NATIONAL UNIVERSITY OF SINGAPORE

CS 5230: Computational Complexity

Semester 2; AY 2024/2025; Midterm Test

Time Allowed: 60 Minutes

INSTRUCTIONS TO CANDIDATES

1. Please write your Student Number. Do not write your name.
2. This assessment paper consists of FIVE (5) questions and comprises ELEVEN (11) 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 page helpsheet.
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 30.

STUDENT NO: ______________________________

This portion is for examiner's use only

Question	Marks	Remarks
Question 1:		
Question 2:		
Question 3:		
Question 4:		
Question 5:		
Total:		

What is known about the relationship of the complexity class **P** with other complexity classes. Tick in each row the corresponding answer. An equality can be (a) yes (known to be true), (b) no (known to be false) or (c) maybe (where, according to current knowledge, either (a) or (b) are possible). Here "N" stands for nondeterministic and "A" for alternating computations.

Is **P** = **LOGSPACE**? ☐ YES ☐ NO ☐ MAYBE

Is **P** = **NLOGSPACE**? ☐ YES ☐ NO ☐ MAYBE

Is **P** = **ALOGSPACE**? ☐ YES ☐ NO ☐ MAYBE

Is **P** = **POLYLOGSPACE**? ☐ YES ☐ NO ☐ MAYBE

Is **P** = **PSPACE**? ☐ YES ☐ NO ☐ MAYBE

Is **P** = **APSPACE**? ☐ YES ☐ NO ☐ MAYBE

Solution. It is known that **P** equals to alternating **LOGSPACE**, **ALOGSPACE** for short. Furthermore, it is known that **APSPACE** equals **EXP** (exponential time) and is thus different from **P**. Furthermore, the lecture had a homework that **P** $\neq$ **POLYLOGSPACE**. The remaining three **LOGSPACE**, **NLOGSPACE** and **PSPACE** maybe equal to **P** provided that the following side-conditions are taken care off: **LOGSPACE** $\subseteq$ **NLOGSPACE** $\subseteq$ **P**, **NLOGSPACE** $\subset$ **PSPACE**. Thus **LOGSPACE** $\subseteq$ **NLOGSPACE** $=$ **P** is possible where the first relation can or cannot be an equality and **NLOGSPACE** $\subset$ **P** $=$ **PSPACE**. Which of these possibilities applies to the real world, is not known.

Is the following implication correct?

If **NP** = **NLOGSPACE** then **CoNP** = **NLOGSPACE**.

Give reasons for the answer.

Solution. By the theorem of Immerman and Szelepcsényi the class **NLOGSPACE** is closed under complement. If **NP** equals this class then for every $L \in \mathbf{NP}$ also the complement H of L is in **NLOGSPACE** and thus in **NP**. Thus $\mathbf{NP} = \mathbf{Co} - \mathbf{NP}$.

Consider the following class $S3SAT$ where every variable x occurs at most in one single clause with three literals; all further clauses containing x have either one or two literals. The following exponential time algorithm solves $S3SAT$ where the input F is a set of clauses:

Function S3SATSOLVE(F): While Program did not return Do Begin

1. If there is no clause then return (ACCEPT);
   If there is an empty clause then return (REJECT);
2. If there is a one-literal clause x, then remove $\neg x$ from all clauses containing $\neg x$ and remove all clauses containing x.
3. If there is a literal x occurring in some clauses while $\neg x$ occurs in no clause, then remove all clauses containing x.
4. If there is a variable x occurring in a 3-literal-clause then compute the values S3SATSOLVE($F \cup \{x\}$) and S3SATSOLVE($F \cup \{\neg x\}$); If both values are REJECT then return(REJECT) else return(ACCEPT).
5. If all variables occur only in two variable clauses, then pick any such variable x and for each pair of clauses $x \vee y$, $\neg x \vee z$ with further literals y, z, put the clause $y \vee z$ into the instance and after this is completed, remove all clauses containing x or $\neg x$.

End. Answer the following questions:

(a) Is the invariant of the algorithm that each variable occurs in at most one clause with more than two literals kept? Give reasons for your answer. Furthermore, verify that the algorithm does not get stuck without solving the instance.

(b) Provide a branching factor for the algorithm which is as good as possible. The underlying measure is the number of variables n. The below table provides for pairs (a, b) what the branching factor is when on one side of the branching a variables and on the other one b variables are eliminated.

a	b	Branching Factor
1	1	2
2	1	1.6181
3	1	1.4656
4	1	1.3803
2	2	1.4142
3	2	1.3248
4	2	1.2721
3	3	1.2600
4	3	1.2208
4	4	1.1893

Provide for the algorithm a branching factor c such that this c with algorithm running in $O(c^n)$ should be as small as possible and explain why this c is a correct upper bound. Partial credit for a suboptimal c with reasons will be given.

Solution. (a) The invariant is kept, as only variables are removed by branching them until no clause with three more more variables exists. Only then resolution is done and it only creates two-literal claues out of other two-literal clauses so that no clause with larger amount of literals is recreated. Thus the invariant is never violated. Furthermore, whenever there are zero, one or three literal clauses, one of the first four lines of the algorithm applies, similarly also when some variable either occurs only positive or only negatively. Thus when none of the steps one to four apply, then the last step applies.

(b) Assume $x \vee y \vee z$ is a three-literal clause. If one branches x the following happens: For $x = 1$ the clause $x \vee y \vee z$ disappears and therefore y, z do no longer occur in the three-literal clause and will therefore not be branched. For $x = 0$ the three-literal clause will be replaced by the two-literal clause $y \vee z$ and therefore again y, z will not be branched. Thus at most $n/3$ of the variables will be branched, that is, the branching factor for $(3, 3)$-branching applies and one has that the algorithm runs in time $O(1.2600^n)$.

Furthermore, consider the following instance of m 3-literal clauses and $n = 3m$ variables $x_1, y_1, z_1, x_2, y_2, z_2, \ldots, x_m, y_m, z_m$: For each $k = 1, 2, \ldots, m$ put the clauses $x_k \vee y_k \vee z_k$, $\neg x_k \vee \neg y_k$, $\neg x_k \vee z_k$, $\neg y_k \vee z_k$ into F and no other claues. Then the algorithm branches in each clause one variable and does not terminate early, thus it descends 2^m branchings and this shows that the above upper bound is also the best. Thus the worst-case performance of the algorithm is $\Theta(2^{n/3})$ though better algorithms for the problem might exist; for example, lazy evaluation of the branching would already require another worst case than this example.

Question 4 [6 marks]

A directed graph is coded as follows onto a string. The entry for the i-th node starts with a 2 (there are $i-1$ 2s before) and then a binary word follows with the j-th bit being 1 in this word if one can go from i to j in the graph. The first symbol of the input word is a 3 and the last symbol is a 4.

(a) Draw the directed graph for the coding 3201121001200011224.

(b) Describe how a nondeterministic two-head read-only Turing machine can verify that there is a path from the first node to the last node in the graph. The heads of the Turing machine can detect when they are on the same position. Also they see the symbol under the position to verify where they are.

Solution. (a) The graph has 5 nodes.

```
P1 <--> P2
|       |
v       v
P3 ---> P4
|
v
P5
```

(b) The Turing machine puts at the beginning both heads onto the field with the 3 and then advances the first head to the first 2. Then nondeterministically the first head goes to the i-th entry 0 or 1 follwoing the 2 (without crossing a further 2) while the second head moves to the i-th entry 2 in the string without crossing the 4 at the end, if the first head crosses a 2 or the second a 4 then abort the computation. Now the Turing machine can go from the first node to the i-th node in the graph if the entry number i of the binary string after the first 2 is 1, if not again the Turing machine aborts the computation. Note that the first head advances by one step in the above loop if and only if the second head reaches a new 2. Thus this operation can be carried out easily.

If now the Turing machine has found a 1 and can move to the i-th node then it does it by moving the first head to the same position as the second head and afterwards resetting the second head at the initial position by moving backwards until the 3 is found. Now again the first head advances to some j-th bit of the binary word after the 2 while the second head advances to the j-th 2 in the input string. If the first head in this loop reaches a 2 or the second head reaches a 4 then the computation aborts. Now again if the j-th bit is 1 then one goes in the graph to the j-th node and does this by moving the first head to the position of the second head and then moving the second head to the 3 at the start.

This is iterated until the Turing machine is with the first head on the last 2 of the coding. Now it verifies that the 4 is the next nonbinary digit after that 2. If this happens then the machine accepts. In all other cases it aborts the computation and does not accept the input string.

Consider the following addition machine program.

1. Read x; if $x < 0$ then let $x = -x$;
   let $x = x + x + x + 1$; let $y = 1$; let $z = 0$;
2. if $y > x$ then goto line 3; $y = y + y + y$; goto line 2;
3. If $x < y$ then let $x = x + x + x$;
4. If $x > y$ then begin let $z = 1 - z$; let $x = x - y$ end;
5. If $x \neq y$ then goto line 3;
6. If $z = 0$ then ACCEPT else REJECT.

A hint is that this program analyses the ternary representation of natural numbers. For example, it accepts the ternary numbers 11 and 20 and rejects the ternary number 12. Provide a natural description for the set of numbers recognised by this addition machine. Which of the following is the run time complexity of the program:

☐ $\Theta(1)$ ☐ $\Theta(n)$ ☐ $\Theta(n \log n)$ ☐ $\Theta(n^2)$.

Here the complexity parameter n in the number of quinary digits of the input x. Run time $\Theta(f(n))$ means that the worst case run time of the program is, up to a constant factor, the value $f(n)$. Primitive operations (additions, subtractions, comparisons, ...) count as $O(1)$. Explain your solution.

Solution. The program first makes x a natural number by doing $x = -x$ when it is negative. Then it puts a coding digit in ternary coding after the input and then processes it ternary digit by ternary digit. Each digit gets added to z and the remainder by two is always taken, this remainder oscilates between 0 and 1 whenever the current digit is downcounted by 1. If at the end this remainder is 0 then the addition machine accepts else it rejects.

For a base system k, if one adds the digits modulo $k - 1$ then the number is a multiple of $k - 1$ if and only if this sum is 0. Thus the addition machine just checks whether the input is a multiple of 2 or not. Note that $-x$ is a multiple of 2 if and only if x is a multiple of 4, thus one can indeed, as this program does, just process natural numbers and map the negative numbers to their positive counterparts.

The runtime of the program is linear. It has two main loops, the first brings y up to a power of 3 above the input x with a coding bit appended and the second gets out all quinary digits and adds them to z modulo 4. Both loops run in time $\Theta(n)$.

The program was tested with the following C++-code for correctness.

```
#include <stdio.h>

long long int x, y, z;

int main() {
        line1: printf("Provide the input x: ");
               scanf("%lld", &x);
```

```
       printf("Input x provided is %lld.\n",x);
               if (x < 0) {x = -x; } y = 1; z=0;
       x = x+x+x+x+x+1;
line2: if (y>x) { goto line3; } y = y+y+y; goto line2;
line3: if (x < y) { x = x+x+x; }
line4: if (x > y) { z=1-z; x=x-y; }
        line5: if (x != y) { goto line3; }
line6: printf("Output z is %lld.\n", z); }
```

END OF MIDTERM TEST.