algorithms:**
  - Sorting, Searching, Graph Algorithms
Algorithmic Techniques: Review

A typical algorithmic problem goes as follows:

**SORTING**
Given a list, produce a list, which has the same elements in sorted order.

**MST**
Given a graph, produce a tree, which is spanning and has minimal weight.

**INPUT**

**OUTPUT**

That satisfies certain CONDITIONS
Suppose you have an algorithmic problem at hand. What are the ways of approaching it?
Generic algorithmic techniques

- Exhaustive search
Generic algorithmic techniques

- Exhaustive search

```
ABDC
ACDB
BACD
BADC
BCDA
BDCA
ACBD
ADBC
CDAB
CABD
DABC
CADB
CBAD

ABCD
```

Algorithmics 14.04.2011
Generic algorithmic techniques

- Iterative improvement (directed search)
  - Start with a random state
  - If it is not optimal:
    - (e.g. there are two nearby positions in wrong order)
    - Improve
  - Repeat until solution optimal
Generic algorithmic techniques

- Iterative improvement (directed search)
  - Start with a random state
  - If it is not optimal:
    - (e.g. there are two nearby positions in wrong order)
    - Improve
  - Repeat until solution optimal

Bubble sort
Generic algorithmic techniques

- **Greedy algorithm**
  - Find the letter with the smallest value and put it to beginning
  - Repeat recursively
Generic algorithmic techniques

- **Greedy algorithm**
  - Find the letter with the smallest value and put it to beginning
  - Repeat recursively
Generic algorithmic techniques

- Reducing to simpler tasks
  - “Divide and conquer”

- Single recurrence (reducing \( f(n) \) to \( f(n-1) \))

- Multiple recurrence (reducing \( f(n) \) to \([f(n-1), f(n-2), \ldots]\))
Generic algorithmic techniques

- Reducing to simpler tasks
  - “Divide and conquer”
Generic algorithmic techniques

- Reducing to simpler tasks
  - “Divide and conquer”

BDCA → BD → BDCA → ABCD

Merge sort
Generic algorithmic techniques

- Reducing to simpler tasks
  - “Divide and conquer”

BDCA → BD → BD → ABCD

BDCA → BA → AB → ABCD

BDCA → CA → AC → ABCD
Generic algorithmic techniques

- Reducing to simpler tasks
  - “Divide and conquer”

```
BDCA ➔ BD ➔ BD ➔ ABCD

BDCA ➔ BA ➔ AB ➔ ABCD
```

Quicksort
Generic algorithmic techniques

- Reducing to simpler tasks
  - “Divide and conquer”

- Single recurrence (reducing $f(n)$ to $f(n-1)$)

  BDCA ➔ B  DCA ➔ B  ACD ➔ ABCD
Generic algorithmic techniques

- Reducing to simpler tasks
  - “Divide and conquer”

- Single recurrence (reducing $f(n)$ to $f(n-1)$)

BDCA → B → DCA → B → ACD → ABCD

[Slide on Insertion sort]
Generic algorithmic techniques

- Reducing to simpler tasks
  - “Divide and conquer”

- Single recurrence (reducing $f(n)$ to $f(n-1)$)

- Multiple recurrence (reducing $f(n)$ to $[f(n-1), f(n-2), \ldots]$)

Diagram:

- BDCA → BDC
- BDC → BCD
- BCD → ACD
- ACD → ABCD

(bad) sort
Generic algorithmic techniques

- Exhaustive search
- Iterative improvement (directed search)
- Greedy algorithm
- Reducing to simpler tasks
  - Divide and conquer
  - Single recurrence
  - Multiple recurrence (Dynamic programming)
Generic algorithmic techniques

- Exhaustive search
- Iterative improvement
- Greedy algorithm
- Reducing to simpler tasks
  - Divide and conquer
  - Single recurrence
  - Multiple recurrence (Dynamic programming)
Generic algorithmic techniques

- Complementary to the abovementioned techniques, there are helpful “tricks”:
  - Time / CPU trade-off (parallel/distributed computation)
  - Time / space trade-off (precomputation, recomputation)
  - Time / quality trade-off (approximation, heuristics)
  - Randomization
Worst-case scenarios

- If you are unlucky, then the complexity of
  - Quicksort is …
  - Hashmap is …
Worst-case scenarios

- If you are unlucky, then the complexity of
  - Quicksort is $O(n^2)$

- Hashmap is $O(n^2)$ for $n$ insertions

For $n = 10000$, this makes a 10 000 difference!

1 second becomes 3 hours!

- But why would you get unlucky?
  - Can’t we assume that inputs are random?
Inputs are NOT random!

- Example:
  - Bro is an IDS/packet filter
  - Bro sniffs IP packets and stores them in a hash map.

Example:

- Bro is an IDS/packet filter
- Bro collects packages and stores them in a hash map.
- It is easy to generate many packets that will hash to the same bucket.
- $O(n^2)$ performance!

<table>
<thead>
<tr>
<th>Packet rate</th>
<th>Packets sent</th>
<th>Drop rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>16kb/s</td>
<td>192k</td>
<td>31%</td>
</tr>
<tr>
<td>16kb/s (clever)</td>
<td>128k</td>
<td>71%</td>
</tr>
<tr>
<td>64kb/s</td>
<td>320k</td>
<td>75%</td>
</tr>
<tr>
<td>160kb/s</td>
<td>320k</td>
<td>78%</td>
</tr>
</tbody>
</table>

Table 2: Total CPU time and CPU time spent in hash table code during an offline processing run of 64k attack and 64k random SYN packets.

Table 3: Overall drop rates for the different attack scenarios.

Inputs are malicious

Example:

<table>
<thead>
<tr>
<th>File version</th>
<th>Perl 5.6.1 program</th>
<th>Perl 5.8.0 program</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perl 5.6.1</td>
<td>6506 seconds</td>
<td>&lt;2 seconds</td>
</tr>
<tr>
<td>Perl 5.8.0</td>
<td>&lt;2 seconds</td>
<td>6838 seconds</td>
</tr>
</tbody>
</table>

Table 1: CPU time inserting 90k short attack strings into two versions of Perl.
Avoiding the worst case

- The algorithm needs to have a “secret key” – something the adversary does not know about!

- If we do not have any information at all about this key, we call it “randomness”.

Algorithmics 14.04.2011
Avoiding the worst case

Instead of considering a single algorithm, consider a **family of algorithms**.

- $\text{HashMap}_1$
- $\text{HashMap}_2$
- $\text{HashMap}_3$
- $\text{HashMap}_4$
- $\ldots$
- $\text{HashMap}_{2^{32}}$
Avoiding the worst case

Instead of considering a single algorithm, consider a family of algorithms.

Each algorithm has its weak points:

- **HashMap₁**: Hard inputs: 43,21,1
- **HashMap₂**: Hard inputs: 10,15,1
- **HashMap₃**: Hard inputs: 22,33,6
- **HashMap₄**: Hard inputs: 26,43,2
- **HashMap₂³²**: Hard inputs: 39,2,74
Avoiding the worst case

Instead of considering a single algorithm, consider a family of algorithms.

But we’ll pick one at random, and the adversary won’t know which one!
Avoiding the worst case

Instead of considering a single algorithm, consider a family of algorithms.

- HashMap
  - Hard inputs: 43, 21, 1
- HashMap
  - Hard inputs: 10, 15, 1
- HashMap
  - Hard inputs: 22, 33, 6
- HashMap
  - Hard inputs: 26, 43, 2

... Universal hashing

- HashMap
  - Hard inputs: 39, 2, 74
Avoiding the worst case

- Instead of considering a single algorithm, consider a **family of algorithms**.

**Randomized QuickSort**

- QuickSort₁(A)
  - Hard inputs: 43, 21, 1
- QuickSort₂(A)
  - Hard inputs: 10, 15, 1
- QuickSort₃(A)
  - Hard inputs: 22, 33, 6
- QuickSort₄(A)
  - Hard inputs: 26, 43, 2
- QuickSort₂₃₂(A)
  - Hard inputs: 39, 2, 74
The worst case can be common

- Instead of considering a single algorithm, consider a **family of algorithms**.

- Search₁(A) → Does badly in 99% cases
- Search₂(A) → Does badly in 99% cases
- Search₃(A) → Does badly in 99% cases
- Search₄(A) → Does badly in 99% cases

- Search₂₃₂(A) → Does badly in 99% cases

Randomized search
You may need to avoid symmetry

Instead of considering a single algorithm, consider a **family of algorithms**.
Las Vegas Algorithms

- This approach is called “Las Vegas algorithms”

- The set of problems which have a polynomial Las-Vegas solution is called ZPP (Zero error Probabilistic Polynomial)

Obviously, $P \subseteq ZPP$.
It is not known whether $P = ZPP$.
Another reason for randomness

- Sometimes, *sampling from a probability distribution* around the correct answer is much easier than producing the answer.

![Diagram showing a family of functions with yes and no outcomes.](image)

Again, a family of functions.

We know that most produce the correct answer (but we don’t know which ones).
Example

- Given a property, that holds for at least 50% of nodes, find a node with that property (e.g. find a leaf in a binary tree)
Example

- Given a property, that holds for at least 50% of nodes, find a node with that property (e.g. find a leaf in a binary tree)

```javascript
function find(objects, randomness) {
    return (random object)
}
```
Example

- **Amplification:**
  - One run: fails with probability 0.50
  - Two runs: fails with probability 0.25
  - 64 runs: fails with probability $2^{-64}$
Monte-Carlo Algorithms

- Monte-Carlo algorithms have nondeterministic output, but deterministic run time. (for Las-Vegas it is the other way around)

- The set of Monte-Carlo solvable problems is called BPP (Bounded error Probability Polynomial-time)

\[ P \subset ZPP \subset BPP, \text{ but it is not known whether } ZPP=\text{BPP}. \]

Modern cryptography is all about Monte-Carlo algorithms

So are statistics, optimization, and data mining.
Remote file comparison

Problem: How to check that a local file is equal to a remote one?
Remote file comparison

- Problem: How to check that a local file is equal to a remote one?
- Solution: Verify that hashes are equal.
Monte-Carlo Examples

- Matrix multiplication verification
  - Problem: verify that $A = BC$
Monte-Carlo Examples

- **Matrix multiplication verification**
  - **Problem:** verify that \( A = BC \)
  - **Solution:** sample random vectors \( v \) and check that \( Av = B(Cv) \)
Monte-Carlo Examples

- Primality testing
  - Problem: verify that $p$ is prime
  - Solution* [Fermat test]: check that $a^p = a \mod p$
Monte-Carlo Examples

- Primality testing
  - Problem: verify that $p$ is prime
  - Solution* [Fermat test]:
    check that $a^p = a \mod p$

PRIMES is in P

Manindra Agrawal  Neeraj Kayal
Nitin Saxena*  2002, 2005
Monte-Carlo Examples

- Max 3-SAT
  - Given a set of 3-CNF formula, find a variable assignment that satisfies the largest number of clauses

\[
\begin{align*}
    x_1 & \lor \overline{x}_2 \lor x_4 \\
    x_2 & \lor x_5 \lor x_6 \\
    \overline{x}_2 & \lor x_5 \lor \overline{x}_8 \\
    x_4 & \lor \overline{x}_8 \lor x_9 \\
    x_5 & \lor x_6 \lor \overline{x}_7 \\
    \overline{x}_5 & \lor x_7 \lor \overline{x}_8 \\
    \overline{x}_6 & \lor \overline{x}_7 \lor x_9
\end{align*}
\]
Monte-Carlo Examples

Solution:

Pick a variable assignment at random!

Let \( Z_i = \begin{cases} 1 & \text{if clause } i \text{ is satisfied} \\ 0 & \text{otherwise.} \end{cases} \)

The total number of satisfied clauses is then

\[ Z = \sum_i Z_i \]

And the expected number is:

\[
E[Z] = E \left[ \sum_i Z_i \right] = \sum_i E[Z_i] = \sum_i \frac{7}{8} = \frac{7}{8}n
\]

Surprisingly, it is close to the best possible algorithm
Summary

- **Randomized algorithms**
  - Use a *family* of algorithms parameterized by $r$.
  - Choose $r$ *randomly* (i.e. $r$ is a total secret)
  - Use this secret to:
    - Avoid adversaries (Las-Vegas algorithms)
    - Speed up search (Las-Vegas algorithms)
    - Break symmetries (Las-Vegas communication)
    - Select one option of many (Monte-Carlo algorithms)
    - Verify things (Monte-Carlo verification)