NATIONAL UNIVERSITY OF SINGAPORE

CS 5230: Computational Complexity
Semester 2; AY 2024/2025; Final Exam

Time Allowed: 2 Hours

INSTRUCTIONS TO CANDIDATES

1. Please write your Student Number. Do not write your name.
2. This assessment paper consists of TEN (10) questions and comprises TWENTY-ONE (21) printed pages.
3. Students are required to answer **ALL** questions.
4. Students should answer the questions in the space provided.
5. This is a **CLOSED BOOK** assessment with one helpsheet of A4 size.
6. You are not permitted to communicate with other people during the exam and you are not allowed to use additional material beyond the helpsheet.
7. Every question is worth SIX (6) marks. The maximum possible marks are 60.

STUDENT NO: ______________________________

This portion is for examiner's use only

Question	Marks	Remarks	Question	Marks	Remarks
Question 1:			Question 6:		
Question 2:			Question 7:		
Question 3:			Question 8:		
Question 4:			Question 9:		
Question 5:			Question 10:		
			Total:		

Complete the following table. Provide the complexity of the operations with n-bit numbers (represented correspondingly in the machines). Some items are prefilled, as they might depend on the machine model or to reduce the number of items to fill. Each correct entry give 1/2 marks and marks are at the end uprounded to the next integer. Turing machines for this question are multi-tape machines which can have several tapes. Please provide the best known upper bound on the corresponding operations.

Operation	Counter Machine	Addition Machine	Turing Machine
Assignment	$O(1)$	$O(1)$	
Addition, subtraction			
Divisible by 3			
Comparison $(<,=,>)$			
Multiplication	$O(4^n)$		

Solution. The table is as follows, provided one follows the conventions from the task. The $O(1)$ for assignments in counter machine is a convention that this operation is a primitive operation, without this convention, the operation takes $O(2^n)$. Turing machine numbers are stored in binary and n is the number of binary digits, this same size measure of input size is also used for the other machine models.

Operation	Counter Machine	Addition Machine	Turing Machine
Assignment	$O(1)$	$O(1)$	$O(n)$
Addition and Subtraction	$O(2^n)$	$O(1)$	$O(n)$
Comparison $(<,=,>)$	$O(2^n)$	$O(1)$	$O(n)$
Is number divisble by 3	$O(2^n)$	$O(n)$	$O(n)$
Multiplication	$O(4^n)$	$O(n)$	$O(n \log n)$

Assume that a Turing machine has on the input tape a ternary number (digits 0, 1, 2) without leading 0s. The input starts with a 3 and ends with a 4. What is the complexity to check whether the number of 0s, 1s and 2s is the same?

☐ CONSTANT SPACE (REGULAR) ☐ LOGSPACE ☐ NLOGSPACE

Provide a multi-head read-only Turing machine to witness the choice. If you choose LOGSPACE, the machine should be deterministic, if you choose CONSTANT SPACE (REGULAR), the machine should have only one head. The machine should go to ACCEPT if the number of 0s, 1s, 2s are all the same and it should go to REJECT if some of the numbers differ. Nondeterministic machines should not have on some input both, runs ending up in ACCEPT and runs ending up in REJECT; runs found to be wrong can be aborted without going to ACCEPT or REJECT.

Solution. The right choice is LOGSPACE and two heads for the read-only Turing machine. At the beginning both heads move to the 3 before the word. In a first scan, the Turing machine checks whether there are exactly one third of the digits a 0. For this both heads scan the word. The first head checks the digit and each time the first head goes over a 0, the second head goes three fields forward. When the first head arrives at the 4 at the end, the Turing mahine checks whether the second head is also on the 4. If not, it rejects. Now both heads move back to the position of the 3 and redo the hole process this time counting 1s. Again if the number of 1s is not a third of that of all digits, the machine again goes to REJECT. A further such scan is done with respect to the number of 2s. If the third pass fails, then the machine goes to REJECT else the machine goes to ACCEPT.

Is there a Turing machine computing something in space NLINSPACE that cannot be computed in deterministic LINSPACE? Please answer as follows below.

☐ YES, ☐ NO, ☐ Unknown by current knowledge.

Give reasons for your answer and explain the background of the topic.

Solution. This is a famous open problem, it is one of the two LBA problems. Kuroda investigated in 1964 the concept of LBAs and proved that they can recognise the same languages as context sensitive grammars can generate. He closed his article with two LBA problems: First whether they are closed under complement. This problem was solved by Immerman and Szelepcsényi. Second whether deterministic and nondeterminisitic linear space coincide. This problem is open until today, though the Theorem of Savitch shows that NLINSPACE is contained in deterministic SQUARESPACE, that is, SPACE(n^2).

Write an addition machine program to compute the following function f where x, y are numbers in $\{1, 2, 3, \ldots\}$; if $x < 1$ or $y < 1$ the output 0 should be given. Let $g(x)$ be the number of decimal digits to write down x, so $g(1) = 1$ and $g(256) = 3$ and $g(9999) = 4$. The function f is now defined by the following formula:

$$f(x, y) = \begin{cases} 5 & \text{if } 0 < x \text{ and } x < y \text{ and } g(x) < g(y); \\ 4 & \text{if } 0 < x \text{ and } x < y \text{ and } g(x) = g(y); \\ 3 & \text{if } 0 < x \text{ and } x = y; \\ 2 & \text{if } 0 < y \text{ and } y < x \text{ and } g(x) = g(y); \\ 1 & \text{if } 0 < y \text{ and } y < x \text{ and } g(x) > g(y); \\ 0 & \text{if } x < 1 \text{ or } y < 1, \end{cases}$$

Note that it is okay to write $10 * w$ in place of $w+w+w+w+w+w+w+w+w+w$ in an addition machine program; however, multiplication should only be with integer constants.

Solution. The program would be written as an addition machine code as follows, where a subprogram is used for the function g:

```
function g(v)
  { variables w,q;
    lineg1: w = 1; q = 1;
    lineg2: w = w+w+w+w+w+w+w+w+w+w;
            if (v < w) { return(q); }
            q++; goto lineg2; }

function f(x,y) { var xx, yy, x, y, z;
begin   line1: read(x);
        line2: read(y);
        line3: xx = g(x); yy = g(y);
               z = 6; if (x<1) then begin z = 0; goto line4; end;
               if (y<1) then begin z = 0; goto line4; end;
               if (x=y) then begin z=3; goto line4; end;
               if (xx < yy) then begin z = 5; goto line4; end;
               if (x < y) then begin z = 4; goto line4; end;
               if (yy < xx) then begin z = 1; goto line4; end;
               if (y < x) then begin z = 2; goto line4; end;
               z = 0; goto line4;
        line4: return(z); end.
```

Furthermore, a C++ code for the function is here:

```
int g(long long int v)
  { long long int w; int q;
    lineg1: w = 1; q = 1;
    lineg2: w = 10*w;
            if (v < w) { return(q); }
            q++; goto lineg2; }
```

```
int main()
      { int xx=0; int yy=0; long long int x; long long int y; int z;
        line1: printf("Provide the input x: ");
               scanf("%lld", &x);
               printf("Input x provided is %lld.\n",x);
        line2: printf("Provide the input y: ");
               scanf("%lld", &y);
               printf("Input y provided is %lld.\n",y);
        line3: xx = g(x); yy = g(y);
               z = 6; if (x<1) { z = 0; goto line5; }
               if (y<1) { z = 0; goto line5; }
               if (x==y) { z=3; goto line4; }
               if (xx < yy) { z = 5; goto line4; }
               if (x < y) { z = 4; goto line4; }
               if (yy < xx) { z = 1; goto line4; }
               if (y < x) { z = 2; goto line4; }
               z = 0; goto line5;
        line4: printf("Output z is %d.\n",z); goto line1;
        line5: printf("Output z is %d.\n",z); return(z); }
```

(a) What is the problem kSAT and how is it defined?

(b) What do the Exponential Time Hypothesis (ETH) and the Strong Exponential Time Hypothesis (SETH) say about lower bounds for solving kSAT instances?

(c) Which is the smallest of the below upper bounds for 3SAT which are consistent with the current knowledge? So there should be a verified 3SAT algorithm for this bound but not for the better bounds (though the bound itself does not need to be optimal). Here n is the number of variables of an input instance.

☐ $O(n^8)$ ☐ $O(n^{\sqrt{n}})$ ☐ $O(1.7^n)$ ☐ $O(2^n)$.

Solution. The class kSAT is the set of all solvable instances which are, for some n, collections of clauses having up to k literals using the variables $x_1, \ldots, x_n$. That is, each literal is of the form y or $\neg y$ for an $y \in \{x_1, \ldots, x_n\}$. The problem kSAT with $k \geq 3$ is a standard example of a problem which is on one hand NP-complete and on the other hand allows an algorithm solving the problem in time $O(c^n) \cdot Poly(n+m)$ where $c < 2$, n is the number of variables and m the number of clauses. Such an algorithm has not yet been found for the main problem SAT itself.

Now ETH says that, for each $k \geq 3$, there are constants $c_k > 1$ and d_k such that for any correct algorithm for kSAT, there are infinitely many kSAT instances in which each variable appears in at most d_k clauses such that the algorithm uses at least time c_k^n on these instances.

Furthermore, SETH says that one can choose the above constants such that in addition the c_k converge to 2 for $k \to \infty$.

The best upper bound among the four choices is $O(1.7^n)$. The slides of the class have the upper bound $O(1.6181^n)$ and the best known upper bound is $O(1.3280^n)$.

Provide a DPLL algorithm for solving 2SAT, list the simplification rules and branching rules needed. DPLL algorithms are algorithms which have a lot of rules and always choose the first rule which applies until the problem is decided. Branching rules are rules where the algorithm fixes some variables and calls itself recursively using the values for these variables. State whether the algorithm uses branching rules and if so, why.

Solution. Branching rules are not needed, as the decision problem for 2SAT is in **P**. The following algorithm which uses simplificaiton rules only works.

While the algorithm has not terminated, simplify the instance by always selecting the first of the below rules which applies to the current instance.

1. If the instance contains a clause which is empty then terminate with REJECT.

2. If the instance does not contain any clause then terminate with ACCEPT.

3. If there is a literal x occurring in a single literal clause then remove all clauses containing the literal x and remove the literal $\neg x$ from all further clauses where it occurs.

4. If there is a literal x occurring in some clause while $\neg x$ does not occur in any clause then remove all the clauses which contain x.

5. If there is a clause containing, for some variable x, both the literals x and $\neg x$ then remove that clause.

6. Find a variable x such that there are clauses of the form $x \vee y$ and $\neg x \vee z$. Then put for each such pair $x \vee v$ abd $\neg x \vee w$ of clauses the new clause $v \vee w$ into the instance and afterwards remove all clauses containing x or $\neg x$. This step is called resolution by x. Note that putting a clause includes no change if the clause is already in the instance, so instances should be duplication free.

Note that there are at most $O(n^2)$ clauses in the instance at the start and during any time inside the algorithm. Furthermore, every check whether the corresponding item applies is done in polynomial time for the given instance and each item which applies either leads to termination or removes one variable from the instance. Thus the overall runtime is n times the runtime of the slowest item.

Karatsuba provided an algorithmic method to multiply two n-bit numbers in $O(n^{3/2})$. Subsequent work improved this bound further with the final bound being $O(n \log n)$ on a multi-tape Turing machine.

(a) Use this last bound to show that the Fibonacci number F_n can be computed from n in time $O(n \log n)$ where n is given as a binary number. For this use the following formulas: $F_{2m} = F_{m+1}^2 - F_{m-1}^2$, $F_{2m+1} = F_{m+1}^2 + F_m^2$, $F_{2m+2} = F_{m+2}^2 - F_m^2$, $F_0 = 0$, $F_1 = 1$, $F_2 = 1$.

(b) Compare this with the complexity used up by the standard algorithm doing the following:

$F_0 = 0$, $F_1 = 1$,
for $m = 2, \ldots, n$ do begin $F_m = F_{m-1} + F_{m-2}$ end.

Here additions are linear in the size of the numbers added; the size of F_m is $\Theta(m)$ bits.

Solution. (a) Let $n = a_1 a_2 \ldots a_k$ as a k bit binary number. The algorithm goes iteratively from 1 to k. As $a_1 = 1$ (leading digit), one starts with $F_1 = 1, F_2 = 1$. Now one computes for $m = a_1 a_2 \ldots a_h$ from F_m, F_{m+1} the values $F_{2m+a_{h+1}}, F_{2m+a_{h+1}+1}$ from F_m, F_{m+1} as follows: $F_{m-1} = F_{m+1} - F_m$, $F_{2m} = F_{m+1}^2 - F_{m-1}^2$, $F_{2m+1} = F_{m+1}^2 + F_m^2$, $F_{2m+2} = F_{2m+1} + F_{2m}$. Then one chooses the two values $F_{m'}, F_{m'+1}$ from these three by letting $m' = 2m + a_{h+1}$. Once this is done, one replaces m by m' and h by $h+1$ and continues until n and F_n are reached.

Note that one has on each of the levels 3 multiplications and $O(1)$ subtractions and additions, all together in $O(m \log m)$ time. Furthermore, m becomes doubled in each round, thus measured in the final n, the time complexity is bounded by $c \cdot n \log n + c/2 \cdot n \log n + c/4 \cdot n \log n + \ldots \leq 2c \cdot n \log n$ for some constant c, thus the algorithm runs in time $O(n \log n)$.

(b) The standard algorithm makes $n/2$ additions of numbers of size $\Omega(n)$ — the upper half of the updates $F_m = F_{m-2} + F_{m-1}$ — and thus the algorithm needs at least $\Omega(n^2)$ steps. On the other hand, this bound is also an upper bound, so that the algorithm runs in $\Theta(n^2)$. Using the fast algorithm with multiplications is definitely better.

Recall that an integer expression is formed by starting with finite sets (given as explicit lists of binary numbers) and then forming either the union or the sum of two integer expressions to get a further one. Recall that the sum of two sets A, B of integers $A + B$ equals to the set $\{a + b : a \in A \wedge b \in B\}$. The following questions aim at properties of this sum operation.

(a) Is there a set of 8 elements which is the sum of two sets of four elements each? If so, provide an example, if not, explain why this is not the case.

(b) Is there a set of 4 elements which is the sum of two sets of three elements each? If so, provide an example, if not, explain why this is not the case.

Solution. (a) The answer for the first question is YES. The integer expression $\{0, 1, 2, 3\} + \{0, 2, 3, 4\}$ describes the set $\{0, 1, 2, 3, 4, 5, 6, 7\}$ which has 8 elements.

(b) The answer for the second question is NO. If $\{x, y, z\}$ is a three element set added to $\{u, v, w\}$, then it has the elements $x + u, x + v, y + v, y + w, z + w$. Assuming that the sets are ordered by $x < y < z$ and $u < v < w$, the order of the elements $x + u, x + v, y + v, y + w, z + w$ is also strictly ascending, as from each sum to the next one, one of the two summands is replaced by a strictly larger one. Thus there are at least 5 elements and 4 is not possible.

(a) For the following problems, one is NP-complete, one is Σ_3^P-complete (complete for third level of the polynomial hierarchy) and one is PSPACE-complete. The three problems are these:

1. For inputs being an integer expression L and a binary number x, is $x \in L$?

2. For inputs being an integer expression L and a binary number y, is there an integer x such that $x+1, x+2, \ldots, x+y \in L$?

3. For a Reversi game position on an $n \times n$ board and a number c, can the player to move play such that this player has at the end, independently of what the opponent moves, at least c pieces more than the opponent?

Assuming that the Polynomial Hierarchy does not collapse to one of the first four levels, that is, has at least five distinct levels, say which problem is complete for which class.

(b) As all three problems are NP-hard, prove for any of these NP-hardness; one proof is enough.

Note that Reversi is a board game in which two players alternately put a new piece and turn all pieces between their new piece and some other own piece on a straight line (vertically, horizontally or diagonally). When no player can go on to move (as the board is full or no one can move in a way that a piece is turned) then the game finishes and the player with the most pieces wins; equal number of pieces is a draw. The game is also known under the name Othello. Usual sizes is 6*6 and 8*8 boards, but for complexity theory, one has to consider all possible board sizes.

Solution. (a) The first problem is NP-complete, the second is complete for Σ_3^P and the third problem is PSPACE-complete.

(b) For the first problem, note that a special case of the problem is that given n numbers $x_1, \ldots, x_n$ and a target y whether $y \in \{0, x_1\} + \{0, x_2\} + \ldots + \{0, x_n\}$. This is equivalent to saying that the SUBSETSUM problem with target y is satisfied, that is, y is the sum of a the members of some subset of $\{x_1, x_2, \ldots, x_n\}$. As SUBSETSUM is NP-complete, this problem is also NP-hard.

(a) Explain what the elementary sets are.

(b) Is there a universal set $E \subseteq \Sigma^* \times \Sigma^*$ of all elementary subsets of Σ^* which is elementary itself? Here E is a universal set of all elementary subsets of Σ^* if for all $F \subseteq \Sigma^*$, F is elementary if and only if there is an x satisfying for all y that $F(y) = E(x, y)$.

(c) Is every set which can be decided by an algorithm using arbitrary computation time and space also elementary?

☐ YES, ☐ NO, ☐ Unknown by current knowledge.

Explain your answer.

Solution. (a) Elementary sets are all sets which can be computed in time $O(f_k(n))$ for some fixed k where $f_1(n) = 2^n$ and $f_{k+1}(n) = 2^{f_k(n)}$ for all k and all inputs x with length n.

(b) By a hierarchy result in computational complexity, there is for every k a set E_k which can be computed in time $O(f_{k+1}(n))$ but not in time $O(f_k(n))$. If there would be a universal set, then this set would be computable with some bound $O(f_k(n))$, however the set E_k would then not be among the sets listed in this array. Thus no universal set of all elementary sets is elemantary. However, there are non-elementary universal sets of all elementary sets.

(c) All complexity classes have a decidable universal set and the complement of the diagonal of the universal set is then a decidable set which is not in the class. This is also true for the complexity class of all elementary sets.

END OF QUESTION PAPER.