NATIONAL UNIVERSITY OF SINGAPORE

CS 5230: Computational Complexity

Semester 2; AY 2023/2024; 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:		

Question 1 [6 marks]

Assume that a addition machine runs the following program:

```
Line 1: Read x;
Line 2: y = 1; z = 0;
Line 3: If y >= x Then Goto Line 5;
Line 4: y = y+y; z = z+1; Goto Line 3;
Line 5: v = 0;
Lint 6: w = 0;
Line 7: y = y+y; w = w+1; If w < z Then Goto Line 7;
Line 8: y = y+y; v = v+1; If v < z Then Goto Line 6;
Line 9: Write y.
```

It is assumed that the input x is a positive integer. Let $n = \min\{m : 2^m \geq x\}$ be the size of x. Recall that addition machines have integer variables and can add, subtract, compare, assign and excecute goto-commands in unit time. Find a function F in terms of n such that the runtime of the above program is $\Theta(F(n))$.

Is there a faster program for an addition machine with runtime $o(F(n))$ producing the same output y for each input $x > 0$? Prove the answer.

Solution. The program computes in the first loop the size n in the variable z and has $y = 2^n$. In Lines 5-8, it goes through a nested loop. After that, the size of y is $\Theta(n^2)$ at the output and also the run time used so far is $\Theta(n^2)$. Then the program outputs y.

An addition machine can which each operation increase the size of the largest register at most by 1. Thus it needs $\Theta(n^2)$ operations to get an output of the same size as the output of this program. Therefore there is no faster program computing the same output and running in time $o(n^2)$.

Is there a Turing machine computing something in space LINSPACE what cannot be computed in NLOGSPACE?

Please answer as follows: ☐ YES, ☐ NO, ☐ Unknown by current knowledge.

Give reasons for your answer.

Solution. The answer is YES. By Savich's Theorem, NLOGSPACE is contained in SPACE($\log^2(n)$). This bound is space constructible and strictly below the much larger space constructible bound n. Thus one can provide a function which is computable by a LINSPACE Turing machine which is not computable by an NLOGSPACE machine.

Let P be the class of polynomial time decidable languages, PH be the polynomial hierarchy (P, NP, NP[NP], ...) and PSPACE be the class of languages decided by Turing machines using polynomial space. Find for each of these classes a characterisation using alternating computations and explain these characterisations in detail.

Solution. P is the class of problems recognised by an alternating LOGSPACE machine. This is a two-tape Turing machine with one input tape and one work tape, where the Turing machine uses on all inputs of length n at most space $c \cdot (\log(n+2))$ on the work tape for some constant c. This two-tape machine has sometimes several options to branch and a player (Anke or Boris) has to decide which option is taken. An input word is in a language L recognised such a machine if the player Boris can enforce that the machine accepts; an input word is outside L if the other player Anke can enforce with her decision that the machine rejects.

PSPACE is the class of alternating time computations where the time constraint is some polynomial in n and again two player, depending on the situation, decide how the computation goes on. Independent of their decisions, the computation comes after $p(n)$ steps to a halt and the machine says either ACCEPT or REJECT. Again a word w is in the language L of the machine iff Boris can enforce that the machine says ACCEPT; a word is outside the language L if Anke can enforce that the machine says REJECT.

The class PH is defined similarly with only one additional constraint: That for each language L there is a constant c how often the right to choose is transferred from one player to the other. In the case of NP, the right to choose always stays with Boris; in the case of NP[NP], the right to choose is initially with Boris, but the machine might at some point decide that the right is transferred to Anke and then it never comes back. Similarly for higher levels in the hierarchy.

Is the problem SAT (Boolean Satisfiability of CNF formulas) complete for EXPSPACE? Give reasons for your answer.

Solution. The problem SAT is in LINSPACE, as one can make an algorithm which stores the current Boolean values of the n variables as an n-bit word on the work tape and which then goes lexicographically through all n-bit words and checks for each combination, whether it satisfies all clauses. However EXPSPACE allows to use much more space than LINSPACE and therefore one can construct problems in EXPSPACE which do not many-one reduce to any problem in LINSPACE.

Provide an algorithm which counts all solutions of a kSAT instance in time $O((2^k - 1)^{n/k})$ where n is the number of variables. What is the limit of the branching factors $(2^k - 1)^{1/k}$ of this algorithm for $k \to \infty$? Why can this limit not be better than the value computed under assumption of the Strong Exponential Time Hypothesis?

Solution. A kSAT instance is given by a set of k-clauses. The idea is that every kSAT clause provides for k variables to occur in this clause (if occur less, one can add unrelated variables) there is at least one choice of truth-values of these k variables which cannot occur. Thus one branches these k variables simultaneously and makes $2^k - 1$ recursive calls of the algorithm for the $2^k - 1$ possible choices for these k variables which make the clause true. Note that the branching factor is $(2^k - 1)^{1/k}$ as one does in every way from the start to the end exactly n/k branchings and not n branchings. Each recursive call returns the number of solutions in this branch and the calling instance returns the sum of all these numbers as its output.

The Strong Exponential Time Hypothesis says that there is no $c < 2$ such that one can, for all k, decide kSAT in time $O(c^n)$. If one counts the number of solutions, then one can decide also the solvability of the instance, as the instance is solvable iff the number of solutions is at least 1.

The Unique Exponential Time Hypothesis says that there are constants c_3, d_3 such that for every correct 3SAT algorithm which is correct on instances with up to one solution there are infinitely many n for which there is an instance with n variables and up to $d_3 n$ clauses and up to one solution such that the 3SAT algorithm takes longer than time c_3^n.

Use this to construct 2SAT instances with $n' = (1 + 5d_3)n$ variables such that, when counting the number of 2SAT solutions modulo 17, a correct counting algorithm uses at least time $(c_3^{1/(2+5d_3)})^{n'}$ on infinitely many input instances.

Solution. One does this by translating the 3SAT instances. One uses variables $x_1, \ldots, x_n$ plus $y_{k,1}, \ldots, y_{k,5}$ for the k-th clause for $k = 1, 2, \ldots, d_3 n$. For each z occurring in the k-th clause, one puts the implications $z \to y_{k,h}$ for $h = 1, 2, 3, 4, 5$. Furthermore, for each k one puts the implications $\neg y_{k,1} \to y_{k,h}$ for $h = 2, 3, 4, 5$. Thus if no literal in the k-th clause is 1 then $(y_{k,1}, \ldots, y_{k,5})$ can take any of the values $01111, 10000, 10001, \ldots, 11110, 11111$ which are 17 possible values for this five-bit tuple in total. If some literal z in the k-th clause is true then this literal implies that every solution of the 2SAT system takes the values 11111 for the corresponding five variables $y_{k,1}, \ldots, y_{k,5}$. Thus each satisfying assignment of the coded 3SAT instance contributes 1 number of 2SAT solutions while each nonsatisfying assignment of the 3SAT instance contributes a multiple of 17 of solutions to the constructed 2SAT instance. Thus if the coded 3SAT instance has a unique solution than the number of solutions of the 2SAT instance modulo 17 is 1 and if it has no solution than the number of solutions of the 2SAT instance modulo 17 is 0. It follows that counting the 2SAT solutions decides coded 3SAT instances with at most one solution and thus the overall runtime of the 2SAT solution counter including the time to make the translation of the instances is at least $(c_3^{1/(1+5d_3)})^{n'}$ on infinitely many input instances. As one has to exclude the translation time for making the 2SAT instance, one makes the base of the exponent a bit smaller by giving the lower bound $(c_3^{1/(2+5d_3)})^{n'}$.

Karatsuba provided an algorithmic method to multiply two n-bit numbers by splitting them into numbers of the form $a + b \cdot 2^{n/2}$ and $c + d \cdot 2^{n/2}$ such that he then computed with three multipliction of $n/2$-bit numbers values e, f, g for which $(a + b \cdot 2^{n/2}) \cdot (c + d \cdot 2^{n/2}) = e + 2^{n/2} \cdot f + 2^n \cdot g$.

Which three multiplications of $n/2$-bit numbers are needed to compute e, f, g together with some operations like adding and subtracting and shifting bits by $n/2$ or n positions. Provide the formulas for e, f, g explictly using a, b, c, d. What are the complexity of those operations?

Provide the recursion-formula for the time complexity of the algorithm as $Time(n) = i \cdot Time(n/2) + O(n^j)$. What are i and j? The solution to the recursion formula is $O(n^{\log(i)})$ when $\log(i) > j$ and $O(n^j)$ when $\log(i) < j$. Say which of these two cases arises. The logarithm is base two with $\log(1) = 0, \log(2) = 1, \log(4) = 2$.

Solution. The lecture provides methods which give every c with $c > 1$; however, the nearer the c is to 1, the more complicated the method. The easiest method is Karatsuba's Algorithm from 1960, which is done by recursive splitting of the input numbers in half portions (first $n/2$ bits and second $n/2$ bits). So if the input is $(a+b\cdot 2^{n/2})\cdot(c+d\cdot 2^{n/2})$ then one first computes $a\cdot c$, $b\cdot d$ and $(a+b)\cdot(c+d)$, all of these go with half amount of digits. After that, using that adding, subtracting and comparing are $O(n)$ and multiplication with powers of 2 is just shifting and thus also $O(n)$, one computes $e = a \cdot b$, $g = c \cdot d$ and $f = (a + b) \cdot (c + d) - e - g$ and outputs the number $e+f\cdot 2^{n/2}+g\cdot 2^n$. This gives the following recurrence: $Time(n) = 3\cdot Time(n/2)+O(n)$. The overall performance is then $Time(n) \in O(n^{log(3)}) \subseteq O(n^{1.58497})$.

Let 3EOR be the problem to check whether in a set of n m-bit strings there are three strings such that their bitwise exclusive or gives the null-string. Provide an $O(n^2 \log(n)m)$ algorithm for this task.

Solution. The idea for the algorithm is this: First write each of the strings into a data base. This is $O(n \log(n) \cdot m)$. Then for each two strings compute in time $O(m)$ the bit-wise exclusive or and check then in time $\log(n) \cdot m$ whether it is in the data base. This is time $O(n^2 \cdot \log(n) \cdot m)$ by standard data base techniques, as there are $\Theta(n^2)$ pairs to deal with.

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. Answer the following questions and give reasons for the answers.

(a) Is there an integer expression describing a set with infinitely many integers in it?

(b) Let the size of an integer expression be how many number constants are used in it; using a number constant twice counts twice and so on. Give an integer expression of size as small as possible for the set $\{0, 1, 2, 3, 4, 5, 6, 7, 8\}$.

Solution. (a) There is no integer expression describing an infinite set. The base expressions describe only finite sets. The union of two finite sets is finite. If A has a and B has b elements, then $A + B = \{x + y : x \in A, y \in B\}$ has at most $a \cdot b$ elements, thus is also finite. So one can prove by induction over all expressions that each expression is finite.

(b) One possibility is to write $\{0, 1, 2\} + \{0, 3, 6\}$, this expression has six elements and generates the full set. Furthermore, the expression $\{0, 1\} + \{0, 2\} + \{0, 4\} \cup \{8\}$ has seven numbers, so it is suboptimal. If one combines a finite set of four or five elements with a further set, then this further set needs to have at least two elements for generating a set with at least seven elements, thus it is not better. A combination of expressions with three and with two elements, respectively, has at most six elements and needs to be combined with a further expression having at least two numbers, thus is also suboptimal. Thus the first expression is the best, no strictly smaller expression generates the same set.

What is the complexity of integer expression containment? Here the size of an expression is (in contrast to the last question) the number of symbols to write it down (set brackets, commas, digits of binary numbers, union and sum symbols, brackets to order the priority of operations). What is the complexity class of comparing two integer expressions with respect to the relation "subset or equal"? No proof is required. Furthermore, show (by a suitable reduction) that one can first reduce the question "Is $H \subseteq L$?" by a many-one reduction to a question of the type "Is $H' \subset L'$?" for suitable H', L' and second reduce "Is $H \not\subseteq L$?" also to an equivalent question of the form "Is $H'' \subset L''$?" for suitable H'', L''. The reductions have to be polynomial time computable.

Solution. The complexity of the relation "subset or equal" is Π_2^P-complete, that is, $CoNP[NP]$-complete. Assume now that a question whether $H \subseteq L$ is given and one wants to reduce it to the question whether a proper subset holds.

Now one first constructs two expressions H', L''' such that $H' = \{2x : x \in H\}$ and $L''' = \{2y : y \in L\}$. This is done by appending a single digit 0 to all binary numbers occurring in the expressions. One has that $H \subseteq L$ iff $H' \subseteq L'''$. Now one let $L' = (L''') \cup \{1\}$. Thus the same question becomes equivalent to "Is $H' \subset L'$?". Note that 1 cannot be a member of H' which contains only even numbers, thus the subset relation, whenever it holds, must be proper. Furthermore, if x witnesses that $H \not\subseteq L$ then $2x$ witnesses that $H' \not\subseteq L''$ and thus that $H' \not\subset L'$. Thus the reduction given is a many-one reduction. Furthermore $H \not\subseteq L$ iff $H'' \subset L''$ with $H'' = L$ and $L'' = H \cup L$, thus both $\subseteq$ and $\not\subseteq$ are many-one reducible to $\subset$; the many-one reduction is polynomial time computable and does not increase the size more than doubling the length of the formula in the worst case.

END OF QUESTION PAPER.