03 – Randomization

Randomization

- Types of randomized algorithms
- Randomized Quicksort
- Randomized primality test
- Cryptography
- Verifying matrix multiplication

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

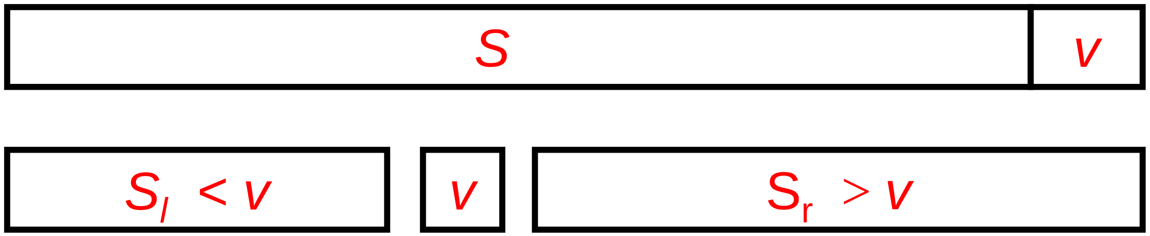
2. Quicksort

Input: List S of n distinct elements over a totally ordered universe.
Output: The elements of S in (ascending) sorted order.

Idea of Quicksort: Identify a splitter $v \in S$.

Determine set S_l of elements of S that are $< v$.
Determine set S_r of elements of S that are $> v$.
Sort S_l, S_r recursively.
Output sorted sequence of S_l, followed by v,
followed by sorted sequence S_r.

Quicksort



function Quick (S: sequence): sequence;

{returns the sorted sequence *S*}

begin

    if #S ≤ 1 then Quick:=*S*;

    else { choose splitter element v in S;

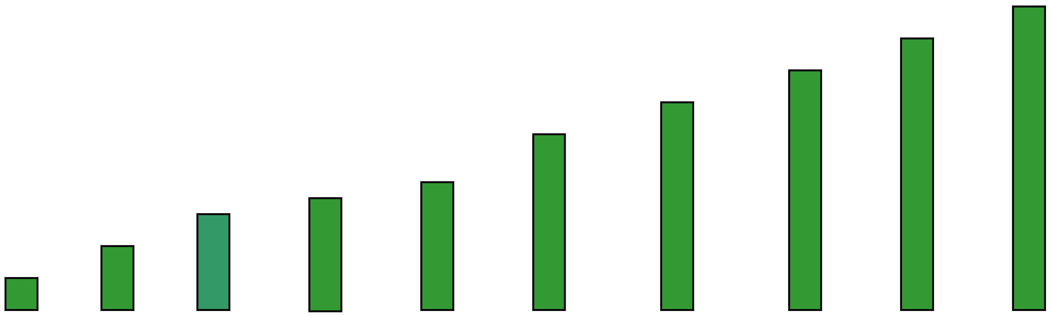
        partition *S* into S_l with elements $< v$,

        and S_r with elements $> v$;

        Quick:= $\boxed{\text{Quick}(S_l)\,|\,v\,|\,\text{Quick}(S_r)}$ }

end;

Worst-case input



n elements

Running time: $(n-1) + (n-2) + \ldots + 2 + 1 = n(n-1)/2$

Choice of the splitter element

Suppose that a splitter v with $|S_l| \leq n/2$ and $|S_r| \leq n/2$
can be found in cn step.

Then $T(n) \leq 2\,T(n/2) + an$, for some $a \geq c$, and $T(n) \leq an \log n$.

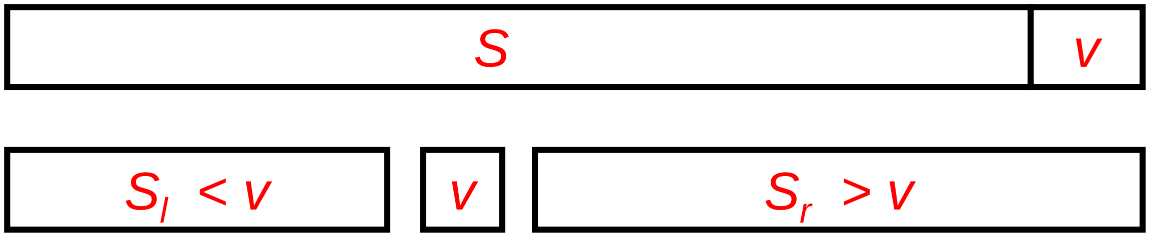
$T(k)$ = worst-case number of steps to sort k elements

Problem: Find splitter v with above property.

But: Running time of $O(n \log n)$ can be maintained if S_l, S_r have roughly equal size, i.e. $\frac{1}{4}|S| \leq |S_l|, |S_r| \leq \frac{3}{4}|S|$.

Thus randomly chosen splitter is „good“ with probability $\geq \frac{1}{2}$.

Randomized Quicksort



function RandQuick (*S*: sequence): sequence;

{returns the sorted sequence *S*}

begin

    if #*S* ≤ 1 then Quick:=*S*;

    else { choose splitter element *v* in *S* uniformly at random;

        partition *S* into S_l with elements $< v$,

        and S_r with elements $> v$;

        RandQuick:= $\boxed{\text{RandQuick}(S_l)}\boxed{v}\boxed{\text{RandQuick}(S_r)}$ }

end;

Analysis 1

n elements; let s_i be the i-th smallest element

With probability $1/n$, s_1 is the splitter element:
subproblems of sizes 0 and n-1

•
•
•

With probability $1/n$, s_k is the splitter element:
subproblems of sizes k-1 and n-k

•
•
•

With probability $1/n$, s_n is the splitter element:
subproblems of sizes n-1 and 0

Analysis 1

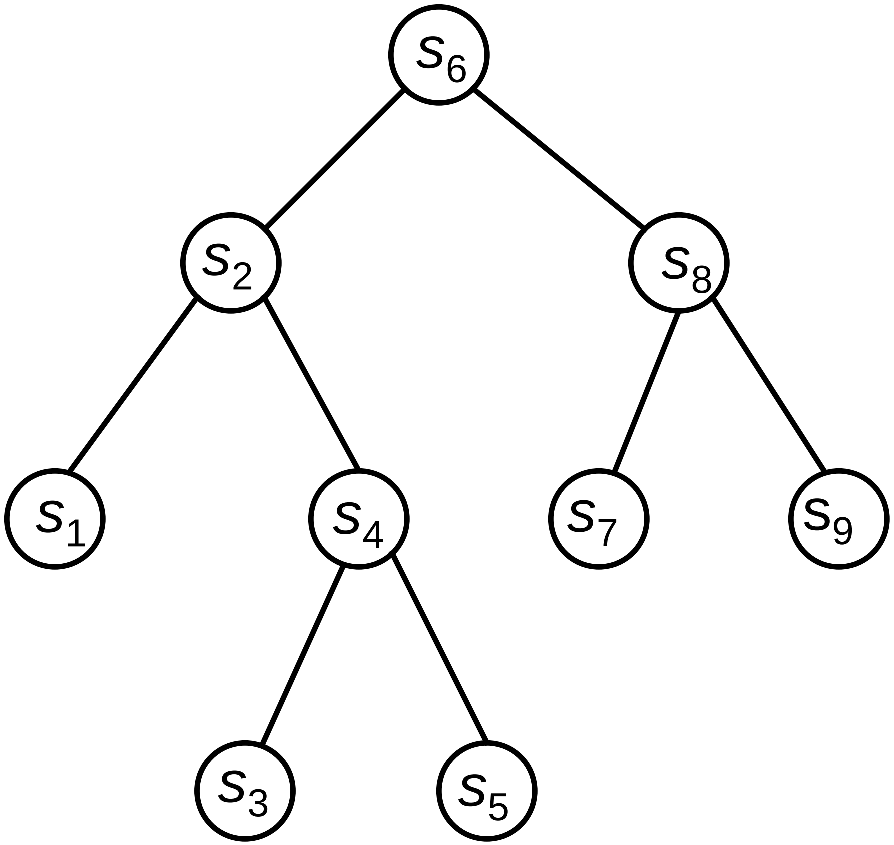
Expected running time:

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

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

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

Analysis 2: Representation of QS as a tree



Analysis 2: expected #comparisons

Running time is linear in the number of element 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

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

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. Hence $p_{ij} = 2 / (j-i+1)$

Analysis 2

Expected number of comparisons:

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

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

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

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

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

3. 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** $\lfloor \sqrt{n}/2 \rfloor$ **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: If algorithm always returns "prime", then with probability at most p^k, n is composite.

Simple 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

```
Choose a in the range [2, n-1] uniformly at random;
Compute a^(n-1) mod n;
if a^(n-1) mod n = 1
    then n is probably prime
    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

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: $n = 35$ $\quad$ $6^2 \bmod 35 = 1$

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.

Fast exponentiation

Method for computing a^n:

Case 1: [n is even]

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

Case 2: [n is odd]

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

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

Fast exponentiation

Example:

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

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

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

$a^{7} \quad = (a^{3})^2 * a$

$a^{3} \quad = (a)^2 * a$

Fast exponentiation

```
boolean isProbablyPrime;

function 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 then return 1;
    x := power(a, p div 2, n);
    result := x * x mod n;
    /* check if x^2 mod n = 1 and x ≠ 1, n-1 */
    if result = 1 and x ≠ 1 and x ≠ n –1 then isProbablyPrime := false;
    if p mod 2 = 1 then result := a * result mod n;
    return result;
}
```

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

Miller Rabin primality test

```
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 or !isProbablyPrime then
        return false;
   else return true;

}
```

Miller Rabin primality test

Theorem:

If n is composite, then there are at most

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

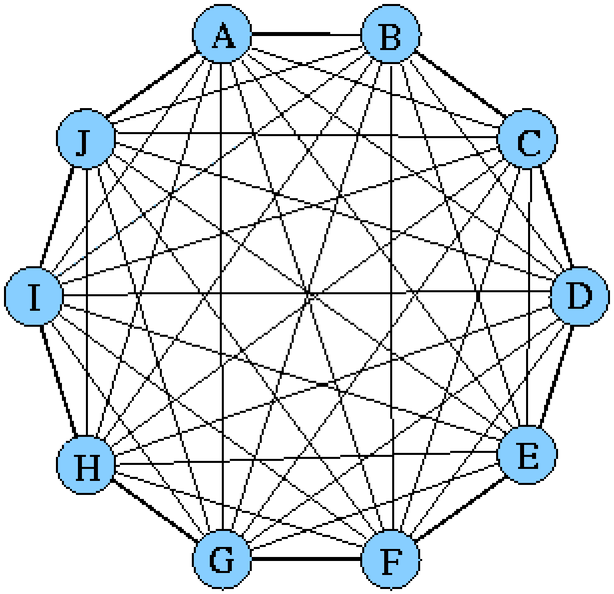
numbers $0 < a < n$ for which the algorithm primeTest fails.

Public-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.

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 secret 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

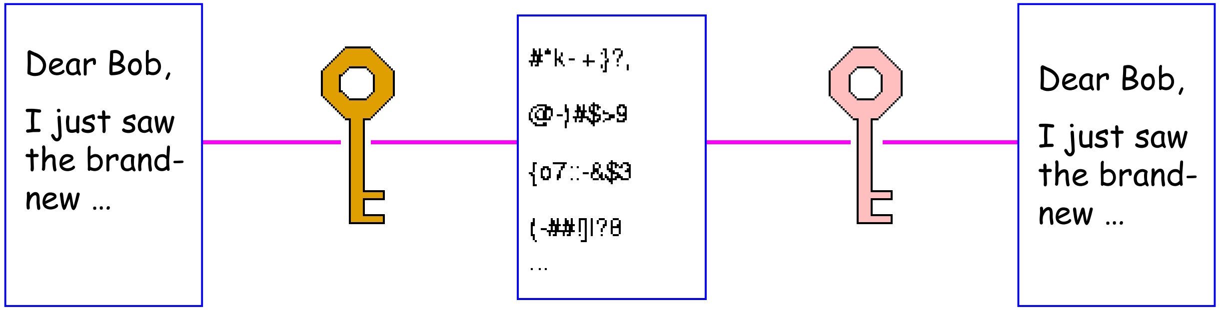
$$P_A(\ ), S_A(\ ) : D \xrightarrow{1-1} D$$

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 secret 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 her secret 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 secret keys:

1. Select at random two large primes p and q of l+1 bits ($l > 2000$).

2. Compute $n = pq$.

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

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

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

RSA cryptosystem

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

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

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

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

Recovering a message

To show: $S_A(P_A(M)) = P_A(S_A(M)) = M^{ed} \bmod n = M$, for any $0 \leq M < 2^{2l}$.

Theorem: (Fermat‘s little theorem)
If p is prime, then for any integer a that is not divisible by p,

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

Since $d \cdot e \equiv 1 \bmod (p-1)(q-1)$ there holds $ed = 1+k(p-1)(q-1)$, for some integer k.

Suppose that $M \bmod p \neq 0$. Then by Fermat’s little theorem, $M^{p-1} \bmod p = 1$ and thus $M^{k(p-1)(q-1)} \bmod p = 1$.

Hence $M^{ed} \bmod p = M^{1+k(p-1)(q-1)} \bmod p = M \bmod p$, and $M^{ed} - M = l_1 p$, for some integer l_1.

If $M \bmod p = 0$, then again $M^{ed} - M = l_2 p$, for some integer l_2.

Recovering a message

In any case, for any M, $M^{ed} - M = l \cdot p$, for some integer l.

Similarly, for any M, $M^{ed} - M = l' \cdot q$, for some integer l'.

Since p and q are prime numbers, $M^{ed} - M = l^* pq$, for some integer l^*.

We conclude that, for any M, there holds $M^{ed} \bmod n = M$.

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

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

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: $a = (p-1)(q-1)$, $b = e$

The algorithm returns numbers x and y with

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

5. Verifying matrix multiplication

Problem: Three $n \times n$ matrices A, B and C. Verify whether or not $AB=C$.

Simple solution: Multiply A, B and compare to C.

O(n^3) multiplications/operations, can be reduced to roughly O($n^{2.37}$).

Goal: Design fast verification algorithm that may err with a certain probability.

Verifying matrix multiplication

Algorithm: Choose $\vec{r} = (r_1, \ldots, r_n) \in \{0,1\}^n$ uniformly at random.
Compute $AB\vec{r}$ by first computing $B\vec{r}$ and then $A(B\vec{r})$.
Then compute $C\vec{r}$.

If $A(B\vec{r}) \neq C\vec{r}$, then return $AB \neq C$. Otherwise return $AB = C$.

Running time: O(n^2)

Theorem: If $AB \neq C$ and if $\vec{r}$ is chosen uniformly at random from $\{0,1\}^n$, then $\Pr[AB\vec{r} = C\vec{r}] \leq ½$.

We next prove this theorem.

Analysis

Law of Total Probability: Let Ω be a probability space and $A_1,\dots,A_n$ be mutually disjoint events. Let B be an event with $B \subseteq \bigcup_{i=1}^{n} A_i$. Then

$$\Pr[B] = \sum_{i=1}^{n} \Pr[B \cap A_i] = \sum_{i=1}^{n} \Pr[B \mid A_i] \Pr[A_i].$$

By assumption $AB \neq C$. Hence $D \coloneqq AB - C \neq 0$ and the matrix D contains at least one non-zero entry $d_{ij} \neq 0$.

On the other hand, $AB\vec{r} = C\vec{r}$ translates to $D\vec{r} = 0$.

Let $P = D\vec{r} = (p_1, \dots, p_n)^T$.

There holds $p_i = \sum_{k=1}^{n} d_{ik} r_k = d_{ij} r_j + y$, for some constant y.

Analysis

Hence

$\Pr[P = 0]$

$$\le \Pr[p_i = 0] = \Pr[p_i = 0 \mid y = 0] \cdot \Pr[y = 0] + \Pr[p_i = 0 \mid y \neq 0] \cdot \Pr[y \neq 0].$$

There holds:

$\Pr[p_i=0 \mid y=0] = \Pr[r_i=0] = ½$

$\Pr[p_i=0 \mid y \neq 0] = \Pr[r_i=1 \wedge d_{ij} = -y] \le \Pr[r_i=1] = ½.$

We conclude

$$\Pr[P = 0] \le \Pr[p_i = 0] \le ½ \cdot \Pr[y = 0] + ½ \cdot \Pr[y \neq 0]$$
$$= ½ \cdot \Pr[y = 0] + ½ \cdot (1 - \Pr[y = 0]) = ½.$$

Analysis

Repeating the algorithm k times reduces the error probability to $1/2^k$, using a running time of $O(kn^2)$.

For k=100, the error probability is upper bounded by $1/2^k$, while the running time is still $O(n^2)$.