CS3800: Theory of Computation — Summer II '22 — Drew van der Poel

Homework 6
Due Friday, August 19 at 11:59pm via Gradescope

Name:
Collaborators:

- Make sure to put your name on the first page. If you are using the LaTeX template we provided, then you can make sure it appears by filling in the `yourname` command.
- This assignment is due Friday, August 19 at 11:59pm via Gradescope. No late assignments will be accepted. Make sure to submit something before the deadline.
- Solutions must be typeset. If you need to draw any diagrams, you may draw them by hand as long as they are embedded in the PDF. I recommend using the source file for this assignment to get started.
- I encourage you to work with your classmates on the homework problems. *If you do collaborate, you must write all solutions by yourself, in your own words.* Do not submit anything you cannot explain. Please list all your collaborators in your solution for each problem by filling in the `yourcollaborators` command.
- Finding solutions to homework problems on the web, or by asking students not enrolled in the class is strictly forbidden.

Problem 1. *Big-O* (3 points)

For each of the following statements, state whether it is true or false. If it is true, provide an argument as to why (this doesn't have to be overly formal). If it is false, provide a counterexample.

Note the formal definition of Big-O is as follows: $f(n) = O(g(n))$ if there exist $n_0, c > 0$ such that $f(n) \leq c * g(n)$ for all $n \geq n_0$.

We assume that all functions are positive, continuous, non-decreasing and that they are of the form $\mathbb{N} \to \mathbb{R}$.

(a) **[1 pt.]** For any pair of functions $f(n)$ and $g(n)$, either $f(n) + g(n) = O(f(n))$ or $f(n) + g(n) = O(g(n))$.

Solution:
True, you can break $f(n)$ and $g(n)$ into 3 cases:

$f(n) > g(n) \quad \forall n \geq n_0$:

In this case, $f(n) + g(n) = O(f(n))$

(b) **[1 pt.]** For any pair of functions $f(n)$ and $g(n)$, if $f(n) + g(n) = O(f(n))$ then $f(n) > g(n)$ for all $n \geq n_0$.

Solution:
False, if we let $f(n) = n$ and $g(n) = n$ then $f(n) + g(n) = 2n$

$f(n) + g(n) \leq c * f(n) \quad \forall n \geq n_0$

$2n \leq c * n$

This is true if $c = 2$ and $n_0 = 1$

Therefore, $f(n) + g(n) = O(f(n))$ but $f(n) = g(n)$ for all $n \geq n_0$

(c) **[1 pt.]** For any pair of functions $f(n)$ and $g(n)$, if $f(n)+g(n) = O(f(n))$ and $f(n)+g(n) = O(g(n))$, then $f(n) = g(n)$.

Solution:
False, if we let $f(n) = n$ and $g(n) = 2n$ then $f(n) + g(n) = 3n$:

1a. $f(n) + g(n) \leq c * f(n) \quad \forall n \geq n_0$
1b. $3n \leq c * n$
1c. This is true if $c = 3$ and $n_0 = 1$
Therefore, $f(n) + g(n) = O(f(n))$

2a. $f(n) + g(n) \leq c * g(n) \quad \forall n \geq n_0'$
2b. $3n \leq c * 2n$

2c. This is true if $c = 1.5$ and $n_0' = 1$
Therefore, $f(n) + g(n) = O(g(n))$

$f(n) + g(n) = O(f(n))$ and $f(n) + g(n) = O(g(n))$ but $f(n) \neq g(n)$

Problem 2. *7th-Smallest is P (*4 points*)*

Consider the following problem 7TH-SMALLEST. You are given an array containing $n > 100$ natural numbers and are tasked with finding the 7th smallest value. Show that 7TH-SMALLEST $\in P$. State the running time of your algorithm on a deterministic single-tape Turing machine in terms of the length of the input array on the tape. You can use any reasonable encoding, just be sure to state the encoding and what the length of the input array is.

Solution:
Let M be the Turing Machine that decides 7TH-SMALLEST where the input to M is $< A >$, the encoding of the input array.

The input array will use a unary encoding for each number with a $*$ delimiter separating each number in the array. Let n be the length of the input array and let L be the length of the string encoding for the largest number in A, then the length of the encoding, $\langle A \rangle$, is:
$O(nL)$

```
TM $M$ = On input $\langle A \rangle$:
    while
```

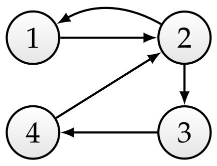
Problem 3. *Reduction Practice* (7 points)

Consider the following two problems:

DIRECTEDHAMILTONIANCYCLE = $\{\langle G\rangle|$ directed graph G contains a directed Hamiltonian cycle$\}$
UNDIRECTEDHAMILTONIANCYCLE = $\{\langle G\rangle|$ undirected graph G contains a Hamiltonian cycle$\}$

Note that a (directed) Hamiltonian cycle is a simple (directed) cycle, which visits every node in the graph exactly once. Recall that a simple cycle means that no vertex or edge repeats (except for the first/last node).

Show that DIRECTEDHAMILTONIANCYCLE is mapping reducible to UNDIRECTEDHAMILTONIANCYCLE. Show why your reduction works on the following graph (which has no directed Hamiltonian cycle) by applying your reduction and arguing why the resulting graph has no undirected Hamiltonian cycle:



Solution:

Problem 4. *Thought Experiment* (6 points)

(a) **[2 pts.]** Recall E_{TM}, the Turing machine emptiness problem, and the $\overline{EQ_{TM}}$, the complement of the Turing machine equivalence problem. We know E_{TM} is undecidable. Use a general reducibilty argument to show that $\overline{EQ_{TM}}$ is undecidable.

Solution:
Assume true that $\overline{EQ_{TM}}$ is decidable, so there exists a Turing Machine, D, that decides $\overline{EQ_{TM}}$.

Let S be a Turing Machine that takes in the same inputs as the E_{TM} problem, $\langle M \rangle$ where M is some TM.

```
TM $S$ = on input $\langle M \rangle$:

    1. Construct a new Turing Machine, $B$ that takes in an arbitrary string $\langle x \rangle$

    TM $B$ = on input $\langle x \rangle$:

        a. $B$ rejects $x$

    (So $L(B) = \emptyset$)

    2. Run $D$ on $\langle M, B \rangle$, if $D$ rejects by halting then $S$ should accept
    else if $D$ accepts then $S$ should reject
```

Therefore, with the help of D, the TM that decides $\overline{EQ_{TM}}$, there exists a TM S that decides E_{TM} which is a contradiction since E_{TM} is undecidable. Hence, TM D cannot exist which means that $\overline{EQ_{TM}}$ is undecidable.

(b) **[2 pts.]** We said in class that just because we are able to use a general reducibility argument to show undecidability, doesn't mean that that argument can be extended to show unrecognizabiility. We will show an example of such a case with attempting to extend the reduction from part (a). Note that we showed in class that E_{TM} is unrecognizable.

Suppose we try to make an unrecognizability argument by reducing E_{TM} to $\overline{EQ_{TM}}$. What is our initial assumption? What does this imply about the existence of a Turing machine? What is the input and behavior of this machine?

Solution:
We assume that $\overline{EQ_{TM}}$ is recognizable, so this implies the existence of a Turing Machine, D, that recognizes $\overline{EQ_{TM}}$. The input to D would be $\langle M_1, M_2 \rangle$ where M_1 and M_2 are Turing Machines.

(c) **[1 pt.]** Now we want to work towards a contradiction. Given that we are reducing from E_{TM}, what is the contradiction we'd like to arrive at? What exactly would we need to show exists?

Solution:
We need to show that with the existence of D, we can construct a Turing Machine, S, that

recognizes E_{TM}. This is the contradiction we would like to arrive at since we know E_{TM} is unrecognizable and therefore, S, cannot exist.

(d) **[1 pt.]** Given what we are assuming in part (b) and what we'd like to show in part (c), explain why we aren't able to use our Turing machine from part (b) to complete the necessary task in part (c).

Solution:

Problem 5. *Fun with Encodings (*10 points*)*

Consider a Turing machine which takes as input a natural number x and repeatedly decrements x (sets $x = x - 1$) until $x = 0$.

We use this simple process to illustrate the impact that the string encoding has on the runtime. To further highlight the differences, we require that when going from x to 0, the Turing machine goes through a series of checkpoints where only $\langle x \rangle$ is on the tape, then only $\langle x - 1 \rangle$ is on the tape, and so on until only $\langle 1 \rangle$ is on the tape, and finally only $\langle 0 \rangle$ is on the tape. Note that other strings may be on the tape in between $\langle i \rangle$ and $\langle i - 1 \rangle$, we just need to ensure that each checkpoint is hit.

(a) **[2 pts.]** First consider using a unary encoding. That is, if $x = 5$, $\langle x \rangle = 11111$. We'll assume $x = 0$ is represented by a blank tape. Give an algorithm which follows the checkpoints as stated above. Analyze the running time in terms of (1) the length of the encoding of x ($|\langle x \rangle|$) and (2) the value of x (x).

Solution:
Let M be the TM on this process, on input $\langle x \rangle$:

```
1. move right until we've found the last 1 on the tape

2. while we're currently on a 1:

        a. replace the 1 with a ⊔ (empty cell) and move to the left by one

3. if we're currently on a ⊔, M will halt on an accept state
```

(b) **[2 pts.]** Now we update the notion of a checkpoint to enforce that the tape head is in the first (leftmost) cell when the encoding of each value is on the tape. Give an algorithm which follows this new notion of checkpoint. Analyze the running time in terms of (1) the length of the encoding of x and (2) the value of x.

Solution:
Let M be the TM on this new process, on input $\langle x \rangle$:

```
1. while we're currently on a 1:

        a. mark our current cell as 1*
        b. move right until we've found the last 1
        c. if we're currently on a 1 or 1*, replace it with a ⊔ (empty cell)
        d. move left once
        e. move left until we've at a 1* or ⊔
        f. if we're at a 1* replace it with 1

3. if we're currently on a ⊔, M will halt on an accept state
```

(c) **[4 pts.]** Now we consider a binary encoding. That is, if $x = 5$, $\langle x \rangle = 101$. Give an algorithm which follows the original notion of checkpoint used in part (a). Analyze the running time in terms of (1) the length of the encoding of x and (2) the value of x.

Solution:
Let M be the TM on this process, on input $\langle x \rangle$:

```
1. if we're currently on a 1 or 0:
    a. mark the start as 1*
    b. move right until we've found the last 1 or 0 on the tape
    c. if we are on a 1, replace it with a 0
    d. else if we are on 1*: replace it with ⊔
    e. else if we are on 0:
        i. move left until we find the next 1 or 1*:
            while we are moving left, replace 0's with 1's
        ii. if we've found a 1, replace it with a 0
        iii. else if we've found a 1*, replace it with 0*
        iv. else if we can't find anymore 1's or 1*'s, move right and replace all 0's, 1's and 0*'s with ⊔
        and proceed to step 3
    f. move left until we've found a 1* or 0*:
        i. if we've found a 1*, replace it with 1
        ii. if we've found a 0*, replace it with ⊔ and shift all of the 1s and 0s to the left by 1
    g. repeat step 1 if condition applies

2. if we're currently on a ⊔, $M$ will halt on an accept state
```

(d) **[2 pts.]** State the ordering of the algorithms from parts a, b, and c when measuring runtime in terms of the length of the encoding of x. Then state the ordering of the algorithms from parts a, b, and c when measuring runtime in terms of the value of x

Solution: