Algorithms and Data Structures

Yerlan Kuzbakov

Lecture 1 – Introduction
Complexity, Data structures

Algorithms

- Comes from the name of mathematician **al-Khwarizmi** (c. 9th century).
- Precise, step-by-step procedure.

In 1962, Procter & Gamble offered a **$10,000 prize** to solve a challenging instance of the Traveling Salesman Problem.

The instance involved 33 cities.

Figure: TSP instance with 33 cities (P&G, 1962)

Combinatorial Explosion

- Many problems have solution spaces that grow **exponentially** with input size.
- Example: Traveling Salesman Problem — $n!$ possible tours.

Asymptotic Notation Examples

- **Big-**O: Upper bound (worst-case) growth.
- **Omega** Ω: Lower bound (best-case guarantee).
- **Theta** Θ: Tight bound (exact asymptotic growth).

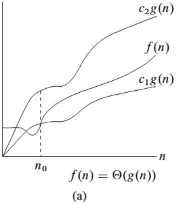


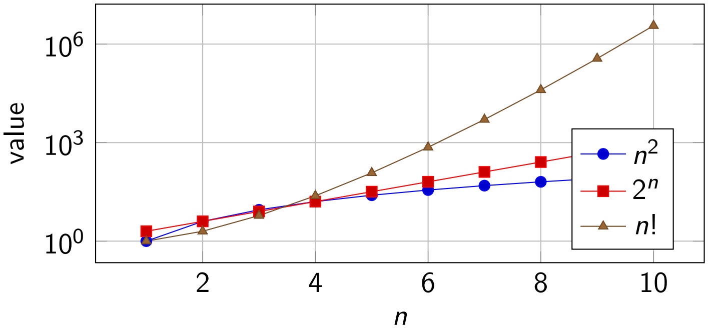
Illustration of O, Ω, Θ examples

Complexity and Memory

- Algorithms consume two main resources:
  - **Time** (steps, operations, "crank turns", instructions).
  - **Space** (memory cells, registers, tape length).

Examples of Exponential Growth

- **Password Breaking**
  - n characters, each from 26 letters.
  - Possibilities: 26^n.
  - For $n = 8$: $26^8 \approx 2 \times 10^{11}$.
- **Traveling Salesman Problem (TSP)**
  - n cities, brute force checks all tours.
  - Complexity: $(n-1)!/2 \sim O(n!)$.
  - For $n = 20$: $\approx 2.4 \times 10^{18}$ tours.



10^6
10^3
10^0
value
2
4
6
8
10
n
n^2
2^n
$n!$

Prime Factorization: RSA Numbers

- **RSA numbers**: large composite integers published by RSA Laboratories.
- Constructed as the product of two large primes.
- Challenge (1991–2007 but people are still trying): factor these numbers to test practical hardness of integer factorization.
- Sizes ranged from 100 to 617 decimal digits.
- **Motivation:** Basis of RSA cryptosystem security — difficulty of factoring.
- RSA-250 (250 decimal digits) factored in 2020

Algorithm: Trial Division (Naïve Factorization)

Input: Integer n to be factored
Output: List F of prime factors of n

```
algorithm Trial-Division(n):
    P <- set of all primes <= sqrt(n)
    F <- empty list

    for each prime p in P do
        while n mod p = 0 do
            add p to F
            n <- n / p

    if F is empty then
        add n to F     # n is prime
    return F
```

Complexity: $O\left(\frac{\sqrt{n}}{\log n}\right)$ divisions (using primes up to $\sqrt{n}$).

Greatest Common Divisor (GCD)

- **Definition:** For integers a, b, not both zero, $\gcd(a, b)$ is the largest integer d such that $d \mid a$ and $d \mid b$.

Euclid's Algorithm (Recursive Definition)

Definition:

$$\text{EUCLID}(a, b) = \begin{cases} a & \text{if } b = 0, \\ \text{EUCLID}(b,\ a \bmod b) & \text{otherwise.} \end{cases}$$

Example: $\gcd(30, 21)$

$$\begin{aligned} \text{EUCLID}(30, 21) &= \text{EUCLID}(21,\ 30 \bmod 21) = \text{EUCLID}(21, 9) \\ &= \text{EUCLID}(9,\ 21 \bmod 9) = \text{EUCLID}(9, 3) \\ &= \text{EUCLID}(3,\ 9 \bmod 3) = \text{EUCLID}(3, 0) \\ &= 3 \end{aligned}$$

Each step reduces the second argument, guaranteeing termination.

Stack in C++

- **Definition:** A stack is a linear data structure that follows the **LIFO** principle (Last In, First Out).
- **C++ STL:** Provided by `<stack>` container adapter.
- **Basic operations:**
  - `push(x)` – insert element on top
  - `pop()` – remove top element
  - `top()` – access current top element
  - `empty()` – check if stack is empty
  - `size()` – number of elements

Stack Example (C++)

```
#include <stack>
#include <iostream>
using namespace std;

stack<int> st;
st.push(10);
st.push(20);
cout << st.top(); // 20
st.pop();         // remove 20
```

Queue in C++

- **Definition:** A queue is a linear data structure that follows the **FIFO** principle (First In, First Out).
- **C++ STL:** Provided by `<queue>` container adapter.
- **Basic operations:**
  - `push(x)` – insert element at the back
  - `pop()` – remove element from the front
  - `front()` – access first element
  - `back()` – access last element
  - `empty()`, `size()`

Queue Example (C++)

```
#include <queue>
#include <iostream>
using namespace std;

queue<int> q;
q.push(10);
q.push(20);
cout << q.front(); // 10
q.pop();           // remove 10
```

Deque in C++

- **Definition:** A deque (double-ended queue) is a sequence container allowing insertion and deletion at both ends.
- **C++ STL:** Provided by `<deque>`.
- **Basic operations:**
  - `push_front(x)`, `push_back(x)`
  - `pop_front()`, `pop_back()`
  - `front()`, `back()`
  - `empty()`, `size()`

Deque Example (C++)

```
#include <deque>
#include <iostream>
using namespace std;

deque<int> d;
d.push_back(10);
d.push_front(5);
cout << d.front(); // 5
d.pop_back();      // remove 10
```