Algorithm Theory

03 - Randomization

Dr. Alexander Souza

Randomization

- Types of randomized algorithms
- Randomized Quicksort
- Randomized primality test
- Cryptography

1. Types of randomized algorithms

- **Las Vegas algorithms**
  always correct; expected running time

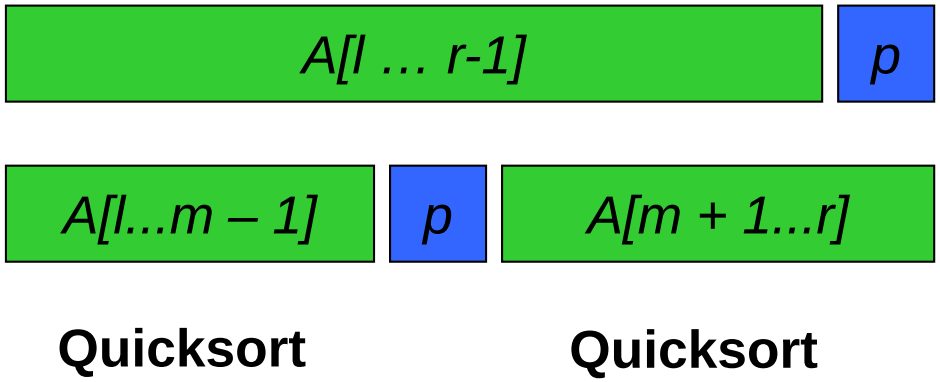
  Example: randomized Quicksort

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

  Example: randomized primality test

2. Quicksort

Unsorted range *A*[*l*,*r*] in an array *A*:



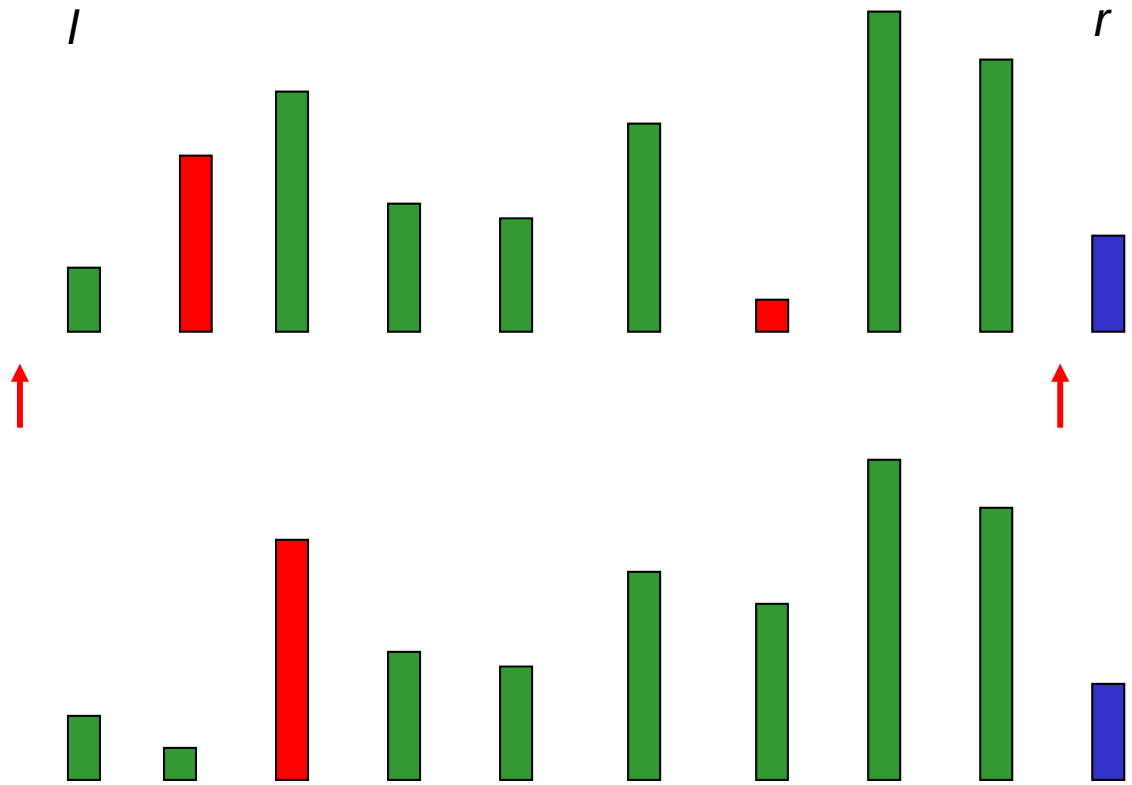
Quicksort

Algorithm: Quicksort

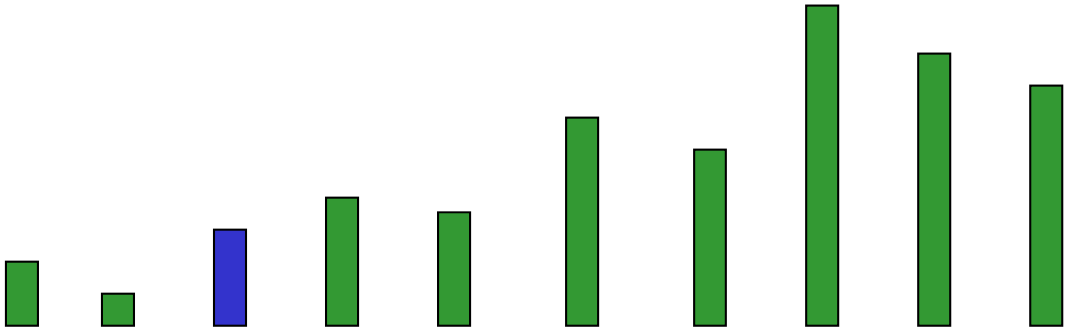
Input: unsorted range [*l, r*] in array *A*
Output: sorted range [*l, r*] in array *A*

```
1  if  r > l
2     then choose pivot element p = A[r]
3          m = divide(A, l , r)
           /* divide A according to p:
           A[l],....,A[m – 1] ≤ p ≤ A[m + 1],...,A[r]
           */
4          Quicksort(A, l , m - 1)
5          Quicksort (A, m + 1, r)
```

Partitioning step



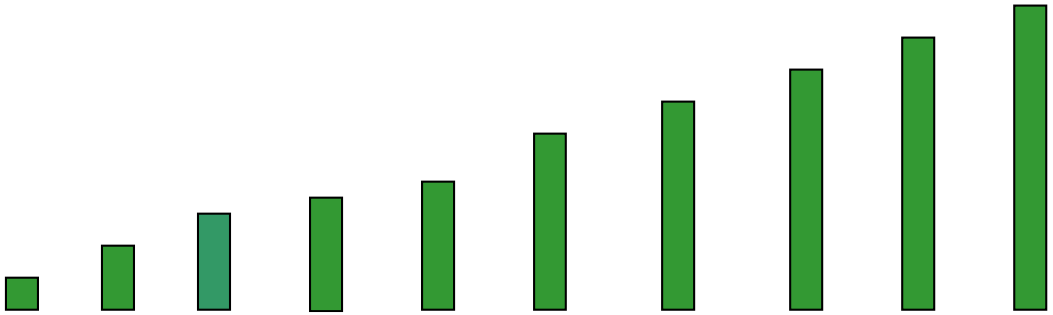
Partitioning step



divide(*A, l , r*):

- returns the index of the pivot element in *A*
- running time *O(r – l)*

Worst-case input



n elements*:*

Running time: *(n-1) + (n-2) + … + 2 + 1 = n(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  choose pivot element p = A[i] in the range [l, r] at random
3           swap A[i] and A[r]
4           m = divide(A, l, r)
            /* divide A according to p:
            A[l],....,A[m – 1] ≤ p ≤ A[m + 1],...,A[r]
            */
5           Quicksort(A, l, m - 1)
6           Quicksort(A, m + 1, r)
```

Analysis 1

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

With probability $1/n$, S_1 is the pivot element:
subproblems of sizes *0* and *n-1*

-
-
-

With probability $1/n$, S_k is the pivot element:
subproblems of sizes *k-1* and *n-k*

-
-
-

With probability $1/n$, S_n is the pivot element:
subproblems of sizes *n-1* and *0*

Analysis 1

Expected running time:

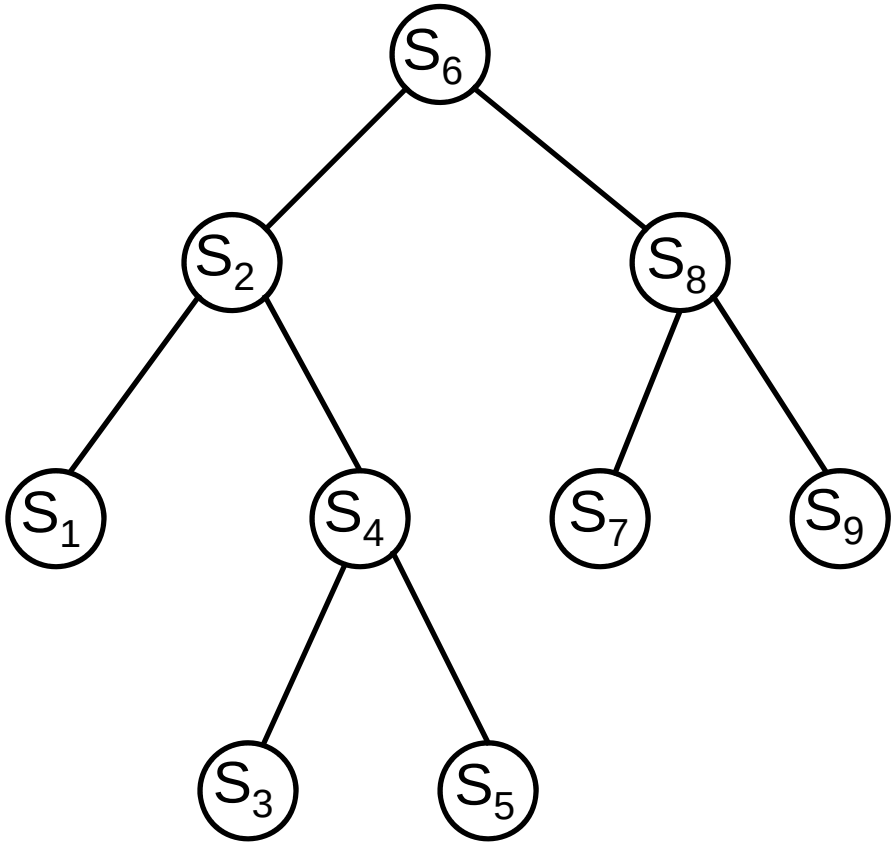
$$T(n) = \frac{1}{n}\sum_{k=1}^{n}\left(T(k-1)+T(n-k)\right)+n-1$$

$$= \frac{2}{n}\sum_{k=0}^{n-1}T(k)+n-1$$

$$= O(n\log n)$$

Analysis 1

Analysis 2: Representation of QS as a tree



$\pi = S_6\ S_2\ S_8\ S_1\ S_4\ S_7\ S_9\ S_3\ S_5$

Analysis 2

Expected number of comparisons:

$$X_{ij} = \begin{cases} 1 & \text{if } S_i \text{ is compared to } S_j \\ 0 & \text{otherwise} \end{cases}$$

$$E\left[\sum_{i=1}^{n} \sum_{j>i} X_{ij}\right] = \sum_{i=1}^{n} \sum_{j>i} E[X_{ij}]$$

p_{ij} = probability that S_i is compared to S_j

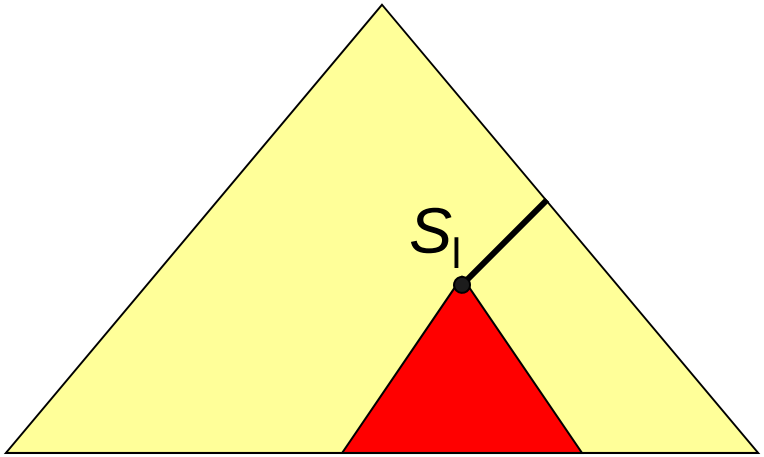
$$E[X_{ij}] = 1 \cdot p_{ij} + 0 \cdot (1 - p_{ij}) = p_{ij}$$

Linearity of Expectation

Computing p_{ij}

- S_i is compared to S_j iff S_i or S_j are chosen as pivot element before any S_l , $i<l<j$.
  $\{S_i \ \ldots \ S_l \ \ldots \ S_j\}$

- Any element S_i , … , S_j is chosen as pivot element with the same probability.



$\{\ldots S_i \ \ldots \ S_l \ \ldots \ S_j \ \ldots\}$

$$p_{ij} = 2 \,/\, (j\text{-}i\text{+}1)$$

Analysis 2

Expected number of comparisons:

$$\sum_{i=1}^{n} \sum_{j>i} p_{ij} = \sum_{i=1}^{n} \sum_{j>i} \frac{2}{j-i+1}$$

$$= \sum_{i=1}^{n} \sum_{k=2}^{n-i+1} \frac{2}{k}$$

$$\leq 2 \sum_{i=1}^{n} \sum_{k=1}^{n} \frac{1}{k}$$

$$= 2n \sum_{k=1}^{n} \frac{1}{k}$$

$$H_n = \sum_{k=1}^{n} 1/k \approx \ln n$$

4. Primality test

Definition:

A natural number $p \geq 2$ is prime iff $a \mid p$ implies that $a = 1$ or $a = p$.

We consider primality tests for numbers $n \geq 2$.

Algorithm: Deterministic primality test (naive approach)

Input: Natural number $n \geq 2$

Output: Answer to the question „Is n prime?“

```
if n = 2 then return true
if n even then return false
for i = 1 to √n / 2 do
        if 2i + 1 divides n
        then return false
return true
```

Running time: $\Theta(\sqrt{n})$

Primality test

Goal:

Randomized algorithm

- Polynomial running time.
- If it returns “not prime”, then *n* is not prime.
- If it returns “prime”, then with probability at most *p, p>0,* *n* is composite.

After *k* iterations: with probability p^k , *n* is composite .

Primality test

Fact: For any odd prime number *p:* $2^{p-1} \bmod p = 1$.

Examples: $p = 17$, $2^{16} - 1 = 65535 = 17 * 3855$

$p = 23$, $2^{22} - 1 = 4194303 = 23 * 182361$

Simple primality test:

1 Compute $z = 2^{n-1} \bmod n$

2 **if** $z = 1$

3 **then** *n* is possibly prime

4 **else** *n* is composite

Advantage: polynomial running time.

Simple primality test

Definition:

A natural number $n \geq 2$ is a base-2 pseudoprime if n is composite and

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

Example: $n = 11 * 31 = 341$

$$2^{340} \bmod 341 = 1$$

Randomized primality test

Theorem: (Fermat's little theorem)
If p is prime and $0 < a < p$, then

$$a^{p-1} \bmod p = 1.$$

Example: $n = 341$, $a = 3$: $3^{340} \bmod 341 = 56 \neq 1$

Algorithm: Randomized primality test 1

1 Choose a in the range $[2, n-1]$ uniformly at random
2 Compute $a^{n-1} \bmod n$
3 **if** $a^{n-1} \bmod n = 1$
4 **then** n is probably prime
5 **else** n is composite

Prob(n is composite but $a^{n-1} \bmod n = 1$) ?

Problem: Carmichael numbers

Definition:

A natural number $n \geq 2$ is a base-a pseudoprime if n is composite and

$$a^{n-1} \bmod n = 1.$$

Definition: A number $n \geq 2$ is a Carmichael number if n is composite and for any a with GCD(a, n) = 1 we have

$$a^{n-1} \bmod n = 1.$$

Example:

Smallest Carmichael number: 561 = 3 * 11 * 17

Randomized primality test 2

Theorem: If p is prime and $0 < a < p$, then the equation

$$a^2 \bmod p = 1$$

has exactly the two solutions $a = 1$ and $a = p - 1$.

Definition: A number a is a non-trivial square root mod n if

$$a^2 \bmod n = 1 \text{ and } a \neq 1, n - 1.$$

Example: $n = 35$

$6^2 \bmod 35 = 1$

Fast exponentiation

Idea:

While computing a^{n-1}, where $0 < a < n$ is chosen uniformly at random, check if a non-trivial square root mod n exists.

Method for computating a^n:

Case 1: [n is even]

$a^n = a^{n/2} * a^{n/2}$

Case 2: [n is odd]

$a^n = a^{(n-1)/2} * a^{(n-1)/2} * a$

Fast exponentiation

Example:

$a^{62} = (a^{31})^2$

$a^{31} = (a^{15})^2 * a$

$a^{15} = (a^7)^2 * a$

$a^7 = (a^3)^2 * a$

$a^3 = (a)^2 * a$

Running time: $O(\log^2 a^n \log n)$

Fast exponentiation

```
boolean isProbablyPrime;

power(int a, int p, int n){

    /* computes a^p mod n and checks if a number x with x^2 mod n = 1
    and x ≠ 1, n-1 occurs during the computation */

    if (p == 0) return 1;

    x = power(a, p/2, n)
    result = (x * x) % n;
```

Fast exponentiation

```
    /* check if x^2 mod n = 1 and x ≠ 1, n-1 */

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

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

    return result;
}
```

Running time: $O(\log^2 n \log p)$

Randomized primality test 2

```
primeTest(int n) {

    /* executes the randomized primality test for a chosen at random */

    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 composite, then there are at most

$$\frac{n-9}{4}$$

numbers $0 < a < n$, for which the algorithm primeTest fails.

5. Application

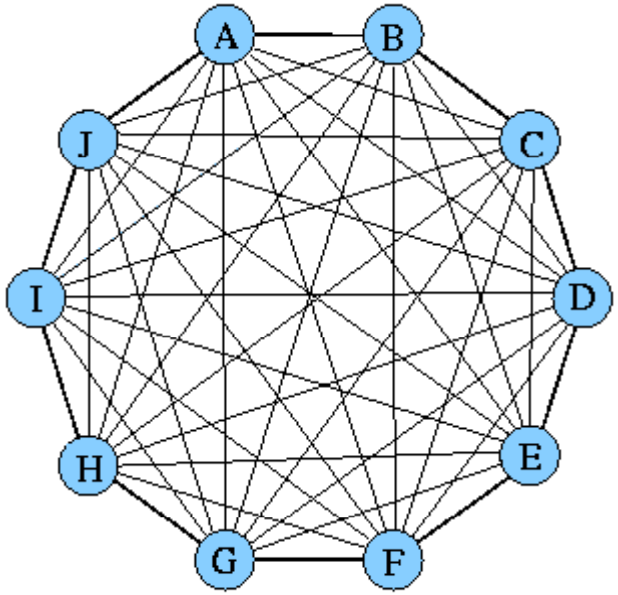
Public-Key Cryptosystems

Secret key cryptosystems

Traditional encryption of messages

Disadvantages:

1. Prior to transmission of the message, the key *k* has to be exchanged between the parties A und B.
2. For encryption of messages between *n* parties, $n(n-1)/2$ keys are required.



Secret key encryption systems

Advantage:

Encryption and decryption are fast.

Electronic security services

Guarantees:

- Confidentiality of the transmission
- Integrity of the data
- Authenticity of the sender
- Liability of the transmission

Public-key cryptosystems

Diffie and Hellman (1976)

Idea: Each participant *A* holds two keys:

1. A public key P_A , accessible to all other participants.
2. A private key S_A that is kept secret.

Public-key cryptosystems

D = Set of all valid messages,
e.g. set of all bitstrings of finite length

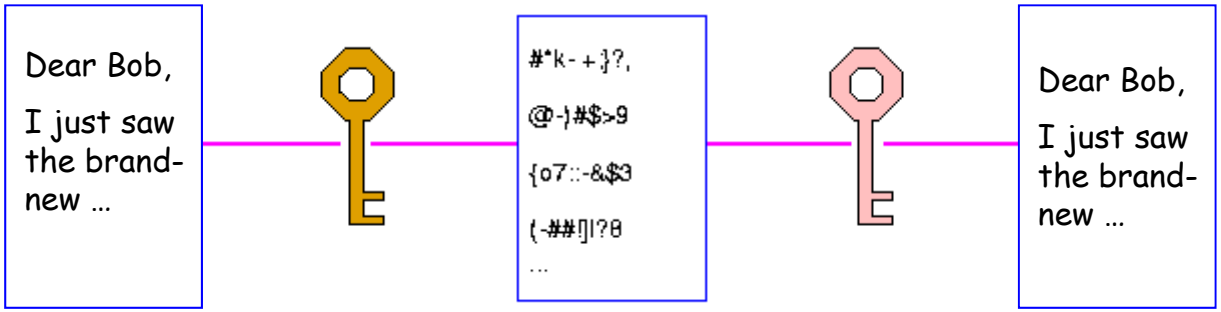
$$P_A(\ ), S_A(\ ): D \overset{1-1}{\rightarrow} D$$

Three constraints:

1. $P_A()$, $S_A()$ efficiently computable
2. $S_A(P_A(M)) = M$ and $P_A(S_A(M)) = M$
3. $S_A()$ is not computable from $P_A()$ (with realistic effort)

Encryption in a public-key system

A sends a message *M* to *B*:



Encryption in a public key system

1. *A* receives *B`s* public key P_B from a public directory or directly from *B*.

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

3. After receiving message *C*, *B* decrypts the message using his 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 σ for M' using his private key:

$$\sigma = S_A(M')$$

2. *A* sends the pair (M', σ) to *B.*

3. After receiving (M', σ), *B* checks the digital signature: $P_A(\sigma) = M'$

Anybody is able to check σ using P_A (e.g. for bank checks).

RSA cryptosystem

R. Rivest, A. Shamir, L. Adleman

Generating the public and private keys:

1. Select at random two large primes p and q of $l+1$ bits ($l \geq 500$).

2. Compute $n = pq$.

3. Select a natural number e is that is relatively prime to $(p-1)(q-1)$.

4. Compute $d = e^{-1}$

$$d*e \equiv 1 \pmod{(p-1)(q-1)}$$

RSA cryptosystem

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

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

Split the (binary coded) message into blocks of length $2l$.
Interpret each block M as a binary number: $0 \leq M < 2^{2l}$

$$P(M) = M^e \bmod n \qquad S(C) = C^d \bmod n$$

Multiplicative inverse

Theorem: (GCD recursion theorem)

For any numbers a and b with $b>0$:

$$\mathrm{GCD}(a,b) = \mathrm{GCD}(b, a \bmod b).$$

Algorithm: Euclid

Input: Two integers a and b with $b \geq 0$

Output: GCD(a,b)

if $b = 0$

then return a

else return Euclid(b, a mod b)

Multiplicative inverse

Algorithm: extended-Euclid

Input: Two integers a and b with $b \geq 0$

Output: GCD(a,b) and two integers x and y with

$xa + yb = \text{GCD}(a,b)$

if $b = 0$ **then return** (a, 1, 0);

(d, x', y') := extended-Euclid(b, a mod b);

$x := y'$; $y := x' - \lfloor a/b \rfloor y'$;

return (d, x, y);

Application: $a = (p\text{-}1)(q\text{-}1)$, $b = e$

The algorithm returns numbers x and y with

$x(p\text{-}1)(q\text{-}1) + ye = \text{GCD}((p\text{-}1)(q\text{-}1),e) = 1$