I have to formulate the main notions of the theory of algorithms.

1. Turing Machines.

Definition. Turing Machine M (usually called for brevity TM) is uniquely defined by a triple $\langle S, t\, A, Q, q_0, \delta \rangle$, where S is a finite alphabet; $A \mu S$ an external alphabet; $t \in SnA$, an empty symbol.

Q is a finite set of control states.

$$\delta : Q \times S \qquad Q \times S \times \{-1, 0, 1\}$$

(means that our function may be not everywhere defined). Intuitively the computation on TM may be represented as following:

Finite State Control
with the set of control states Q

r Read/write head

S_0	S_1	¢	¢	¢	¢	¢	¢	S_p				

1. At the beginning the control state is q_0.

2. If at a moment t the head reaches the symbol S_p and the control state is q and

$$\delta(q, s_p) = (q', S, \Delta p)$$

The head:

1. replace s_p by s in the square p;

2. replace the control state q by q'

3. moves to the square $p + \Delta p$.

If $p + \Delta p < 0$ or $\delta(S_p q)$ is not defined, thus the TM halts.

Let us give the formal definition of computations by TM M.

A state λ of M is a triple

$$\lambda = \langle \sigma; p; q \rangle,$$

where σ is an infinite sequence of elements of S (σ_n - the nth letter of σ) (infinite word in the alphabet S), $p \in \mathbb{N}$, $q \in Q$. Let $\alpha \in A^*$ (a finite word in the alphabet A). The computation $M(\alpha)$ is the sequence of the states (finite or infinite): $\lambda(t) = (\sigma(t), p(t), q(t))$

$$\begin{aligned} \lambda(0) &= \langle \alpha; 0, q_0 \rangle \\ \lambda(t+1) &= \langle \sigma(t+1), p(t) + \Delta p, q' \rangle, \end{aligned}$$

where

$$\sigma_n(t+1) = \begin{cases} \sigma_n(t) & n \neq p(t) \\ S & n = p(t) \end{cases}$$

and

$$\sigma\left(\sigma_{p(t)}(t), q(t)\right) = \langle q, s, \ \Delta p \rangle$$

If $p + \Delta p < 0$ at $\delta\left(\sigma_{p(t)}(t), q(t)\right)$ is not defined then the computation $M(\alpha)$ halts. $\lambda(t)$ is the final state of $M(\alpha)$ and we write $M(\alpha)$ # (belts).

The final moment of computation $M(\alpha)$ will be denoted by $t(\alpha)$ (or $TM(\alpha)$)

It is easy to see that for any t the infinite word is $\sigma(t)$ contains only finitely many symbols not equal to t. If $M(\alpha)$ # then

$$s(\alpha) = \max \{n | \exists t \ \cdot \ t(\alpha)\sigma_n(t) \neq t\}.$$

$s(\alpha)$ is the size of memory, necessary for the computation $M(\alpha)$; sometimes we'll write $s_M(\alpha)$.

If $\sigma \in S^*$ let us denote by $P_A(\sigma)$ the word that is obtained from σ by erasing all letters from $S \cap A$. For example if s with subindeces denotes the letters of $S \cap A$ and a with subindeces denotes the letters of A then

$$P_A(s_1 a_1 a_2 s_2 s_b a_1 s a_2 a_2) = a_1 a_2 a_1 a_2 a_2$$

We say that the Turing Machine M computes the function $\varphi_M : A^* \quad A^*$ if

1. $\alpha \in \mathrm{dom}\,\varphi_M$, $M(\alpha)$ #

2. if $\alpha \in \mathrm{dom}\,\varphi_M$ then $\varphi_M(\alpha) = P_A(\sigma(t(\alpha))$;

We say that a function $\varphi : A^* \to {}^*A$ if there exists a Touring Machine M such that $\varphi = \varphi_M$. If we assume that our input alphabet is of the form $A \cup \{\#\}$ then we may define in a similar way the function $\varphi_{M;n} : (A^*)^n \to A^*$ $\langle\alpha_1, \ldots, \alpha_n\rangle \in \mathrm{dom}\,\varphi_{M;n}$ iff $M(\alpha_1\#\alpha_2\#\ldots\#\alpha_n\#)$ # $M(\alpha)$ # iff $\exists\alpha_1 \ldots \alpha_n \in A^*$ $\alpha = \alpha_1\#\alpha_2\#\ldots\#\alpha_n\#P_M(\sigma(\alpha)) = \varphi_{M;n}(\alpha_1, \ldots, \alpha_n)$.

Church-Turing Thesis. Any computable in some intuitive sense function is TM-computable.

In other words, Ch. T, Thesis means that any algorithm can be simulated on Turing Machines. In what follows, when we consider computable functions $\varphi : \mathbb{N}^n \to \mathbb{N}$ we assume taht our input alphabet $A = \{0, 1\} \cup \{\#\}$ and the natural numbers are represented in the binary form. Notation: $\mathbb{B} = \{0, 1\}$.

It is easy to see that the set of all Turing Machines can be effectively enumerated. It means that there exists a bijection ν between the set of all TM and N such that given a description of TM M by definition 1 we can effectively in finitely many steps compute $\nu(M)$, and given any natural number n we can effectively in finitely many steps find the description of the TM M such that $n = \nu(M)$.

Assuming that our alphabet A contains 0,1 we may construct the function $\Psi : (A^*)^2 \to A^*$ such that $(\alpha, \beta) \in \mathrm{dom}\,\varphi$ iff $\beta \in \mathrm{dom}\,\varphi_{\circ^{-1}(\circledR)}$ and in this case $\Psi(\alpha, \beta) = \varphi_{\circ^{-1}(\circledR)}(\beta)$. The function Ψ is called universal for all TM-computable function $\varphi : A^* \to A^*$.

The function $\Psi_n : (A^*)^{n+1} \to A^*$ universal for all TM computable functions $\varphi : (A^*)^n \to A^*$ is defined in the similar way. It is easy to see that the universal function Ψ is computable in intuitive sense. Indeed, given $\alpha, \beta \in A^*$ we find effectively $\nu^{-1}(\alpha) = M$ and start the calculation $M(\beta)$. It halts iff $(\alpha, \beta) \in \mathrm{dom}\,\Psi$ and $\Psi(\alpha, \beta) = \sigma(t_M(\beta))$. All this is done effectively. According to Ch.-T Thesis Ψ is TM-computable. A TM that computes Ψ is called a universal Turing Machine.

We say that a predicate P on A^* is decidable if its characteristic function $K_p : A^* \to \{0, 1\}$ is

TM computable. Since K_p is everywhere defined P is decidable if there exists an algorithm that gives for any $\alpha \in A^*$ after finitely many steps whether $P(\alpha) = T$ or $P(\alpha) = F$. Indeed since K_p is everywhere defined and TM-computable then a Turing Machine M that computes K_p halts for any initial state α.

Let us consider an example of undecidable predicates

$$P(\alpha) = T \quad \text{iff} \quad \nu^{-1}(\alpha)(\alpha) \#$$

If P is decidable then $K_p(\alpha)$ is TM-computable. Consider the function:

$$\varphi(\alpha) = \begin{cases} 1 & \text{if } K_p(\alpha) = 0 \\ 0 & \text{not defined if } K_p(\alpha) = 1 \end{cases}$$

According to C.-T. Thesis $\varphi(\alpha)$ is also computable. Let M_0 be a TM that computes φ, i.e. $\varphi = \varphi_{M_0}$ $\varphi(\alpha) = 1$ iff $\nu^{-1}(\alpha)(\alpha)$ ", $\alpha \notin \operatorname{dom}(\varphi_o^{-1}(\alpha))$. $\varphi(\alpha)$ is not defined if $\nu^{-1}(\alpha)(\alpha) \#$, $\alpha \in \operatorname{dom}\varphi_{o-1(\alpha)}$. Let $\alpha_0 = \nu(M_0)$. Then $\varphi(\alpha_0) = 1$ iff $M_0(\alpha_0)$ ", $\in \operatorname{dom}\varphi_{M_0}$, $\alpha_0 \in \operatorname{dom}\varphi$—impossible. $\varphi(\alpha_0)$—is defined iff $M_0(\alpha_0) \# \alpha_0 \in \operatorname{dom}\varphi_{M_0}$, $\alpha_0 \in \operatorname{dom}\varphi$ contradiction. The undecidability of P implies the undecidability of the following problem which is called the Halt Problem for TM:

Given TM M and an $\alpha \in A^*$ to find out whether $M(\alpha) \#$ or $M(\alpha)$ ".

If there exists an algorithm that answers this question for any α infinitely many steps then applying this algorithm to $\nu(M)$ we obtain the answer to the question whether $P(\alpha(M)) = T$ or $P(M) = F$.

Thus the Halt Problem for TM is undecidable.

To prove that any problem P is undecidable it is enough to show that if there exists an algorithm that solves P then there exists an algorithm that solves the Halt Problem. The undecidability of all well-known problems such as the 10th Hilbert's problem and the word in the group theory was proved in this way.

In what follows we'll consider only decidable problems. We'll be interested in the complexity of computation. If $\alpha \in A^*$ then $|\alpha|$ is the length of α.

$$\begin{aligned} T_M(n) &= \max\{t_M(\alpha) \mid |\alpha| \cdot n\} \\ S_M(n) &= \max\{s_M(\alpha) \mid |\alpha| \cdot n\} \end{aligned}$$

We say that a function $f : \mathbb{N} \to \mathbb{N}$ is of a polynomial growth if $f(n) \cdot C_n^d$ for all big enough n. Notation: $f(n) = \text{poly}(n)$.

Definition.

1. We say that a function $\varphi : A^* \to A^*$ is computable in polynomial time if there exists a TM M that computes φ and $T_M(n) = \text{poly}(n)$. We denote by P the class of all predicates on $\mathbb{B}^*$, which characteristic functions are computable in polynomial time.

2. We say that a function $\varphi : A^* \to A^*$ is computable in polynomial memory if there exists a TM M, computing φ, such that $S_M(n) = \text{poly}(n)$.

The class of all predicates in $\mathbb{B}^*$ such that their characteristic function are computable in polynomial memory is denoted by PSPACE.

Schemes.

Recall that any system F of Boolean function. F the functions of $f : \mathbb{B}^n \to \mathbb{B}$ is complete if any Boolean function may be represented as a composition of some functions of the system F. An example of complete system is the system

$$\{\_, \wedge, :\}$$

If $\sigma \in \{0, 1\}$ and x is a variable that range over them $x^{\sigma} = \begin{cases} x, & \sigma = 1 \\ :x & \sigma = 0 \end{cases}$. Then

$$f(x_1, \ldots, x_n) = \_ x_1^{\sigma_1} \wedge \ldots \wedge x_n^{\sigma_n}$$

$$f(\sigma_1 \ldots \sigma_n) = 1$$

One more example of a complete system is a system $\{|1\}$, where $x|y = :x \wedge :y$. Indeed $x|x = :x$, $x \wedge y = :x|:y = (x|x)|(y|y)$, $x \_ y = :(:x \wedge :y) = (x|y)|(x|y)$.

One of the algorithms of computation of a function $f : \mathbb{B}^n \to \mathbb{B}^m$ is a scheme in a base F. A scheme S uses input variables $x_1, \ldots, x_m$ and also some additional variables $y_1, \ldots, y_s$. A scheme S is determined by a sequence of assignments $Y_1, \ldots, Y_s$, where each of the assignments Y_i is of the

form $y_i := f_j(u_{k_r}, \ldots, u_{k_4})$ where $f_j \in F$ and each u_{k_p} is either one of the input variables or one of the additional variables y_t where $t < i$. The result of the computation is the triple of values of last m additional variables $y_{s-m+1}, \ldots Y_s$. Any scheme can be represented by an acyclic oriented graph. The vertexes of input degree 0 are marked by input variables. The other vertexes are marked by functions of F and the edges are marked by variables. The vertexes of output degree 0 are marked by output variables.

Example 1. $F = \{1\}$ x_i/x_2

$$\begin{aligned} y_i &:= x_1 | x_2 \\ y_2 &:= y_1 | y_1 \end{aligned}$$

Example 2. Let us consider the single digit adder. When we add to numbers written in binary digit form in each digit we add the corresponding bites of summand and one bite that we kept in mind from the addition of previous digit. We also have to outputs: the sum and the bit that we keep in mind for next digit.

So we have three inputs x_1, x_2, x_3 and two outputs, let us denote them at first u, v (sum and ??). We'll write $\overline{X}$ instead of $\overline{x}$ and $x \cdot y$ instead of $x \wedge y$.

$$\begin{aligned} u &= \overline{x}_1\overline{x}_2x_3 \vee \overline{x}_1x_2\overline{x}_3 \vee x_1\overline{x}_2\overline{x}_3 \vee \overline{x}_1\overline{x}_2\overline{x}_3 \\ v &= \overline{x}_1x_2x_3 \vee x_1\overline{x}_2x_3 \vee x_1x_2\overline{x}_3 \vee x_1x_2x_3 \end{aligned}$$

A scheme in the base $\{:, \vee, \wedge\}$ will be rather complicated. Let us introduce one more function $x \oplus y$—addition module 2.

x_1	x_2	x_3	u	v
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

x	y	$x+y$
0	0	0
0	1	1
1	0	1
1	1	0

$x \copyright y = \overline{x}y \_ x\overline{y}$

Now it is easy to see that

$u = x_1 \copyright x_2 \copyright x_3$ (addition is associative)

$v = (x_1 \copyright x_2)x_3 \_ x_1x_2(x_3 \_ \overline{x}_3 = (x_1 \copyright x_2)x_3 \_ x_1 \diamondsuit x_2$

$y_1 := x_1 \copyright x_2$

$y_2 := y_1 \diamondsuit x_3$

$y_3 := x_1 \diamondsuit x_2$

$y_4 = y_1 \copyright x_3$

$y_5 = y_2 \_ y_3$

Better to draw this scheme in the following way:

It is easy to see that there may exist many schemes that compute a function $f : \mathbb{B}^n \to \mathbb{B}^m$. In the previous example we can construct, for example, also the following scheme:

$$y_1 := x_1 \copyright x_2; y_2 := (x_2 \copyright x_3); y_3 := x_1 \diamondsuit x_2; y_4 := y_1 \diamondsuit x_3; y_5 = x_1 \copyright y_2; y_6 := y_4 \_ y_3$$

The size of a scheme is the number of assignments in this scheme. The scheme complexity of a function $f : \mathbb{B}^n \to \mathbb{B}^m$ in a base F is the minimum size of a scheme in a base F that computes f. The computational complexity of f in a base F will be denoted by $C_F(f)$. The choice of F is not very important—it only multiplies the computation complexity by some constant, so in what follows index F will be omitted.

Let $\varphi : \mathbb{B}^* \to \mathbb{B}$ be a predicate on $\mathbb{B}^*$ (we identify a predicate and its characteristic function). We have $\mathbb{B}^* = \bigcup_{n=0}^{\infty} \mathbb{B}^n$, where $\mathbb{B}^0 = \{\Lambda\}$, Λ—is the empty word—the word of the length 0 (not the same as $\{t\}$!!). Let $\varphi_n = \varphi|\mathbb{B}^n$.

Definition. We say that a predicate φ is in the class P/poly if $C(f_n) = \text{poly}(n)$.

Theorem. $P \subseteq P/\mathrm{poly}$.

Proof. If a computation on an MT M is completed in a polynomial time then the size of memory that it uses is also bounded by a polynomial ($P \subseteq$ PSPACE). And thus the computation starting from the input word x of the length n can be represented by the computational table of the size $T \times S$, where $T = \mathrm{poly}\,(n)$, $S = \mathrm{poly}\,(n)$.

The row number of j determines the state of M after j steps of computation. The symbols $\Gamma_{j;k} = \langle S, (\alpha, Q)\rangle$, $\Gamma_{j;k} = \langle \sigma_k(j), \mu\rangle$, where $\mu \begin{cases} q(j) & \text{if } p(j) = k \\ \phi & \text{otherwise} \end{cases}$.

If for some input the computations halts at a moment $T' < T$ then we assume that all rows of number $j > T'$ are the same as the row of number T'. Now a scheme that computes the values of our predicate on the words of the length ℓ can be constructed in the following way. The state of each square of our table can be encoded by a number of Boolean variables that does not depend on n. More exactly it is enough $x_1, \ldots x_L$, where L is the least number such that $|S| \cdot (|Q| + 1) < 2^L$. It is easy to see by the definition of computation that the state Γ of any square in the row is $j+1$ is uniquely determined by the state of the square with the same number in the previous row and its left and right neighbors. Thus each of variables that encode the state Γ is a function of variables that encode the state $\Gamma', \Gamma', \Gamma'_\Gamma$. The functions can be computed by the schemes of finite (independent of n) size. The union of these schemes determines the scheme that computes φ for any word x of the length n. □

It is easy to see that P/poly is larger than P. Indeed consider any predicate f on $\mathbb{N}$. It determines the predicate $\varphi(f)$ on $\mathbb{B}^*$ such that $\varphi(f)(x) = f\,(|x|)$. So $\varphi(f)|\mathbb{B}^n$ is either identically

0 or identically 1. The scheme complexity of such function is bounded by a constant. So such predicates are in the class $P\text{poly}\,(a)$. But among them there are even undecidable predicates. The following theorem is almost obvious:

Theorem. A predicate f is in P iff

1. $f \in P/\text{poly}$
2. there exists a TM M such that constructs for any n a scheme computing f_n during a polynomial of n time.

The Class NP

Definition. The predicate L is in the class NP if $L(x) = \exists y\,(|y| < q\,(|x|) \wedge R(x,y))$, where $q(n)$ is a polynomial of n, and the predicate $R(x,y)$ is in the class P.

Intuitively, $L(x) = T$ iff there exists such a group y that allows us to verify that $L(x) = T$ in polynomial time.

For example. We say that a sequence $v_1, \ldots v_m$ of the vertices of a graph is a cycle if (v_i, v_{i+1}) is an edge as well as $(v_m v_1)$. A cycle is simple if none of the vertices is repeated. A simple cycle that meets all the vertices is called Hamiltonian. Consider the following problem: Given a graph determine if it has the *Hamiltonian cycle*.

If x is a binary coding of a graph and y is a binary code of Hamiltonian cycle it can be taken such that $|y| < |x|$ then $R(x,y) = T$ iff y is the Hamiltonian cycle in x is polynomial.

2) Consider a problem SAT that was introduced on the first lecture. Given a formula $F(p_1, \ldots p_n)$ to determine if there exists an n-triple $\langle\sigma_1 \ldots \sigma_n\rangle$ such that $F((\sigma_1 \ldots \sigma_n) = 1$.

Once again if x is the binary code of $F(p_1 \ldots p_n)$ and y is an n-triple $\sigma_1 \ldots \sigma_n$ then we may calculate $F(\sigma_1, \ldots, \sigma_n)$ in a polynomial time of the length of $|x|\,(|y| \cdot |x|)$. Thus this problem is also NP.

It is easy to see that $P \subseteq$ NP. Indeed if $L(x)$ is a polynomial predicate we may take for $R(x,y)$ $y = y \wedge L(x)$. The problem to prove (or disapprove) that $P \neq$ NP is open for few dozens of years already!

Definition. We say that a predicate L_1 is reduced to a predicate L_2 ($L_1 \alpha L_2$) is there exist. A polynomially computable function f, such $\forall x_1 L_1(x) \Leftrightarrow L_2(f(x))$.

Lemma.

1. $L_2 \in P \Rightarrow L_1(P)$
2. $L_2 \in$ NP $\Rightarrow L_1 \in$ NP.

Proof. Obvious.

Definition. We say that a predicate L is NP-complete if any NP-predicate can be reduced to L.

Theorem. (Cook, Levin). SAT is NP complete. Let $L(x) = \mathsf{9}y\,(\mathsf{j}y\mathsf{j} < q\,(\mathsf{j}x\mathsf{j})\ \mathsf{\wedge}\ R(x,y))$. $R(x,y)$ is polynomial. Consider the number of boolean variables that is enough to code the input $x\#y$ it is $\mathsf{j}x\mathsf{j} + 1 + q\,(\mathsf{j}x\mathsf{j}) + 1$. And consider a logical scheme for R under the fixed x.

$$y_1 = y_1(u_{i_1}, \ldots u_{i_n}) \ldots y_{\mathsf{s}} = Y_{\mathsf{s}}(u_{i_1} \ldots u_{\mathsf{s}})$$

for variables that code x we substite their concrete values.

$$y_{\mathsf{x}}(u) = (y_1\ \mathsf{\$}\ Y_{\wedge})\ \mathsf{\wedge} \ldots (y\ \mathsf{\$}\ Y_{\mathsf{s}})$$

$$a\ \mathsf{\$}\ b\ \mathsf{,}\ \ a\,b\,v\,\overline{a}\,\overline{b}$$

$$\varphi_{\mathsf{x}}(y,u) = 1 \quad \text{for some } y,u \quad \text{iff} \quad L(x) = 1.$$

Lemma. If L_1 is NP complete and $L_1 \alpha L_2$ and L_2 $\mathsf{2}$ NP then L_2 is NP complete.

Corollary. If SAT is in P then $P = \text{NP}$.

Probability Algorithms—Algorithms for Primary Testing

It is easy to see that the predicate x is a *composite number* is in NP. More exactly we consider the predicate x is the binary representation of a composite number but we'll usually speak about the numbers themselves and consider the time of computation as the function of $\log x$ (the length of x is proportional to $\log x$).

Now "x is composite" $\leftrightarrow \exists y\,(y|x \wedge y \neq 1 \wedge y \neq x)$. Obviously $\log y < \log x$ and $y|x$ is a polynomial predicate—we may use the usual algorithm for finding the quotient and remainder that work for polynomial time.

More difficult is the following.

Theorem 1. The predicate "x is prime" is in NP. The proof of this theorem will be based on the following.

Theorem 2. Let $n > 1$. Assume that there exists a primative root of n i.e an integer as such that

1. $a^{n-1} \equiv 1 (\mathrm{mod}\, n)$
2. $a^{\frac{n-1}{p}} \neq 1 (\mathrm{mod}\, n)$ for each prime factor of $p - 1$.

Then n is prime.

Proof. Since $a^{n-1} \equiv 1 (n) \mathrm{ord}_n a | n - 1$. We'll show that $\mathrm{ord}_n a = n - 1$. Suppose $\mathrm{ord}_n(a) \neq n - 1$ then $\exists k > 1 (n - 1) = k \mathrm{ord}_n(a)$. If p is a prime factor of k then

$$a^{\frac{n-1}{p}} \equiv a^{\frac{k \mathrm{ord}_n(a)}{p}} \equiv \left[a^{\mathrm{ord}_n(a)}\right]^{\frac{k}{p}} \equiv 1(n)$$

Contradiction. By Euler's theorem.

$$a^{\varphi(n)} \equiv 1(n)$$

So $(n - 1) = \mathrm{ord}_n a | \varphi(n)$ thus $n - 1 \geq \varphi(n) \geq n - 1$ by definition of $\varphi(n)$), and $\varphi(n) = n - 1$.

$\varphi(n) = \left(p_1^{\alpha_1} - p_1^{\alpha_1 - 1}\right) \ldots (p_s^{\alpha_2} - p_s^{\alpha_s - 1})$ where $n = p_1^{\alpha_1} \ldots p_s^{\alpha_1}$.

Obviously this is equal to $n - 1$ only if $s = 1$, $\alpha = 1$. Now we can prove theorem 1. We have to prove that if n is prime then this fact can be proved in polynomial time. It is not the same as to find out whether n is priem or not in polynomial time.

If p is prime let a be a primative root $\bmod n$, and $p_1, \ldots p_s$, $\alpha_1 \ldots \alpha_s$ are such that $(p - 1) = p_1^{\alpha_1} \ldots p_s^{\alpha_s}$(¤). Now we have to verify that $a^{n-1} \equiv 1 (\bmod n)$, the equality (¤) and each of inequalities $a^{\frac{n-1}{p_j}} \neq 1 (\bmod n)$. Let $n = \log p$. Then $S = 0(n)$. The computation of f^2 requires no more than $0(\log q)$ multiplications. (Indeed if $q = 2^{t_1} + 2^{t_2} + \ldots + 2^{t_m}$ where $t_1 > z_2 > \ldots > t_m >$????. It is enough to use t_1 to obtain all $f^{2^{t_m}}, \ldots, f^t$ and no more than t_1 multiplications to obtain $q(m < t_m)$. Now to verify (¤) we need no more than $0(n^2)$ multiplications (each $\alpha_i < n$ thus for each $p_i^{\alpha_i}$ we need $0(n)$-multiplications). Each multiplication requires no more than n^2 operations. Thus we held up to now $0(n^4)$ operations. For a^{n-1} and $a^{\frac{n-1}{p}}$ we need even less operations. The problem is that we have to verify the primarity of all p; and thus to repeat all that was before for each of them (and then four prime decomposition of $p_j - 1$, etc.. How many times have we to repeat all stuff? Notice that $|x - y| \leq |x| + |y|$, $|x| + |y| - 1$. The product of all primes at the kth step of our recursion is less than p. Then

$$|p_1| + |p_2| + \ldots + |p_s| - s \cdot |p_1 p_2 \ldots p_0| < |p|$$

thus

$$|p_1| + |p_2| + \ldots |p_2| \cdot 2|p|.$$

Indeed if

$$|p_1| + |p_2| + \ldots + |p_s| \geq 2|p|$$

then

$$|p_1| + |p_2| + \ldots + |p_s| - s \geq |p| + |p| - s \geq |p|$$

since $|p| - s \geq 0$.

The maximum number for prime testing of the kth step of recursion is at least twice less than maximum number for the prime testing on the $(k - 1)$-th step. Indeed, it is a nontrivial factor of

a number obtained on $(k - 1)$th step. Thus the maximum number of the steps of recursion is nor more than n, and thus the sum of all lengths of numbers for prime testing is $0(n^2)$. So we need no more than $0(n^8)$ operations. (Indeed it is enough $0(n^2)$ operations.

The *Probability Turing Machines* (PTM) are defined in the same way as usual Turing Machine only for some (q, s) $\delta(q, s)$ is a pair of triples $\langle q_i^1, s_i^1, \Delta p_i \rangle$ $(i = M, T)$ and we decide which of them to take by coin toss. If coin toss lands heads we take $\langle q_p^1, s_n^1 \Delta p_n \rangle$, is tails—then $\langle q_t^1, s_t^1, \Delta p_t^1 \rangle$. So at each step of computation we choose the next step with the probability $\frac{1}{2}$. It is easy to compute the probability of any output for a fixed input α.

Definition. A predicate L is in the class BPP if there exist a PTM M and a polynom $p(n)$ such that M with probability $1 - t_M(x) \cdot p(???)$ and $L(x) = 1 \Rightarrow M$ gives the answer "yes" with probability $\geq \frac{2}{3}$, $L(x) = 0 \Rightarrow M$ gives the answer "no" with probability $\geq \frac{2}{3}$. We can take any number greater than $\frac{1}{2}$. The class PBR will not change. For each such number we can get the right number with probability as close to 2 as desired. We have to take several machines and to start them from the same input simultaneously and take as the result—the opinion of the majority. If for any copy of BPT the probability of wrong answer $c < \frac{1}{2}$ then the probability of wrong answer after such vote of n machines will be

$$\sum_{\substack{S \subseteq \{1,\ldots n\} \\ |S| \leq \frac{n}{2}}} (1 - c)^{|S|} c^{n-|S|} = ((1 - c)c)^{\frac{n}{2}} \sum_{\substack{S \subseteq \{1,\ldots n\} \\ |S| \leq \frac{n}{2}}} \left(\frac{c}{1 - c}\right)^{\frac{n}{2}-|S|}$$

$$< \left(\sqrt{(1 - c)c}\right)^n 2^n = \lambda^n,$$

where

$$\lambda = 2\sqrt{c(1 - c)} < 1$$

(since $\frac{c}{1-c} < 1 - \sum_{\substack{S \subseteq \{1\ldots n\} \\ |S| \leq \frac{n}{2}}} \left(\frac{c}{1-c}\right)^{\frac{n}{2}-|S|} \cdot 2^n$.)

Theorem. A predicate L is in the class BPP iff there exists such polynom $q(n)$ and predicate

$R(x,y)$ 2 P such that:

$$L(x)=1 \quad) \quad \frac{\left|\left\{r\mathbf{j}|r|\le q(|x|)\wedge R(x;y)=1\right\}\right|}{\left|\left\{r\mathbf{j}|r|\le q(|\circledR|)\right\}\right|} \ , \ \frac{2}{3}$$

$$L(x)=0 \quad) \quad \frac{\left|\left\{r\mathbf{j}|r|\le q(|x|)\wedge R(x;y)=1\right\}\right|}{\left|\left\{r\mathbf{j}|r|\le q(|\circledR|)\right\}\right|} \ , \ \frac{1}{3}$$

Proof.

1. Let L 2 BPP with PMT M and polynoms $p(n)$. Let $q(n)=p(n)$ (the number of coin tosses is no more than all steps of computations).

   $R(x,r)=n$ if M gives at the input x the answer "yes" for the sequence of triples given in Γ (Γ is the sequence of h and t at those moments when we use the coin toss.

2. Those randomly the word Γ of the length $q\,(\mathbf{j}x\mathbf{j})$ and plug into our predicate for variable Γ.

Let us now consider probability polynomial algorithm for primary testing.

Step 1. Check if n is even, if so and $n=2$ then n ??? if not n is composit.

Step 2. Check if n is $a^{\mathbf{k}}$ for $k=2,3,\ldots[\log_2 n]$

Let us now consider probability polynomial algorithm for primary testing.

Step 1. Check if n is even; if so then if $n=2$ then n is prime, otherwise composite so n is odd.

Step 2. Check if n is $a^{\mathbf{k}}$ for $k=2,3\ldots[\log_2 n]$ (we discussed already that this can be done in polynomial time).

Step 3. Represent n ¡ 1 in the form $2^{\mathbf{k}},\ell$, where $k>0$ and ℓ-odd.

Step 4. Choose random a 2 $\mathbf{f}1,2,\ldots,n\mathbf{g}$

Step 5. Compute $a^{\grave{}},a^{2^{\grave{}}},\ldots,a^{n-1}(\mathrm{mod}\,a)$

Test 1. If a^{n-1} 6 $1(\mathrm{mod}\,n)$ then the answer is "n-composite";

Test 2. If $\mathbf{9}j\left(a^{2^{j}j}\ \mathbf{6}\ \mathbf{\S}1\right)\left(a^{2^{i+1}\grave{}}\ \acute{}\ 1(\mathrm{mod}\,n)\right)$ then n-composite, otherwise n-prime.

Theorem

1. If n is prime then we obtain the answer n is prime with probability 1 (always)

2. If n is composite then the answer n is compo. site will be obtained with probabilty $\geq \frac{1}{2}$.

Remark. Thus the probability of mistake is $\leq \frac{1}{2}$ and if we apply this algorithm twice then the probability of mistake will be $\leq \frac{1}{4}$.

If n is prime then $a^{n-1} \equiv 1(n)$—small Fermat theorem. If $a^{2^{i+1}} \equiv 1(\mathrm{mod}\, a)$ $\Rightarrow$ $(a^{2^j} - 1)(a^{2^j} + 1) \equiv 0(n)$ and since $a^{2^j} \neq \pm 1$ n is composite. This proves the first part of our theorem. Let us prove the second part.

Let $n = u \cdot v \equiv 1(2) \mathrm{ged}\,(u, v) = 1$. If we cannot represent n in such a way then we'll find that n is composite on the step 2. If $\gcd(a, n) > 1$ then test 1 shows us that n is composite. So we have to show that no less than $\frac{1}{2}$ of all numbers prime with n give by our algorithm the answer n is composite.

The group of reversible elements of a ring R is usually denoted by R^*

Denote $(\mathbb{Z}/u\mathbb{Z})^* = U$, $(\mathbb{Z}/v\mathbb{Z})^* = V$. Then the group $(\mathbb{Z}/n\mathbb{Z}) \cong U \times V$—Chinese Remainder Theorem!

Let us consider the subgroups

$$U^R = \left\{x^k | x \in U\right\}, \quad V^k = \left\{x^k | x \in V\right\}$$

We have $\frac{|U|}{|U^k|}$ and $\frac{|V|}{|V^k|}$ are integers. The map $x \to x^k$ is a homomorphism and so the cardinality of all sets $U_y = \left\{x | x^k = y\right\}$ is the same—does not depend on $y \in U^k$. Obviously $U^2 \subset U$ and thus we have the following inclusions:

$$\begin{array}{ll} U & \supset U^2 \supset \ldots \supset U^{n-1} \supset \{1\} \\ V & \supset V^2 \supset \ldots \supset V^{n-1} \supset \{1\} \end{array}$$

1. Let $U^{n-1} \neq \{1\}$ (or $V^{n-1} \neq \{1\}$) to pass the test 1 we have to obtain after taking $(n-1)$th power the point of remainders $(1, 1)$. Since any number in $U^{n-1} \neq \{1\}$ has the same cardinality of the inverse image we obtain $(1, 1)$ with probability no more than $\frac{1}{2}$ ($\frac{1}{2}$ in the case when U^{n-1} contains exactly 2 elements). And thus we obtain the answer "n-composite" after already the first test with probability $\geq \frac{1}{2}$.

2. Let $U^{n-1} = V^{n-1} = \{1\}$ since ℓ is odd $\Rightarrow 1 \in U^{\ell} \setminus V^{\ell}$, i.e., there $t = 2^s$, $0 \leq s < k$ $(n = 2^k \ell)$ such that $U^{\ell \cdot 2t} = V^{\ell \cdot 2t} = \{1\}$, but either $U^{\ell \cdot t} \neq \{1\}$ or $V^{\ell \cdot t} \neq \{1\}$. Let us show that in this case also with the probability $\geq \frac{1}{2}$ the second test of our algorithm will give the answer "n-composite". We have $a^{2t \cdot \ell} \equiv 1(n)$ and we have to understand what probability $a^{t\ell} \neq \pm 1(\mathrm{mod}\, n)$

   (a) $U^{t \cdot \ell} = \{1\}$ (or $V^{t\ell} = \{1\}$ the same). So this case $a^{\ell t} \neq -1(\mathrm{mod}\, n)$ because for (-1) we have to obtain the pair of remainders $(-1, -1)$. Now as in the case one with probability not less than $\frac{1}{2}$ we'll obtain the pair of remainders $(1, \alpha), \alpha \neq 1$ and thus $a^{+} \neq 1(\mathrm{mod}\, n)$.

   (b) $U^{t \cdot \ell} \neq \{1\}$ $V^{t \cdot \ell} \neq \{1\}$, so $|U^{t\ell}| = c \geq 2$ $|V^{t\ell}| = d \geq 2$. Then the probability to obtain $\{1, 1\}$ or $\{-1, -1\}$ is no more than $\frac{2}{cd} \leq \frac{1}{2}$