Algorithms and Data Structures DV3
Arne Andersson
Today’s lectures
Textbook chapters 1-4, 5

- Introduction
- Overview of Algorithmic Mathematics
- Recurrences
  - Introduction to Recurrences
  - The Substitution Method
  - The Recursion Tree Method
  - The Master Method (Master Theorem)
- Probabilistic analysis and Randomized Algorithms
  - The Hiring Problem
  - Balls and Bins
Algorithmic Mathematics

- Used to analyze running time of algorithms
- With running time, we can mean
  - Running time on some specific or all input
  - Worst Case time
  - Best Case time
  - Average time or Expected time
  - (Expected Worst Case time)
  - More to come...
Algorithmic Mathematics

- Some major components
  - The logarithm function
  - Asymptotics
  - Recurrence equations (induction)
Algorithmic Mathematics

- Som major components
  - The logarithm function
    Binary search among $n$ keys requires $\lceil \log(n+1) \rceil$ comparisons
  - Asymptotics
    Binary search takes $\Theta(\log n)$ time
  - Recurrence equations (induction)
    \[
    \begin{cases}
    T(n) = T(n/2) + C_1 \\
    T(1) = C_2 \\
    \Rightarrow T(n) = \Theta(\log n)
    \end{cases}
    \]
Algorithmic Mathematics

- Som major components
  - The logarithm function
    Binary search among $n$ keys requires $\lceil \log(n + 1) \rceil$ comparisons
  - Asymptotics
    Binary search takes $\Theta(\log n)$ time
  - Recurrence equations (induction)

\[
\begin{align*}
T(n) &= T(n/2) + C_1 \\
T(1) &= C_2
\end{align*}
\]
\[\Rightarrow T(n) = \Theta(\log n)\]

Solution:
\[
T(n) = T(n/2) + C_1 = T(n/4) + 2C_1 = T(n/8) + 3C_1 \quad \text{...etc...} \\
&= T(1) + \log n \cdot C_1 = \Theta(\log n)
\]
The Logarithm function

- In Computer Science, \( \log(n) \) or \( \lg(n) \) means the logarithm in base 2
- (In Mathematics, we often use \( \ln \), the natural logarithm, and \( \lg \) as the logarithm in base 10)
- So, here,

\[
\log n = \lg n = \log_2 n
\]
What Is the logarithm function, really?

\[
\log n
\]
What Is the logarithm function, really?

- \( \log n \) is the number of bits we need to represent the numbers \( 0..n \)
- \( \log n \) is height of a perfectly balanced binary tree with \( n \) nodes
- \( \log n \) is the number of times we can repeat halving, starting with \( n \), until we only have one left
Some other functions, based on log

- \( \log n \) is the number of times we can repeat halfing, starting with \( n \), until we only have one left
- \( \log \log n \) is the number of times we can repeat taking the square root, starting with \( n \), until we only have two left
- \( \log^* n \) is the number of times we can repeat taking the logarithm, starting with \( n \), until we only have one left
<table>
<thead>
<tr>
<th></th>
<th>log $n$</th>
<th>$\lceil \log \log n \rceil$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>32</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>64</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>128</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>256</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>512</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>1024</td>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>$10^3$</td>
<td>$\approx$ 10</td>
<td>4</td>
</tr>
<tr>
<td>$10^6$</td>
<td>$\approx$ 20</td>
<td>5</td>
</tr>
<tr>
<td>$10^9$</td>
<td>$\approx$ 30</td>
<td>5</td>
</tr>
</tbody>
</table>
Asymptotic notation: $O, \Omega, \Theta, o, \omega$

\[
\begin{align*}
  f(n) &= O(g(n)) \quad f(n) \text{ grows as most as } g(n) \\
  f(n) &= \Omega(g(n)) \quad f(n) \text{ grows as least as } g(n) \\
  f(n) &= \Theta(g(n)) \quad f(n) \text{ grows as } g(n) \\
  f(n) &= o(g(n)) \quad f(n) \text{ grows slower than } g(n) \\
  f(n) &= \omega(g(n)) \quad f(n) \text{ grows faster than } g(n)
\end{align*}
\]
A simple Rule of Thumb:

We only need to consider the leading term

\[ f(n) = n^2 + n \log n + \frac{2^n}{\text{leading term}} + \sqrt{\log \log n} \]

\[ f(n) = \Theta(2^n) \]
Som more mathematics

- Monotonicity of functions
- Floors and ceilings
- Modular arithmetic
- Polynomials
- Exponentials, Logarithms
- Factorials
- Functional iteration, f*
- Fibonnaci numbers

**Monotonicity of functions**

\[ \left\lfloor 3.14 \right\rfloor = 3 \quad \left\lceil 3.14 \right\rceil = 4 \]

**Floors and ceilings**

\[ x \mod 2 = 1 \text{ iff } x \text{ is odd} \]

**Modular arithmetic**

If \( f(n) \) is polynomially bounded if \( f(n) = O(n^i) \) for some constant \( i \)

So, \( n^{10} \sqrt{\log n} \) is polynomially bounded

**Polynomials**

\[ \log(2^n) = n \quad 2^{\log n} = n \]

\[ 2^n \cdot 2^{\sqrt{n}} = 2^{n + \sqrt{n}} \quad \log(xy) = \log x + \log y \]

**Exponentials, Logarithms**

\[ n! = 1 \cdot 2 \cdot 3 \cdots n \quad n! = o(n^n) \]

\[ n! = \omega(2^n) \quad \log(n!) = \Theta(n \log n) \]

**Factorials**

**Functional iteration, \( f^* \)**

**Log\(^*\)n**

\[ F_0 = 0 \quad F_1 = 1 \quad F_i = F_{i-1} + F_{i-2} \]

\[ 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots \]
Today’s lectures
Textbook chapters 1-4, 5

- Introduction
- Overview of Algorithmic Mathematics
- **Recurrences**
  - Introduction to Recurrences
  - The Substitution Method
  - The Recursion Tree Method
  - The Master Method (Master Theorem)
- Probabilistic analysis and Randomized Algorithms
  - The Hiring Problem
  - Balls and Bins
Why do we need recurrences?

- A recursive equation corresponds to the standard algorithmic technique of divide-and-conquer:
  1. Divide the problem into subproblems
  2. Solver (some of) the subproblems recursively
  3. Combine the solutions to the subproblems

\[
T(\text{problem}) = T_{\text{Divide and Combine}}(\text{problem}) + \sum_{\text{Subproblems}} T(\text{subproblem})
\]
Examples

- **Binary search**
  \[
  \begin{align*}
  T(n) &= T(n/2) + \Theta(1) \\
  T(1) &= \Theta(1)
  \end{align*}
  \]

- **Merge Sort**
  \[T(n) = 2T(n/2) + \Theta(n)\]

\[
T(\text{problem}) = T_{\text{Divide and Combine}}(\text{problem}) + \sum_{\text{Subproblems}} T(\text{subproblem})
\]
Today’s lectures
Textbook chapters 1-4, 5

- Introduction
- Overview of Algorithmic Mathematics
- Recurrences
  - Introduction to Recurrences
  - **The Substitution Method**
  - The Recursion Tree Method
  - The Master Method (Master Theorem)
- Probabilistic analysis and Randomized Algorithms
  - The Hiring Problem
  - Balls and Bins
General Method

1. Guess the form of the solution
2. Use induction to
   - Find constants in solution
   - Prove the solution
Some tricks

1. Try to ignore all "unimportant" constants and lower-order terms
2. If things only "almost" work out, try to extend the guess by subtracting a constant term
3. Change (substitute) variables
Today’s lectures
Textbook chapters 1-4, 5

- Introduction
- Overview of Algorithmic Mathematics
- Recurrences
  - Introduction to Recurrences
  - The Substitution Method
  - The Recursion Tree Method
  - The Master Method (Master Theorem)
- Probabilistic analysis and Randomized Algorithms
  - The Hiring Problem
  - Balls and Bins
General Method

1. Develop the recursive equation a number of levels
2. If possible, solve by concluding how the entire sum will look like
3. Otherwise, inspect a number of levels, see the tendency, and guess a solution, which can then be proven by other methods
Today’s lectures
Textbook chapters 1-4, 5

- Introduction
- Overview of Algorithmic Mathematics
- Recurrences
  - Introduction to Recurrences
  - The Substitution Method
  - The Recursion Tree Method
  - The Master Method (Master Theorem)
- Probabilistic analysis and Randomized Algorithms
  - The Hiring Problem
  - Balls and Bins
The Master Theorem

Master Theorem

\[ T(n) = aT(n/b) + f(n) \] has the following solutions

1. \[ f(n) = O(n^{\log_b a - \varepsilon}) \quad \Rightarrow \quad T(n) = \Theta(n^{\log_b a}) \]
2. \[ f(n) = \Theta(n^{\log_b a}) \quad \Rightarrow \quad T(n) = \Theta(n^{\log_b a \log n}) \]
3. \[ \begin{cases} f(n) = \Omega(n^{\log_b a + \varepsilon}) \\ af(n/b) \leq cf(n) \end{cases} \quad \Rightarrow \quad T(n) = \Theta(f(n)) \]
Looks complicated, but let’s try to understand it

Master Theorem

\[ T(n) = aT(n/b) + f(n) \] has the following solutions

1. \( f(n) = O(n^{\log_b a - \varepsilon}) \) A dominates \( \Rightarrow T(n) = \Theta(n^{\log_b a}) \)

2. \( f(n) = \Theta(n^{\log_b a}) \) A and B balances \( \Rightarrow T(n) = \Theta(n^{\log_b a} \log n) \)

3. \[ \begin{cases} f(n) = \Omega(n^{\log_b a + \varepsilon}) \\ af(n/b) \leq cf(n) \end{cases} \] B dominates \( \Rightarrow T(n) = \Theta(f(n)) \)
Today’s lectures
Textbook chapters 1-4, 5

- Introduction
- Overview of Algorithmic Mathematics
- Recurrences
  - Introduction to Recurrences
  - The Substitution Method
  - The Recursion Tree Method
  - The Master Method (Master Theorem)
- **Probabilistic analysis and Randomized Algorithms**
  - The Hiring Problem
  - Balls and Bins
The Hiring Problem

- An agency gives you a list of n persons
- You interview them one-by-one
- After each interview, you must immediately decide if this candidate should be hired
- You can change your mind if a better one comes up later, but that will cost
The Hiring Problem: straightforward algorithm

Always change your mind if a better one shows up

Given n candidates, how many times will we change our mind?

Worst case: n times
Expected case: ?
**Worst case:** $n$ times (better and better candidates)
Expected case: ?

We need **probabilistic analysis**
Probabilistic analysis

Use an indicator variable

\[ X_i = \begin{cases} 
1 & \text{if candidate } i \text{ is hired} \\
0 & \text{otherwise} 
\end{cases} \]

\[ E(X_i) \text{ denotes the expected value of } X_i \]

Expected number of re-hiring: \( E \left( \sum_{i=1}^{n} X_i \right) = \sum_{i=1}^{n} E(X_i) \)
Use an indicator variable

\[ X_i = \begin{cases} 1 & \text{if candidate } i \text{ is hired} \\ 0 & \text{otherwise} \end{cases} \]

\( E(X_i) \) denotes the expected value of \( X_i \)

Expected number of re-hiring:

\[ E \left( \sum_{i=1}^{n} X_i \right) = \sum_{i=1}^{n} E(X_i) \]

\[ E(X_i) = \Pr(\text{candidate } i \text{ is hired}) \]

\[ \Pr(\text{candidate } i \text{ is hired}) = \Pr(\text{candidate } i \text{ is better than all before}) = \frac{1}{i} \]

\[ \sum_{i=1}^{n} E(X_i) = \sum_{i=1}^{n} \frac{1}{i} = \ln n + O(1) \]
Probabilistic analysis: problem of "trust"

- We assume that the agency’s list is randomly ordered
- What if not? (If we pay the agency each time we re-hire.....)
Solution: Use a randomized algorithm

- Shuffle the list randomly before the interviews
- Now, the probabilistic analysis holds independent of the agency’s behaviour
The Hiring Problem: randomized algorithm

1. Permute the list randomly
2. Always change your mind if a better one shows up

Given $n$ candidates, how many times will we change our mind?

(Worst case: $n$ times)

Expected case: $\ln n + O(1)$ times
To think about

1. How do we permute an array randomly?

2. On-line Hiring Problem: We can not change our mind, we can only hire one. How do we do this?
Today’s lectures
Textbook chapters 1-4, 5

- Introduction
- Overview of Algorithmic Mathematics
- Recurrences
  - Introduction to Recurrences
  - The Substitution Method
  - The Recursion Tree Method
  - The Master Method (Master Theorem)
- Probabilistic analysis and Randomized Algorithms
  - The Hiring Problem
  - Balls and Bins
Balls and Bins

Balls are thrown in bins, each ball is thrown randomly and independently

n balls, b bins
Balls and Bins

Balls are thrown in bins, each ball is thrown randomly and independently

\( n \) balls, \( b \) bins

- Expected number of balls in a given bin: \( \frac{n}{b} \)
- Expected number of until a given bin gets one: \( b \)
- Expected number of balls until each bin has a ball: \( b \ln b + O(1) \)

If \( b = n \) (i.e., we throw \( n \) balls)

- Expected maximum number of balls in a bin: \( \Theta(\ln n) \)
- Expected number of consecutive empty bins: \( \Theta(\ln n) \)
- Expected value of \( \sum_{\text{all bins}} (\text{number of balls})^2 = \Theta(n) \)