10 Randomized algorithms
Randomized algorithms

- Classes of randomized algorithms
- Randomized Quicksort
- Randomized primality test
- Cryptography
1. Classes of randomized algorithms

- **Las Vegas** algorithms
  always correct; expected running time ("probably fast")

  Example: randomized Quicksort

- **Monte Carlo** algorithms (mostly correct):
  probably correct; guaranteed running time

  Example: randomized primality test
2. Quicksort

Unsorted range $A[l, r]$ in array $A$

Quicksort  Quicksort
Quickso\textit{r}t

\textbf{Algorithm:} Quick\textit{sort}

\textbf{Input:} unsorted range \([l, r]\) in array \(A\)

\textbf{Output:} sorted range \([l, r]\) in array \(A\)

1. \textbf{if} \(r > l\)
2. \textbf{then} choose pivot element \(p = A[r]\)
3. \hspace{1em} \(m = divide(A, l, r)\)
   /* Divide \(A\) according to \(p\):
   \[A[l], \ldots, A[m - 1] \leq p \leq A[m + 1], \ldots, A[r]\]
   */
4. \hspace{1em} Quickso\textit{r}t\hspace{-1em}
   \(A, l, m - 1\)
   \hspace{1em} Quickso\textit{r}t\hspace{-1em}
   \(A, m + 1, r\)
The divide step
The *divide* step
The *divide* step
The *divide* step

\[\text{divide}(A, l, r):\]

- returns the index of the pivot element in \( A \)
- can be done in time \( O(r - l) \)
Worst case input

*n elements:*

Running time: \((n-1) + (n-2) + \ldots + 2 + 1 = n \cdot (n-1)/2\)
3. Randomized Quicksort

Algorithm: Quicksort

Input: unsorted range \([l, r]\) in array \(A\)

Output: sorted range \([l, r]\) in array \(A\)

1. if \(r > l\)
2. then randomly choose a pivot element \(p = A[i]\) in range \([l, r]\)
3. swap \(A[i]\) and \(A[r]\)
4. \(m = \text{divide}(A, l, r)\)
   /* Divide \(A\) according to \(p\):
   \[
   A[l], ..., A[m - 1] \leq p \leq A[m + 1], ..., A[r]
   */
5. Quicksort\((A, l, m - 1)\)
6. Quicksort\((A, m + 1, r)\)
Analysis

$n$ elements; let $S_i$ be the $i$-th smallest element

- $S_1$ is chosen as pivot with probability $1/n$:
  - Sub-problems of sizes $0$ and $n-1$
    - 
      - 
      - 

- $S_k$ is chosen as pivot with probability $1/n$:
  - Sub-problems of sizes $k-1$ and $n-k$
    - 
      - 
      - 

- $S_n$ is chosen as pivot with probability $1/n$:
  - Sub-problems of sizes $n-1$ and $0$
Analysis

Expected running time:

\[ E(T(n)) = \frac{1}{n} \sum_{k=0}^{n-1} (E(T(k)) + E(T(n-k-1))) + \Theta(n) \]

\[ = \frac{2}{n} \sum_{k=0}^{n-1} E(T(k)) + \Theta(n) \]

\[ \leq \frac{2}{n} \sum_{k=2}^{n-1} E(T(k)) + \Theta(n) \quad \text{absorb } k=0,k=1 \text{ in } \Theta(n) \]
Analysis

\[ E(T(n)) = \frac{2}{n} \sum_{k=2}^{n-1} E(T(k)) + \Theta(n) \]

Prove: \( E[T(n)] \leq cn \log n \) for \( n \geq 2 \) and some \( c > 0 \)

Base case:
choose \( c \) large enough s.t. \( E[T(n)] \leq cn \log n \) for \( n = 2 \)

Inductive step:
substitute inductive hypothesis

\[ E(T(n)) \leq \frac{2}{n} \sum_{k=2}^{n-1} ck \log k + \Theta(n) \]
Analysis

\[ E(T(n)) \leq \frac{2c}{n} \sum_{k=2}^{n-1} k \lg k + \Theta(n) \]

\[ \sum_{k=2}^{n-1} k \lg k \leq \frac{n^2}{2} \lg n - \frac{n^2}{8} \]

(Proof as exercise!)

\[ E(T(n)) \leq \frac{2c}{n} \left( \frac{n^2}{2} \lg n - \frac{n^2}{8} \right) + \Theta(n) \]

\[ \leq cn \lg n - \left( \frac{cn}{4} - \Theta(n) \right) \]
Analysis

\[ E(T(n)) \leq c n \log n - \left( \frac{cn}{4} - \Theta(n) \right) \]
Analysis

\[ E(T(n)) \leq cn \log n - \left( \frac{cn}{4} - \Theta(n) \right) \]

desired

Should be \( \geq 0 \)
Analysis

\[ E(T(n)) \leq cn \lg n - \left( \frac{cn}{4} - \Theta(n) \right) \]

desired \hspace{1cm} \text{Should be } \geq 0

\[ E(T(n)) \leq cn \lg n \text{ where we choose } c \text{ large enough} \]
\[ \text{s.t. } cn/4 \text{ dominate } \Theta(n) \]
4. Primality test

**Definition:**
An integer $p \geq 2$ is **prime** iff $(a \mid p \Rightarrow a = 1 \text{ or } a = p)$.

**Algorithm:** deterministic primality test (naive)

**Input:** integer $n \geq 2$

**Output:** answer to the question: Is $n$ prime?

- if $n = 2$ then return **true**
- if $n$ even then return **false**
- for $i = 1$ to $\sqrt{n}/2$ do
  - if $2i + 1$ divides $n$
    - then return **false**
- return **true**

**Complexity:** $\Theta(\sqrt{n})$
Primality test

Goal:
Randomized method
- Polynomial time complexity (in the length of the input)
- If answer is “not prime”, then $n$ is not prime
- If answer is “prime”, then the probability that $n$ is not prime is at most $p>0$

$k$ iterations: probability that $n$ is not prime is at most $p^k$
Observation:
Each odd prime number $p$ divides $2^{p-1} - 1$.

Examples: $p = 17$, $2^{16} - 1 = 65535 = 17 \times 3855$
$p = 23$, $2^{22} - 1 = 4194303 = 23 \times 182361$

Simple primality test:
1. Calculate $z = 2^{n-1} \mod n$
2. if $z = 1$
3. then $n$ is possibly prime
4. else $n$ is definitely not prime

Advantage: This only takes polynomial time
Simple primality test

Definition:

$n$ is called pseudoprime to base 2, if $n$ is not prime and

$$2^{n-1} \mod n = 1.$$

Example:  

$n = 11 \times 31 = 341$

$$2^{340} \mod 341 = 1$$
Randomized primality test

**Theorem:** (Fermat‘s little theorem)
If \( p \) prime and \( 0 < a < p \), then
\[
a^{p-1} \mod p = 1.
\]

**Definition:**
\( n \) is pseudoprime to base \( a \), if \( n \) not prime and
\[
a^{n-1} \mod n = 1.
\]

**Example:** \( n = 341, \quad a = 3 \)
\[
3^{340} \mod 341 = 56 \neq 1
\]
Randomized primality test

Algorithm: Randomized primality test 1

1. Randomly choose \( a \in [2, n-1] \)
2. Calculate \( a^{n-1} \mod n \)
3. if \( a^{n-1} \mod n = 1 \)
4. then \( n \) is possibly prime
5. else \( n \) is definitely not prime
Carmichael numbers

**Problem:** Carmichael numbers

**Definition:** An integer $n$ is called **Carmichael number** if

$$a^{n-1} \mod n = 1$$

for all $a$ with $\text{GCD}(a, n) = 1$.  

(GCD = greatest common divisor)

**Example:**
Smallest Carmichael number: $561 = 3 \times 11 \times 17$
Randomized primality test 2

**Theorem:**
If \( p \) prime and \( 0 < a < p \), then the only solutions to the equation
\[ a^2 \mod p = 1 \]
are \( a = 1 \) and \( a = p - 1 \).

**Definition:**
\( a \) is called *non-trivial square root* of 1 mod \( n \), if
\[ a^2 \mod n = 1 \text{ and } a \neq 1, n - 1. \]

**Example:** \( n = 35 \)
\[ 6^2 \mod 35 = 1 \]
Fast exponentiation

Idea:
During the computation of $a^{n-1}$ (0 < $a$ < $n$ randomly chosen), test whether there is a non-trivial square root $1 \mod n$.

Method for the computation of $a^n$:

Case 1: [$n$ is even]

\[ a^n = a^{n/2} \times a^{n/2} \]

Case 2: [$n$ is odd]

\[ a^n = a^{(n-1)/2} \times a^{(n-1)/2} \times a \]
Fast exponentiation

**Example:**

\[ a^{62} = (a^{31})^2 \]
\[ a^{31} = (a^{15})^2 \cdot a \]
\[ a^{15} = (a^{7})^2 \cdot a \]
\[ a^{7} = (a^{3})^2 \cdot a \]
\[ a^{3} = (a)^2 \cdot a \]

**Complexity:** \( O(\log^2 a^n \log n) \)
boolean isProbablyPrime;

power(int a, int p, int n) {
    /* computes $a^p \mod n$ and checks during the computation whether there is an $x$ with $x^2 \mod n = 1$ and $x \neq 1, n-1$ */

    if (p == 0) return 1;
    x = power(a, p/2, n)
    result = (x * x) % n;
Fast exponentiation

/* check whether \( x^2 \mod n = 1 \) and \( x \neq 1, n-1 \) */

if (result == 1 && x != 1 && x != n - 1)
    isProbablyPrime = false;

if (p % 2 == 1)
    result = (a * result) % n;

return result;

Complexity: \( O(\log^2 n \log p) \)
Randomized primality test 2

```c
primalityTest(int n) {
    /* carries out the randomized primality test for
       a randomly selected a */

    a = random(2, n-1);

    isProbablyPrime = true;

    result = power(a, n-1, n);

    if (result != 1 || !isProbablyPrime)
        return false;
    else
        return true;
}
```
Randomized primality test 2

Theorem:

If \( n \) is not prime, there are at most \( \frac{n-1}{4} \) integers \( 0 < a < n \), for which the algorithm \texttt{primalityTest} fails.
Application: cryptosystems

Traditional encryption of messages with secret keys

Disadvantages:
1. The key $k$ has to be exchanged between A and B before the transmission of the message.
2. For messages between $n$ parties $n(n-1)/2$ keys are required.

Advantage:
Encryption and decryption can be computed very efficiently.
Desired properties of cryptographic systems

- confidential transmission
- integrity of data
- authenticity of the sender
- reliable transmission
Public-key cryptosystems

Diffie and Hellman (1976)

**Idea:** Each participant $A$ has two keys:

1. a **public** key $P_A$ accessible to every other participant
2. a **private** (or: **secret**) key $S_A$ only known to $A$. 
Public-key cryptosystems

\[ D = \text{set of all legal messages,} \]
\[ \text{e.g. the set of all bit strings of finite length} \]

\[ P_A, S_A : D \rightarrow D \]

**Three conditions:**

1. \( P_A \) and \( S_A \) can be computed efficiently

2. \( S_A(P_A(M)) = M \) and \( P_A(S_A(M)) = M \)
   \( (P_A \) is the inverse function of \( S_A \) and vice-versa)\)

3. \( S_A \) cannot be computed from \( P_A \) (without unreasonable effort)
Encryption in a public-key cryptosystem

A sends a message $M$ to $B$.

Dear Bob,
I just checked the new ...

#*k- + ;}?,, @-) #$<9 
{07:--&$3 (-#!]?8 ...

Dear Bob,
I just checked the new ...

22.05.2012 Theory 1 - Randomized algorithms
Encryption in a public-key cryptosystem

1. **A** accesses **B**’s public key $P_B$ (from a public directory or directly from **B**).

2. **A** computes the encrypted message $C = P_B(M)$ and sends $C$ to **B**.

3. After **B** has received message $C$, **B** decrypts the message with his own private key $S_B$: $M = S_B(C)$
Generating a digital signature

\(A\) sends a digitally signed message \(M'\) to \(B\):

1. \(A\) computes the digital signature \(\sigma\) for \(M'\) with her own private key:
   \[
   \sigma = S_A(M')
   \]

2. \(A\) sends the pair \((M',\sigma)\) to \(B\).

3. After receiving \((M',\sigma)\), \(B\) verifies the digital signature:
   \[
   P_A(\sigma) = M'
   \]

\(\sigma\) can be verified by anybody via the public \(P_A\).
RSA cryptosystems

R. Rivest, A. Shamir, L. Adleman

Generating the public and private keys:

1. Randomly select two primes $p$ and $q$ of similar size, each with $l+1$ bits ($l \geq 500$).

2. Let $n = p \cdot q$

3. Let $e$ be an integer that does not divide $(p - 1) \cdot (q - 1)$.

4. Calculate $d = e^{-1} \mod (p - 1)(q - 1)$
   
   i.e.:
   
   $d \cdot e \equiv 1 \mod (p - 1)(q - 1)$
RSA cryptosystems

5. Publish $P = (e, n)$ as public key

6. Keep $S = (d, p, q)$ as private key

Divide message (described in a binary string) in blocks of size $2^l$.
Interpret each block $M$ as a binary number: $0 \leq M < 2^{2^l}$

$$P(M) = M^e \mod n \quad S(C) = C^d \mod n$$