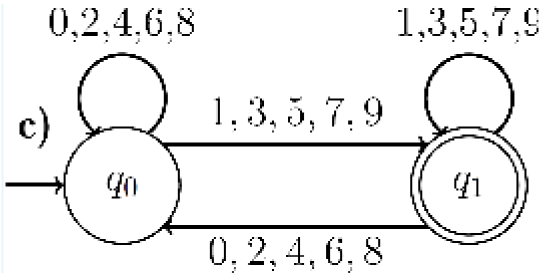
Exercise 1

1. **How many parameters (pairwise distances) are required to fully specify the TSP problem?**
   $n(n-1)/2$ which obeys $n^2/4 \leq n(n-1)/2 \leq n^2/2$ ('quadratic growth')

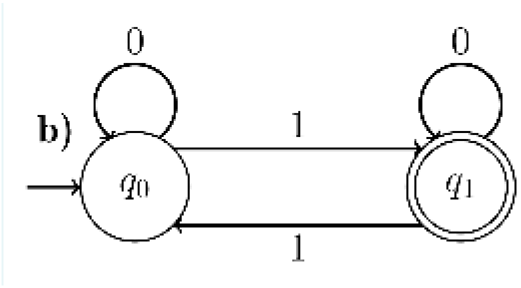
2. **What is the total number of inequivalent routes once n ≥ 3 (and we don't distinguish between taking routes forward or backwards)?**
   $(n-1)!/2$ which obeys $2^n/8 \leq (n-1)!/2 \leq n^n/2$ ('faster than exponential growth')

3. **The aim is to design a DFA that computes parity by processing decimal numbers instead of binary ones. Which of the following four state diagrams describes the correct functionality?**



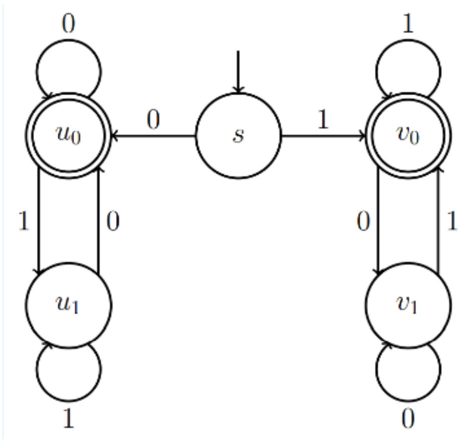
4. **Which of the following statements is correct? Wählen Sie eine Antwort:**
   runtime(n) = $\lceil \log_{10}(n) + 1 \rceil \approx \lceil 0.3 \times \log_2(n) + 1 \rceil$ which is faster *than the runtime for computing binary parity, see Lecture 2*

5. **The aim is to construct a DFA that computes the parity of a sum of bits, i.e. $M(x_1 \cdots x_n)$ = parity($x_1 + x_2 + \cdots x_n$) for any bitstring length $n \in$ N. Which of the following four state diagrams describes the correct functionality?**



6. **If we restrict attention to length-2 bitstrings x_1x_2, then this DFA computes a very prominent logical function (gate). Which one is it?**
   XOR $((x_1 \vee x_2) \wedge \neg(x_1 \wedge x_2))$

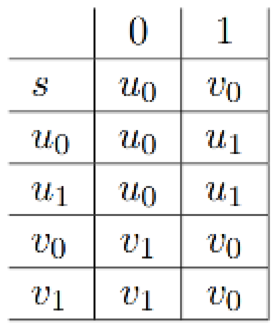
7. **Consider the following state diagram of a DFA working over the binary alphabet {0, 1}:**



$Q = \{s, u_0, u_1, v_0, v_1\}$, $\Sigma = \{0, 1\}$, $q_0 = s$ and $F = \{u_0, v_0\}$

8. **How does the correct transition function $\delta : Q \times \Sigma \rightarrow Q$ behave?**

b)

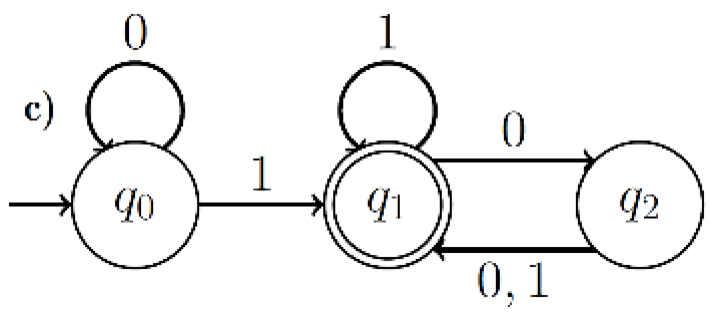


	0	1
s	u_0	v_0
u_0	u_0	u_1
u_1	u_0	u_1
v_0	v_1	v_0
v_1	v_1	v_0

9. **Which one of the following statements correctly describes the underlying functionality:**
This DFA accepts all bitstrings that start and end with the same symbol.

10. **Consider the DFA defined by the 5-tuple $M = (\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_1\})$, where the transition function δ acts like**
$\delta(q_0, 0) = q_0$, $\delta(q_0, 1) = q_1$,
$\delta(q_1, 0) = q_2$, $\delta(q_1, 1) = q_1$,
$\delta(q_2, 0) = q_1$, $\delta(q_2, 1) = q_1$.
Which of the following four state diagrams describes the correct functionality?

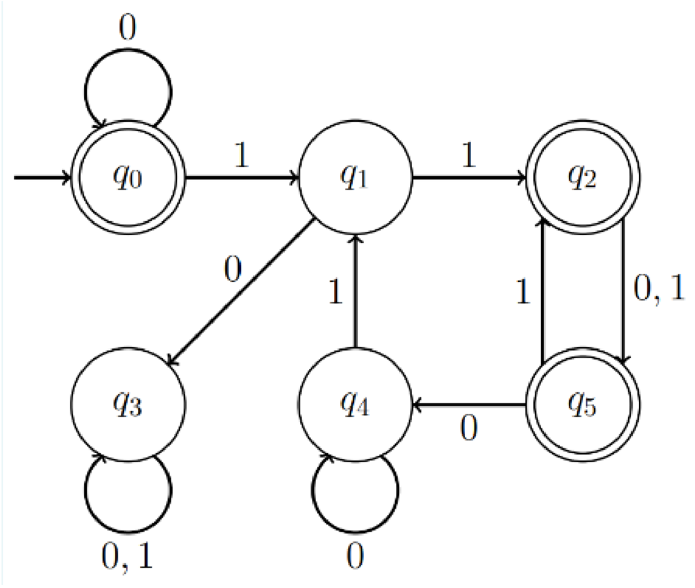
c)



11. **Which one of the following statements is true:**

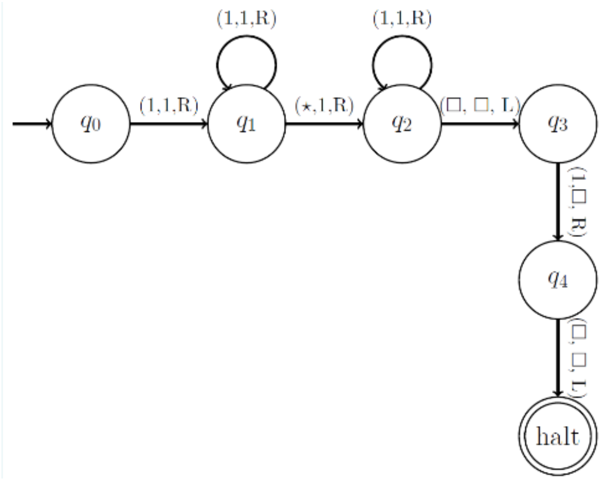
    This DFA accepts all bitstrings that contain at least one 1 and an even number of 0s that follow the last 1

12. **Consider the following state diagram over the binary alphabet Σ = {0, 1}.**



11010101

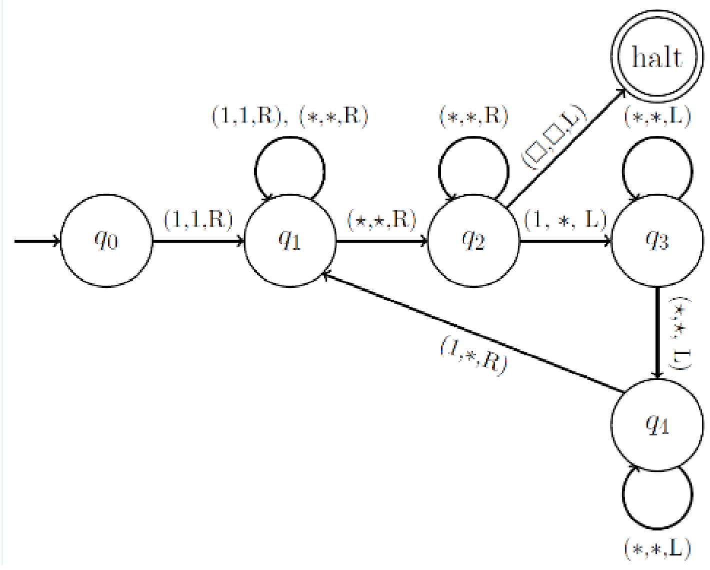
13. **We consider Turing machines over the alphabet Σ = {1, □, *, ⋆}, where (i) 1-symbols are used to write inputs & outputs, □ denotes 'empty' tape space, * is used to cross out input symbols that have already been processed and ⋆ denotes a certain arithmetic operation – the goal of this exercise, and the next one, is to find out which arithmetic operation it is. Throughout this exercise (and the next one), we use unary encoding for inputs and outputs. That is, numbers n ∈ N are represented by strings of exactly n ones: 1 ↔ 1, 2 ↔ 11, 3 ↔ 111, 4 ↔ 1111 and, more generally, n ↔ 1 n = 1 · · · 1 } n times . The Turing machine in question is characterized by the following state diagram:**



The arrows are labelled by 3-tuples that indicate 'read', 'write' and 'move' (L for left, R for right and S for stop). E.g. (⋆,1,R) means 'read ⋆', 'write 1' and 'move right'. In the following, 'underline' denotes the initial and final positions of the TM head. Which of the following statements is correct?

(⋆ = +) the TM computes the sum *of two numbers (in unary encoding) e.g. input <u>1</u> 1 1 ⋆ 1 1 □ produces output 1 1 1 1 1 <u>□</u> □.*

14. We consider Turing machines over the alphabet Σ = {1, □, *, ⋆}, where (i) 1-symbols are used to write inputs & outputs, □ denotes 'empty' tape space, * is used to cross out input symbols that have already been processed and ⋆ denotes a *certain* arithmetic operation – the goal of this exercise, and the previous one, is to find out which arithmetic operation it is. Throughout this exercise (and the previous one), we use *unary encoding* for inputs and outputs. That is, numbers $n \in \mathbb{N}$ are represented by strings of exactly n ones: 1 ↔ 1, 2 ↔ 11, 3 ↔ 111, 4 ↔ 1111 and, more generally, $n \leftrightarrow 1^n = 1 \cdots 1 \; \} n$ times . The Turing machine in question is characterized by the following state diagram:



The arrows are labelled by 3-tuples that indicate 'read', 'write' and 'move' (L for left, R for right and S for stop). E.g. (⋆,1,R) means 'read ⋆', 'write 1' and 'move right'. In the following, 'underline' denotes the initial and final positions of the TM head. Which of the following statements is correct?

(⋆ = −) the TM computes the difference *of two numbers (in unary encoding), e.g. input* <u>1</u> *1 1 ⋆ 1 1 □ produces output 1 * * ⋆ ** <u>*</u> *□.*

Exercise 2

1. **Suppose that we sample a bitstring x with even length n uniformly at random (i.e. we toss an independent coin for each bit to decide whether it gets a 0 or a 1).**

   The probability of obtaining a palindrome (x ∈ palindrome = {x ∈ {0, 1}* : $x^R = x$}), where $x^R = x_{n-1} \cdots x_0$ is the reverse of $x = x_0 \cdots x_{n-1}$) is *1/2^(n/2)*

   The probability of obtaining a 'Passueberquerung' (x ∈ same = {1^k0^k : k ∈ N}) is *1/2^n*

   The probability of obtaining a string with odd parity (x ∈ parity = {x ∈ {0, 1}* : the last bit of x is a 1}) is *½*

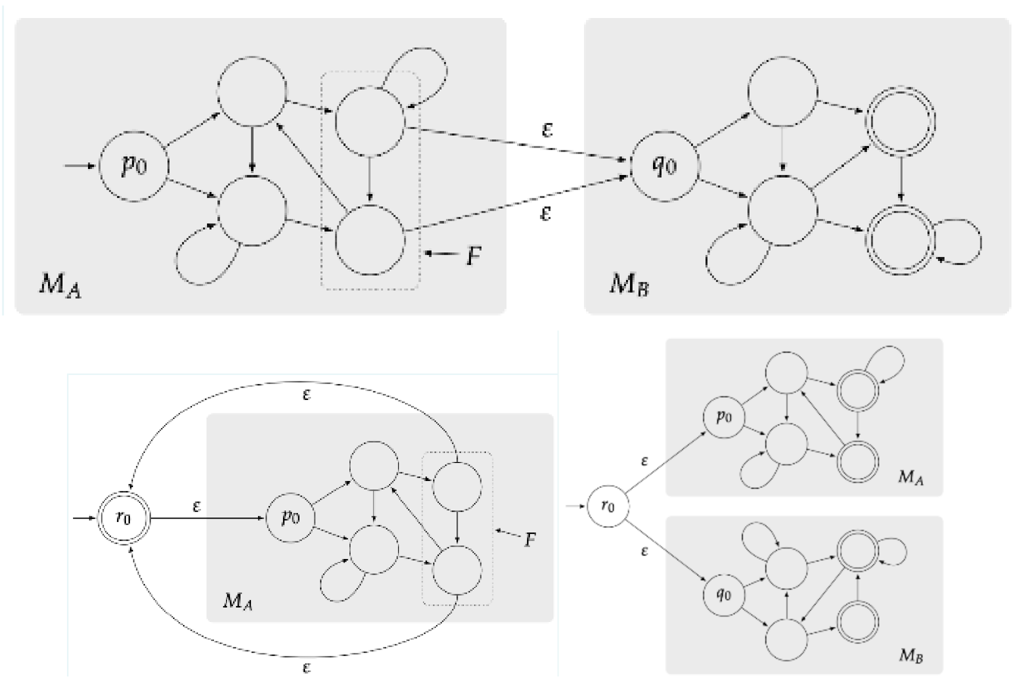
   The probability of obtaining a string with odd parity (x ∈ parity = {x ∈ {0, 1}* : the last bit of x is a 1}) is *½*

2. **The *union* $A \cup B$ of two languages $A,B \subseteq \Sigma^*$ is defined as:** *{x ∈Σ* : x ∈A or x ∈B}*

   The *concatenation* AB of two languages $A,B \subseteq \Sigma^*$ is defined as*:* *{xy ∈Σ* : x ∈A, y ∈B}*

   The *star* A^* of a language $A \subseteq \Sigma^*$ is defined as: *{ε} ∪A ∪AA ∪AAA ∪…*

3. **The illustrations in Figure 1 show how to combine two DFAs – one (M_A) for regular language A and one (M_B) for regular language B – to get another DFA that accepts a certain combination. Which of the following combinations is true?**



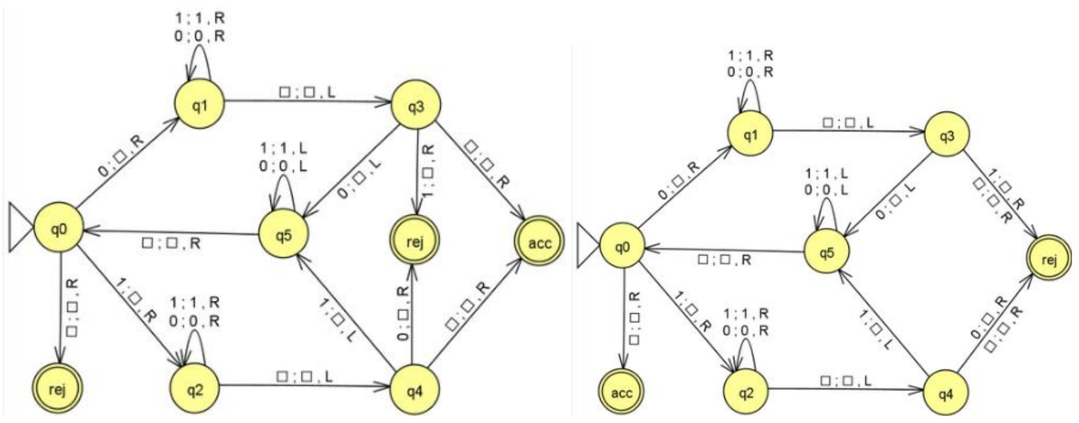
combination c) shows union, *combination a) shows* concatenation, *combination b) shows* star

4. **The *negation* operation $A^{-}=\{x\in\Sigma^*: x\notin A\}$ is a *regular operation*, because**
   we can convert a DFA accepting A into a DFA accepting A^{-} by changing the set of accepting states,

5. **The *intersection* operation $A\cap B=\{x \in \Sigma : x \in A \text{ and } x \in B\}$ is a *regular operation*, because**
   we can negate, then take the union and finally negate again to effectively compute it:
   $A \cap B = \overline{\overline{A} \cup \overline{B}}$

6. **The *reverse* operation $A^R = \{x^R : x \in A\}$, where $x^R = x_{n-1} \ldots x_0$ is the reverse of $x = x_0 \ldots x_{n-1}$, is a *regular operation*, because**
   we can convert a DFA accepting A into a DFA accepting A^R by reversing all arrows in the transition function and swapping starting and accept states;

7. **The TM state diagrams in picture a) and b) are connected to the language *PALINDROME*. Which of the following assertions is correct:**



a) solves odd palindrome, b) solves even palindrome;

8. **Which of the following implications is correct:**
   even PALINDROME and odd PALINDROME are both decidable*, but not* regular.

9. **A language $A \subseteq \Sigma^*$ is *decidable* if**
   we can construct a TM that accepts strings $x \in \Sigma^*$ if they are part of A and rejects them if they are not;

10. **A language $A \subseteq \Sigma^*$ is *regular* if**
    we can construct a DFA that accepts strings $x \in \Sigma^*$ if they are part of A and rejects them if they are not;

11. **A language $A \subseteq \Sigma^*$ is *semidecidable* if**
    we can construct a TM that accepts strings $x \in \Sigma^*$ if they are part of A;

12. **Which of the following inclusions is true:**
    regular $\subseteq$ decidable $\subseteq$ semidecidable;

13. **Let $A \subseteq \Sigma^*$ be a language and $\overline{A}$ its complement, i.e. $\overline{A} = \{x \subseteq \Sigma^* : x \notin A\}$. Which of the following statements is true:**
    A is decidable if and only if A and $\bar{A}$ are both semidecidable (both directions are always true);

14. **Fix an alphabet Σ and let $A \subseteq \Sigma^*$ be a language. The operation star is defined as $A^* = \{\varepsilon\} \cup A \cup AA \cup AAA \cup \cdots$,**
    where $AA = \{xy : x \in A \text{ and } y \in A\}$ denotes concatenation of A with itself and AAA, etc. is defined accordingly. Mark the *strongest* statement that is still correct:
    if A is a semi-decidable language, then A^* is also semi-decidable;

15. **Define the language HALT = { └(M,w)┘ :M is a TM that halts on input w}⊂{0,1}∗, where M describes a Turing machine, w∈{0,1}∗ and └(⟨M,w⟩)┘ is a suitable (and fixed) binary encoding of the two. Only one of the following statements about *HALT* is false. Which one is it?**
    HALT is not semi-decidable;

16. **The aim of this exercise is to show if a Turing Machine whose head may remain stationary can be simulated by a standard TM whose head must move right/left at each step. The key idea is to**
    simulate a stationary tape head (S) by executing two steps: one where the head moves right and one where the head moves left;

17. **The worst-case overhead in runtime (number of steps) is**
    at most a constant factor of two;

18. **The worst-case overhead in the required number of states is**
    at most two times the original number of states;

19. **The aim of this exercise is to show if a Turing Machine with tape alphabet Σ ∪ {□} can be simulated by a binary TM with tape alphabet Γ = {0, 1,□}. The most promising key idea is to**
    encode input symbols $\sigma \in \Sigma$ as bitstrings $\llcorner \sigma \lrcorner$; subsequently, we use additional internal states to 'recognize' these symbols which allows us to effectively execute the original transition function;

20. **The worst-case overhead in runtime (number of steps) is**
    more than factor of two, but at most logarithmic in the size of the input alphabet;

21. **The worst-case overhead in the required number of states is**
    more than two times the number of states, but at most proportional to $|Q||\Sigma|$;

22. **The aim of this exercise is to show if a TM with a one-way infinite tape can efficiently simulate another TM that has an infinitely long tape in both directions. The most promising key idea is**
    to bend the two-way infinite tape somewhere in the middle and introduce a new tape alphabet of size $|\Gamma|2$ to represent two symbols within one box;

23. **The worst-case overhead in runtime (number of steps) is**
    at most a constant factor of four;

24. **The worst-case overhead in the required number of states is**
    at most two times the original number of states;

Exercise 3

1. **What is the asymptotic scaling of the following functions:**
   $f(n) = 7n^7 + 6n^6 + 5n^5 + 4n^4 + 3n^3 + 2n^2 + n$ → *$O(n^7)$*
   $g(n) = \frac{7n^7+6n^6+5n^5+4n^4+3n^3+2n^2+n}{6n^6+5n^5+4n^4+3n^3+2n^2+n}$ → *$O(n)$*
   $h(n) = \frac{6n^6+5n^5+4n^4+3n^3+2n^2+n}{7n^7+6n^6+5n^5+4n^4+3n^3+2n^2+n}$ → *$O(1/n)$*

2. **What is the asymptotic scaling of the following function: $w(n)=n^2 2^n$**
   $w(n)=2^{O(n)}$

3. **Only one of the following reformulations is correct. Which one is it:**
   $2^{O(\log(n))} = n^c$ for some $c \geq 1$

4. **Three of the four following relations are true for any $k \in$ N. Mark the one which is wrong:**
   $2^{O(nk)} = O(2^{n^\wedge k})$

5. **The observation that exponential growth eventually outweighs any type of polynomial growth is central to computational complexity theory. It may be viewed as a consequence of the following big-O relation:**
   $n^k = O(2^n)$ for any polynomial degree $k \in \mathbb{N}$.
   'Prove' this relation by selecting the correct argument:
   For 'small' values of n, polynomials can actually be larger than exponentials. But choosing n_0 to be the smallest natural number that obeys $n_0/\log_2(n_0) \geq k$ ensures $2^n \geq n^k$ for all $n \geq n0$

6. **When analyzing runtimes of algorithms, we also often ignore (poly-)logarithmic factors. This is because polynomial growth outweighs logarithmic growth. This may be viewed as a consequence of the following big-O notation:**
   $\log_2^k(n) = O(n)$ for all $k \in \mathbb{N}$.
   'Prove' this relation by selecting the correct argument:
   We can use a change of variables trick. Writing $n = 2^m$ for some $m \in R$, this claim follows from Eq.(1).

7. **Suppose that we have two algorithms that compute the same Boolean function $f : \{0, 1\}^* \rightarrow \{0, 1\}$. Suppose that the first algorithm runs in time $a(n) = 1.01^n$ and the second algorithm runs in $b(n) = 100\,000 \times n^2$, where n denotes the size of the binary input x. Which of the following runtime comparisons is correct:**
   $b(n) = O(a(n))$;

8. **This asymptotic scaling kicks in once**
   $n \geq 2749$;

9. **Let M = ({q_0, q_1}, {0,1}, {0,1,□}, δ, q_0, q_0, q_1}) be the TM whose transition function is specified by $\delta(q0, 0) = (q_0, 0, R)$ and $\delta(q_0,1) = (q_1,1,R)$, respectively. Other transitions cannot occur. The running time measures the number of computational step, while the space complexity measures the number of TM tape squares that are being used. Which of the following statements is correct:**
   the worst-case runtime of M scales linearly in input length;

10. **Space complexity measures the maximum number of TM tape squares that are being used. Which of the following statements is correct:**
    the worst-case space complexity of M scales linearly in input length;

11. **Informally speaking, the problem class P encompasses problems that are 'easy' to solve with a TM. The P is an abbreviation for**
    ***P**olynomial runtime;*

12. **More formally, a language $A \subseteq \{0, 1\}^*$ is in P if**
    there exists a TM M that accepts $x \in A$ in a runtime that is (at most) polynomial in input size $|x|$, moreover M does not accept any string that is not in the language ($x \notin A$);

13. **Informally speaking, the problem class NP encompasses problems that are 'efficient' to verify. The name NP is an abbreviation for**
    ***N**ondeterministic **P**olynomial runtime*

14. **More formally, a language $A \subseteq \{0, 1\}^*$ is in NP if**
    There is a TM M (the 'verification procedure') such that for every $x \in A$ there is another string $y \in \{0, 1\}^*$ (the 'verifier') such that $M(x, y)$ = accept. Moreover, both the length $|y|$ of y and the runtime of M must be polynomial in $|x|$. Moreover, $x \notin A$, must imply $M(x, y)$ = reject for all possible verifiers y ('no false positives').

15. **Informally speaking, the problem class EXP encompasses problem that are computable, but in a truly expensive fashion. The name EXP is an abbreviation for**
    ***EXP**onential runtime;*

16. **More formally, a language $A \subseteq \{0, 1\}^*$ is in EXP if**
    There exists a TM M that accepts $x \in A$ in a runtime that is (at most) exponential in input size $|x|$. Moreover M does not accept any string that is not in the language ($x \notin A$);

17. **The class EXP contains an enormous number of problems. Which of the following two statements is correct:**
    *There are languages $A \subseteq \{0, 1\}^*$ that are not contained in **EXP**.*

18. **Only one of the following statements is not true. Which one is it:**
    P ≠ EXP would imply P ≠ NP;

19. **Recall the following languages and mark the strongest statement that is (known to be) true:**

    Factoring = {⌞ (M, k) ⌟ : M, k ∈ N and M contains a factor that is at most k}. *⊆ NP*

    Palindrome ={x ∈ {0, 1}* : $x^R = x$}, where R denotes bit reversion. *⊆ P*

    SumOfParities = {x ∈ {0, 1}* : x contains an odd number of 1s} *⊆ P*

    Halt = {⌞ (M,w) ⌟ : M encodes a TM that halts on input w}. *⊆ none of the above*

    It is possible to encode winning strategies for a chess game on a $n \times n$ board (where n can become much larger than 8) into binary languages. We say that ⌞ C ⌟ ∈ Chess if C encodes a (valid) distribution of chess pieces on a $n \times n$ board and white can still win from this position. Mark the strongest statement that is (known to be) true: *⊆ EXP*

20. **Let $\varphi(x0,\ldots,xn-1) \in \{0,1\}$ be a Boolean formula in n truth variables $x0,\ldots,xn-1 \in \{0,1\}$. A decision version of the Boolean satisfiability problem is**
    *SAT is in **NP**, but the explanation in (b) is not complete yet. It is also important to point out that └ϕ┘ ∉SAT necessarily implies that no satisfiable assignment can exist (or that the encoding is wrong). I.e. ϕ(x0,…,xn−1)=0 for all x0,…,xn−1, so there is no proof of correctness that leads us to accept such formulas.*

21. **Recall that the traveling salesperson problem (TSP) asks for evaluating the shortest route that visits a total of n cities exactly once. It is completely specified by a pairwise distance table $D \in R^{n \times n}$. There are many ways to convert TSP into a decision problem. But only one of the following choices is in NP. Which one is it:**
    TSP3 = {⌞ (D, k) ⌟ : D as above and there exists a route that requires at most k kilometers};

Exercise 4

1. **Here, we consider certain Boolean functions f:{0,1}^{3}→{0,1} on 3 bits.**
   $\varphi(x,y,z) = (x \vee y \vee z) \wedge (x \vee \bar{y} \vee \bar{z}) \wedge (\bar{x} \vee y \vee \bar{z}) \wedge (\bar{x} \vee \bar{y} \vee z)$

2. ***f*0(*x*,*y*,*z*) encodes a 3-bit version of an old friend of ours. Which one is it:.**
   SumOfParity, i.e. the total number of 1s is odd

3. **f1(x,y,z) = $(\bar{x} \wedge \bar{y} \wedge z) \vee (\bar{x} \wedge y \wedge z) \vee (x \wedge \bar{y} \wedge z) \vee (x \wedge y \wedge z)$ and mark the CNF that is functionally equivalent:**
   $\varphi(x,y,z) = (x \vee y \vee z) \wedge (x \vee \bar{y} \vee z) \wedge (\bar{x} \vee y \vee z) \wedge (\bar{x} \vee \bar{y} \vee z)$

4. ***f*1(*x*,*y*,*z*) encodes a 3-bit version of an old friend of ours. Which one is it:**
   Parity, i.e. the last bit is a 1

5. **Consider $f2(x,y,z) = (\bar{x} \wedge \bar{y} \wedge \bar{z}) \vee (\bar{x} \wedge \bar{y} \wedge z) \vee (\bar{x} \wedge y \wedge z) \vee (x \wedge y \wedge z)$ and mark the CNF that is functionally equivalent:**
   $\phi(x,y,z) = (x \vee \bar{y} \vee z) \wedge (\bar{x} \vee y \vee z) \wedge (\bar{x} \vee y \vee \bar{z}) \wedge (\bar{x} \vee \bar{y} \vee z)$

6. ***f*2(*x*,*y*,*z*) encodes a 3-bit version of an old friend of ours. Which one is it:**
   Up, i.e. all 1s must be to the right of 0s (e.g. 000,001,011,111).

7. **Consider $f3(x,y,z) = (\bar{x} \wedge \bar{y} \wedge \bar{z}) \vee (\bar{x} \wedge y \wedge \bar{z}) \vee (x \wedge \bar{y} \wedge z) \vee (x \wedge y \wedge z)$ and mark the CNF that is functionally equivalent:**
   $\varphi(x,y,z) = (x \vee y \vee \bar{z}) \wedge (x \vee \bar{y} \vee \bar{z}) \wedge (\bar{x} \vee y \vee z) \wedge (\bar{x} \vee \bar{y} \vee z)$

8. ***f*3(*x*,*y*,*z*) encodes a 3-bit version of an old friend of ours. Which one is it:**
   Palindrome, i.e. reading forwards equals reading backwards

9. **Consider the following simple clause of size 6:$\varphi(x) = (x_0 \vee x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5)$. Which of the following formulas in *x* = $x_0x_1x_2x_3x_4x_5$ and *y* = $y_0y_1y_2$ is a functionally equivalent 3-CNF:**
   $\psi(x,y) = (x_0 \vee x_1 \vee y_1) \wedge (\overline{y_1} \vee x_2 \vee y_0) \wedge (\overline{y_0} \vee x_3 \vee y_2) \wedge (\overline{y_2} \vee x_4 \vee x_5)$

10. **Suppose you had access to an algorithm that could reliably decide 3-SAT** = {└φ┘:φ **is 3-CNF that is satisfiable**} ⊆ **{0,1}∗. Now, you are given a *3*-SAT formula** φ(*x*0,…,*xn*−1) **in n variables that is satisfiable** (φ∈3-SAT).
    What is the smallest number *N* of algorithm invocations (the so-called `query complexity') that is required to also identify a satisfiable assignment, i.e.\ bits *x*♯0,…,*x*♯*n*−1∈{0,1} **such that** φ(*x*♯0,…,*x*♯*n*−1)=1:
    N = 2n

11. **Which of the following phrases describes the Cook-Levin theorem informally:**
    We can map any *NP-problem to an instance of 3-SAT, moreover the overhead is (at most) polynomial;*

12. **Let A ⊆ {0, 1}* be a NP-language with verifier M and let 3-SAT** = {└φ┘:φ is 3-CNF that is satisfiable}. **Which of the following phrases describes the Cook-Levin theorem formally:**
    Same as in (b), but the size of ϕ also depends polynomially on input length |x|. :
    b) *For each x ∈ {0, 1}* , there is a 3-CNF φ such that $x \in A$ if and only if φ ∈ 3-SAT. The function φ only depends on x (input) and M (verifier);*

13. **What is the conceptual insight that makes the proof check out:**
    all of the above:
    - *The elementary logical operations AND, OR and NOT are universal;*
    - *The verification process associated with NP languages is described by a polynomial runtime TM computation;*
    - *We can convert an entire TM computation into an AND of many small Boolean formulas;*

14. **Let $A,B \subseteq \{0, 1\}^*$ be two languages.**
    (a) and (b) and the function f must be computable in polynomial time:
    - *there is a function $f: \{0, 1\}^* \to \{0, 1\}^*$ such that $x \in A$ implies $f(x) \in B$;*
    - *there is a function $f: \{0, 1\}^* \to \{0, 1\}^*$ such that $f(x) \in B$ implies $x \in A$;*

15. **Intuitively, $A \leq_p B$ means**
    language B is at least as difficult as language A;

16. **Only one of the following implications is not true. Which one is it:**
    Palindrome $\leq_p$ 3-SAT implies **P** = **NP**;

17. **Let $A,B \subset \{0, 1\}^*$ be languages and let $A \leq_p B$ denote a Karp reduction (see Exercise 5). We say that a language A is NP-hard if**
    *$B \leq_p A$ for every $B \in$ **NP**;*

18. **We say that a language A is NP-complete if**
    *$B \leq_p A$ for every $B \in$ **NP** and $A \in$ **NP**.*

19. **Which of the following problems is (probably) not NP-complete:**
    Factoring = {└(N,k)┘ :N,k∈N and N has a factor ≤k};

Exercise 5

1. **Recall the definition of a space-bounded computation: Fix a function f : N → N. We say that a language A is contained in SPACE($f(n)$) if there is a Turing Machine (TM) that decides $x \in A$ vs. $x \notin A$ based on a maximum number of const × $f(|x|)$ non-blank tape symbols. In the following, mark the *strongest* assertion that is still true.**

   Let $A \subseteq \{0, 1\}^*$ be a *regular* language, i.e. a language that can be decided by a finite state automaton. Then $A \in$ *SPACE(n)*

   According to the TM discussed in Lecture 3, Palindrome = { $x \in \{0, 1\}^*$: $x^R = x$ } obeys *SPACE(n)*

   Let $M_{\text{binary}\rightarrow\text{unary}}$ be a TM that converts binary encodings of natural numbers into unary encodings: $\llcorner n \lrcorner \mapsto 1^n$ for $n \in$ N. Then $M_{\text{binary}\rightarrow\text{unary}}$ operates in *SPACE(2^n)*

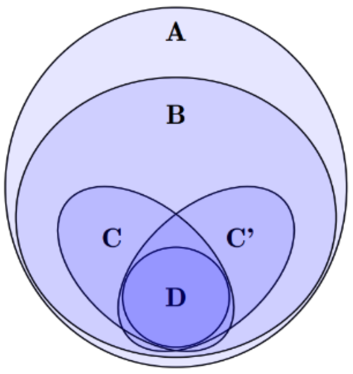
   Let Sum = {$\llcorner (a, b, c) \lrcorner$: $c = a + b$} ⊆ {0, 1}* be the language associated with addition. And suppose that we have access to a TM with separate input, working and output tapes. The space constraint only affects the central working tape. Then Sum ∈ *SPACE(log(n))*

2. **Mark the correct statement that comes with the correct justification:**
   P* ⊆ *PSPACE, *because every polynomial runtime computation can only exhaust a polynomial number of tape cells.*

3. **Mark the correct statement that comes with the correct justification:**
   NP* ⊆ *PSPACE, *because if $A \in$* ***NP***, *we can re-use tape space to cycle through all possible short certificates to decide whether $x \in A$ or $x \notin A$.*

4. **Mark the correct statement that comes with the correct justification:**
   PSPACE* ⊆ *EXP, *because there are 'only' exponentially many TM configurations to visit throughout the course of a computation*

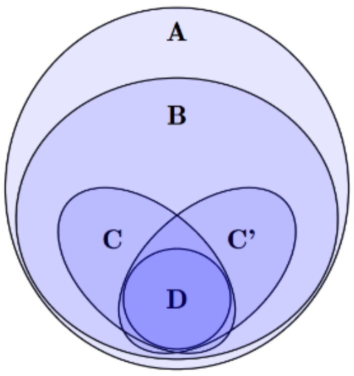
5. **Consider the following caricature of computational complexity classes:**



   The letters A, B, C, C', D are placeholders for different complexity classes.
   The letter A stands for *EXP*
   The letter B stands for *PSPACE*
   The letter C stands for *NP and/or co-NP*
   The letter C' stands for *NP and/or co-NP*
   The letter D stands for *P*

6. **The polynomial hierarchy is contained between**
   *classes **C** and **C'** on the one hand and **B** on the other;*

7. **Consider again the following caricature of computational complexity classes:**



   The letters A, B, C, C', D are placeholders for different complexity classes.
   The language 3-SAT is contained in *C or C'*
   The language Palindrome is contained in *D*
   The True Quantified Boolean Formula (TQBF) problem is contained in *B*
   The Factoring problem is known to be contained in the *intersection of C and C'*
   The question whether an optimal strategy for (Boolean) 2-player games with perfect knowledge exists is in *B*

8. **Consider the following Boolean formula: $\phi(x, y) = (x \vee y) \wedge (!x \vee !y)$. Mark the correct truth assignment for quantified versions:**
   $\exists x \exists y \varphi(x, y) = 1$ and $\forall x \forall y \varphi(x, y) = 0$;

9. **Mark the correct truth assignment for quantified versions:**
   $\forall x \exists y \varphi(x, y) = 1$ and $\exists x \forall y \varphi(x, y) = 0$;

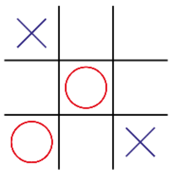
10. **The formula $\phi(x, y)$ describes**
    a 2-bit XOR-gate;

11. **Consider the following Boolean formula: $\phi(x, y) = (!x \wedge !y) \vee (!x \wedge y) \vee (x \wedge !y)$. Mark the correct truth assignment for quantified versions:**
    $\exists x \exists y \varphi(x, y) = 1$ and $\forall x \forall y \varphi(x, y) = 0$;

12. **Mark the correct truth assignment for quantified versions:**
    $\forall x \exists y \varphi(x, y) = 1$ and $\exists x \forall y \varphi(x, y) = 1$;

13. **The formula $\phi(x, y)$ describes**
    a 2-bit NAND-gate.

14. **Consider the following configuration in a standard tic-tac-toe game:**



    Suppose that it is blue's turn (crosses). Then,
    blue does have a winning strategy (100% success);

15. **Suppose that it is red's turn (circles). Then,**
    red does have a winning strategy (100% success);

16. **What is the maximum number n of turns any tic-tac-toe game can last:**
    n = 5;

17. **The problem class co-NP is defined by taking complements of problems in NP. Which of the following definitions is the correct one:**
    a language A is in co-NP if and only if its complement $\bar{A}$ is contained in NP;

18. **Only one of the following intuitive explanations is correct. Which one is it:**
    *A language A is contained in **co-NP** if and only if we can efficiently certify 'no'-instances ($x / \in A$);*

19. **Which of the following languages is (probably) not contained in co-NP:**
    *this is a trick question, all of the problems above are contained in **co-NP**.*

20. **Consider the following claim from the lecture: if NP = P, then co-NP = NP. Which of the following statements is a consequence of this claim?**
    co-NP≠NP implies NP≠P;

21. **The first level of the polynomial hierarchy encompasses familiar complexity classes. Which of the following equivalences is true:**
    $\Sigma p0=P$, $\Sigma p1=NP$ and $\Pi p1=co\text{-}NP$;

22. **Let q:N→N be a polynomial. Then, the problem class $\Sigma^p{}_2$ contains all languages for which**
    $x\in L$ if and only if $\exists a\in\{0,1\}q(|x|)\,\forall b\in\{0,1\}q(|x|)M(x,a,b) = 1$;

23. **Let q:N→N be a polynomial. Then, the problem class $\Pi^p{}_2$ contains all languages for which**
    $x\in L$ if and only if $\forall a\in\{0,1\}q(|x|)\exists b\in\{0,1\}q(|x|)M(x,a,b) = 1$:

24. **The polynomial hierarchy (PH) encompasses many of these complexity classes. What is the formal definition:**
    both of the above;
    - $PH=\bigcup i\geq 0\Sigma^p{}_i$;
    - $PH=\bigcup i\geq 0\Pi^p{}_i$;

25. **When implementing logical functionalities, it is desirable to represent them by Boolean functions that are as short as possible. This motivates defining the following language for Boolean formulas that are in disjunctive normal form (DNF): Min-DNF = {⌞ ϕ ⌟ : ϕ is a DNF formula not equivalent to any smaller DNF formula}. Which of the following inclusions is definitely correct:**
    Min-DNF $\in \Pi^p{}_2$.

Exercise 6

1. **Formally, circuits are**
   directed, acyclic graphs;

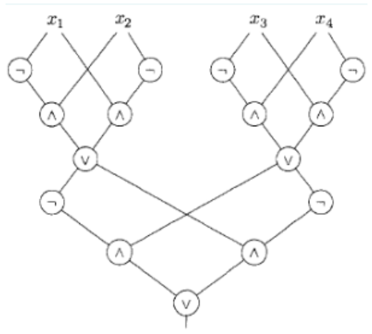
2. **It is important that the graph is**
   directed, because circuits have a directed information flow from inputs to outputs;

3. **In this course, we also assume that the graph is**
   acyclic, because we want other concepts to work out nicely: the absence of cycles allows us us to identify shortest paths;

4. **The *circuit size* is the**
   number of graph nodes that don't correspond to inputs and outputs;

5. **The *circuit depth* is the**
   length of the longest path from an input to an output node (not counting input and output nodes).

6. **Consider the following circuit where information flows from top to bottom:**

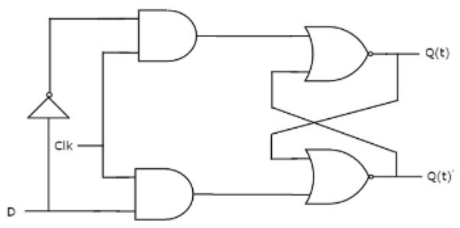
   

   This circuit has size
   15

7. **This circuit has depth**
   6

8. **This circuit computes**
   parity of sum

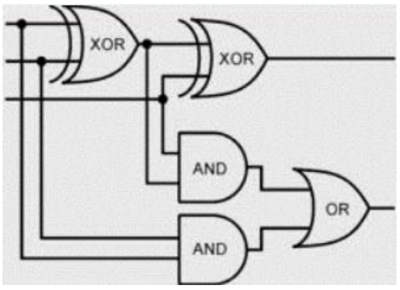
9. **Consider the following circuit where information flows from left to right:**

   This circuit has size
   5

10. **This circuit has depth**
    this is a trick question, circuit depth is not properly defined here.

11. **Consider the following circuit where information flows from left to right:**



This circuit has size

5

12. **This circuit has depth**

3

13. **Let φ:{0,1}n→{0,1} be a Boolean formula. The task is to find a circuit representation $C\varphi$ that is as efficient as possible. Suppose that φ is in CNF and contains m clauses of size k each.**

The best size scaling achievable is

$2km - 1 = O(mk)$

14. **The best depth scaling achievable is**

$\lceil log_2(k)\rceil + \lceil log_2(m)\rceil = O(log(km))$

15. **Would the (worst-case) size and depth bounds change if we considered DNF formulas ϕ instead:**

no, never

16. **Let C be a circuit with n input bits and m output bits (described by a directed acyclic graph). The time-reverse C^T is the circuit obtained by reverting the directed information flow within C (turn leftwards pointing arrows into rightward pointing arrows). A circuit is called *reversible***

$$C^TC(x) = x \text{ for all } x \in \{0, 1\}^n,$$
$$CC^T(y) = y \text{ for all } y \in \{0, 1\}^m.$$

The 3-bit *Toffoli gate* T: $\{0, 1\}^3 \rightarrow \{0, 1\}^3$ is an example of a (small) reversible circuit. It is a 'controlled-controlled-NOT' gate that inverts the third bit if and only if bit one and two are equal to one. Otherwise, all bits stay the same:

$$T(x, y, z) = \{ (x, y, \neg z) \text{ if } x = y = 1,$$
$$\{ (x, y, z) \text{ else.}$$

We denote the last outcome bit of a Toffoli gate by $T'(x, y, z) \in \{0, 1\}$.

Which of the following statements is a direct consequence of reversibility:

the number of input bits n must be equal to the number of output bits m;

17. **Which of the elementary logic gates (viewed as a 1- or 2-bit circuit) is reversible:**

the ¬-gate;

18. **One 3-bit Toffoli gate T and two ancilla bit wires can be used to generate a reversible NOT-gate (¬). Which formulation is correct:**
    NOT(x) = T(1, 1, x).

19. **Two 3-bit Toffoli gate T and one ancilla bit wire can be used to generate a reversible OR-gate (∨). Which formulation is correct:**
    OR(x, y) = T(1, y, T(x, y, x));

20. **One 3-bit Toffoli gate T and one ancilla bit one can be used to generate a reversible AND-gate (∧). Which formulation is correct:**
    AND(x, y) = T(x, y, 0);

21. **Which of the following statements is the strongest consequence of the above reformulations (in the sense that it is true and better than its alternatives):**
    Every circuit C with n inputs can be converted into a reversible circuit $\tilde{C}$ that has the same functionality. The overhead is linear in the sense that the new circuit has size $s(\tilde{C}) = O(s)$ and acts on $n' = n + O(s)$ inputs, some of which are fixed to either 0 or 1;

22. **The total number of reversible circuits with n inputs is exactly**
    $(2^n)!$

23. **The circuit equivalence problem asks whether two given circuits have the same functionality. It is very important in hardware design and design automation. The following language variant**
    Circuit-Equivalence = {⌞ (C_1, C_2) ⌟ : C_1, C_2 are circuits that compute the same function}.
    is definitely contained in one problem class we already know. Which one is it:
    Circuit-Equivalence ∈ co-NP;

24. **We say that a language A is contained in the problem class P if**
    A can be decided by a TM that only uses a polynomial number (in input length) of computational steps;

25. **We say that a language A is contained in the problem class PSPACE if**
    A can be decided by a TM that only uses a polynomial number (in input length) of tape squares;

26. **We say that a language A is contained in the problem class PSIZE if**
    A can be decided by circuits that only contain a polynomial number (in input length) of elementary gates (¬, ∨, ∧);

27. **We say that a language is in P/poly if**
    membership can be decided by a polynomial-runtime Turing machine that also has access to a single, polynomially-sized advice string. This advice string has polynomial length, can depend on the input length n, but nothing else.

28. **Mark the strongest statement that is known to be correct (in the sense that it is true and better than its alternatives):**
    both of the above:
    - *the problem classes PSIZE is contained in P/poly, because we can view a circuit description as a particular form of advice;*
    - *the problem classes P/poly is contained in PSIZE, because we can translate the advice string into a circuit blueprint;*

29. **Which of the following relational statements is correct:**

P $\subset$ ***PSIZE*** *(strict inclusion);*

30. Let $A \subseteq \{0, 1\}^*$ be a language. Mark the strongest claim that is known to be correct (in the sense that it is true and better than its alternatives):

same as in (b), but the converse implication ('$\Leftarrow$') is also true.

b) $A \in P \Rightarrow A$ can be decided with a polynomial-size circuit family that can be constructed by an efficient algorithm (TM);

31. We know that P/poly and NP are distinct, because

PSIZE contains the unary halting problem, while NP doesn't;

32. The Karp-Lipton theorem provides further evidence that the two problem classes are distinct. Formally, it states that:

if NP $\subseteq$ PSIZE, then $\Pi^p_2 \subseteq \Sigma^p_2$

33. A circuit family $\{C_n\}_{n\in N}$ is P-uniform if

there exists a polynomial-runtime TM that takes 1^n as input and outputs a description of the circuit C_n (unary encoding of input length).