University of Toronto
Faculty of Arts and Science

CSC165H1S Midterm 2, Version 2

Date: March 20, 2019 Duration: 75 minutes **Instructor(s)**: David Liu, François Pitt

No Aids Allowed

Name:

Student Number:

- This examination has **4** questions. There are a total of **10 pages, DOUBLE-SIDED**.
- All statements in predicate logic must have negations applied directly to propositional variables or predicates.
- In your proofs, you may always use definitions we have covered in this course. However, you may **not** use any external facts about these definitions unless they are given in the question.
- For algorithm analysis questions, you can jump immediately from an exact step count to an asymptotic bound without proof (e.g., write "the number of steps is $3n + \lceil \log n \rceil$, which is $\Theta(n)$").

Take a deep breath.

This is your chance to show us how much you've learned. We **WANT** to give you the credit that you've earned. A number does not define you.

Good luck!

Question	Grade	Out of
Q1		7
Q2		5
Q3		6
Q4		9
Total		27

1. **[7 marks] Short answer.** You do **not** need to show your work for any parts of this question.

   (a) **[1 mark]** Write down the balanced ternary representation of the decimal number 100. The representation should not have any leading zeros. HINT: $3^3 = 27$, $3^4 = 81$. You do not need any higher powers of 3.

   Solution

   $100 = 81 + 27 - 9 + 1 = (11T01)_{bt}$

   (b) **[1 mark]** Let $n \in \mathbb{Z}^+$. What is the *largest* number that can be expressed by an n-digit binary representation? (Your answer should be in terms of n and can be in the form of a summation.)

   Solution

   $$\sum_{i=0}^{n-1} 1 \cdot 2^i = 2^n - 1$$

   (c) **[2 marks]** Let $f(n) = \dfrac{n^2 - n + 7}{4n + 3}$ and $g(n) = 5\sqrt{n}$. For each statement below, check one box to indicate whether the statement is true or false.

	TRUE	FALSE		TRUE	FALSE		TRUE	FALSE
$f(n) \in \mathcal{O}(n)$	✓		$g(n) \in \Omega(n)$		✓	$f(n) \in \Omega(g(n))$	✓	
$f(n) \in \Theta(g(n))$		✓	$g(n) \in \Theta(\log_3 n)$		✓	$f(n) + g(n) \in \Theta(f(n))$	✓	

   (d) **[1 mark]** Consider the following algorithm.

```
def f(n: int) -> None:
    """Precondition: n >= 0."""
    i = 2
    while i + i < n:
        i = i * i * i
```

   Find a formula for i_k, the value of variable i after k iterations (where $k \in \mathbb{N}$).

   Solution

   $i_k = 2^{3^k}$

   (e) **[2 marks]** Use your answer from the previous part to find the exact number of iterations this function's loop will run. Use floor or ceiling to ensure that the number of iterations is an integer.

Solution

We need to find the smallest value of k that satisfies the inequality $2 \cdot i_k \geq n$, i.e., $2 \cdot 2^{3^k} \geq n$.
$k = \lceil \log_3(\log_2 n - 1) \rceil$

2. [**5 marks**] **Induction.** Prove the following statement using induction.

$$\forall n \in \mathbb{N},\ n \geq 2 \Rightarrow \prod_{i=1}^{n} \frac{2i-1}{2i} \geq \frac{1}{2n}$$

(Recall that $\prod$ is *product notation*, similar to summation except each term is multiplied rather than added.)

Solution

Proof. We'll prove this statement by induction on n.

Base case: let $n = 2$. We'll prove that $\prod_{i=1}^{n} \frac{2i-1}{2i} \geq \frac{1}{2n}$.

The left side is

$$\prod_{i=1}^{2} \frac{2i-1}{2i} = \frac{2(1)-1}{2(1)} \cdot \frac{2(2)-1}{2(2)} = \frac{1}{2} \cdot \frac{3}{4} = \frac{3}{8}$$

The right side is $\frac{1}{2(2)} = \frac{1}{4}$.

So the left side is greater than the right side.

Induction step: let $k \in \mathbb{N}$, and assume that $k \geq 2$ and that $\prod_{i=1}^{k} \frac{2i-1}{2i} \geq \frac{1}{2k}$. We'll prove that $\prod_{i=1}^{k+1} \frac{2i-1}{2i} \geq \frac{1}{2(k+1)}$.

We start with the left side of the target inequality:

$$\begin{aligned}
\prod_{i=1}^{k+1} \frac{2i-1}{2i} &= \left(\prod_{i=1}^{k} \frac{2i-1}{2i}\right) \cdot \frac{2(k+1)-1}{2(k+1)} \\
&\geq \frac{1}{2k} \cdot \frac{2(k+1)-1}{2(k+1)} && \text{(by I.H.)} \\
&= \frac{1}{2k} \cdot \frac{2k+1}{2(k+1)} \\
&\geq \frac{1}{2(k+1)} && \text{(since } 2k+1 \geq 2k\text{)}
\end{aligned}$$

□

3. **[6 marks] Asymptotic analysis.** You may refer to the following definitions for this question.

$$\begin{aligned} g \in \mathcal{O}(f): &\quad \exists c, n_0 \in \mathbb{R}^+,\ \forall n \in \mathbb{N},\ n \geq n_0 \Rightarrow g(n) \leq c \cdot f(n) \\ g \in \Omega(f): &\quad \exists c, n_0 \in \mathbb{R}^+,\ \forall n \in \mathbb{N},\ n \geq n_0 \Rightarrow g(n) \geq c \cdot f(n) \\ g \in \Theta(f): &\quad g \in \mathcal{O}(f) \land g \in \Omega(f) \end{aligned}$$

Disprove the following statement. Begin by writing its negation; you may, but are not required to, expand the definitions of Big-Oh, Omega, and/or Theta in the negated statement.

$$\exists a \in \mathbb{R}^+,\ an^2 + 1 \in \Theta(n^4)$$

__Solution__

The negation is

$$\forall a \in \mathbb{R}^+,\ an^2 + 1 \notin \mathcal{O}(n^4) \lor an^2 + 1 \notin \Omega(n^4)$$

Proof. Let $a \in \mathbb{R}^+$. We'll prove that $an^2 + 1 \notin \Omega(n^4)$.

Let $c, n_0 \in \mathbb{R}^{\geq 0}$. Let $n = \left\lceil \max(n_0, \sqrt{\frac{2a}{c}}, \left(\frac{c}{2}\right)^{\frac{1}{4}} \right\rceil + 1$. We'll prove that $n \geq n_0$ and that $an^2 + 1 < cn^4$.

__Part 1__: proving that $n \geq n_0$.

By the definition of n, we know that $n \geq \max(n_0, ...) \geq n_0$.

__Part 2__: proving that $an^2 + 1 < cn^4$.

First:

$$\begin{aligned} n &> \sqrt{\frac{2a}{c}} && \text{(by definition of } n\text{)} \\ n^2 &> \frac{2a}{c} \\ \frac{c}{2}n^4 &> an^2 && \text{(multiplying by } \frac{c}{2}n^2\text{)} \end{aligned}$$

Also:

$$\begin{aligned} n &> \left(\frac{2}{c}\right)^{\frac{1}{4}} && \text{(by definition of } n\text{)} \\ n^4 &> \frac{2}{c} \\ \frac{c}{2}n^4 &> 1 \end{aligned}$$

Adding these two inequalities yields the desired $an^2 + 1 < \frac{c}{2}n^4 + \frac{c}{2}n^4 = cn^4$. $\square$

Use this page for rough work. If you want work on this page to be marked, please indicate this clearly *at the location of the original question.*

4. **[9 marks] Running time analysis.**

   (a) **[3 marks]** Consider the following algorithm.

```python
def f(n: int) -> None:
    """Precondition: n >= 0."""
    for i in range(n, 0, -1):   # Loop 1  (i = n, n-1, ..., 1)
        j = 0
        while j < i * i:        # Loop 2
            print(2 * j)
            j = j + 2
```

Find the **exact total number of iterations of Loop 2** when `f` is run, in terms of its input n. To simplify your calculations, you may ignore floors and ceilings. Use the following formula to simplify any summations you find in your expression (valid for all $m \in \mathbb{N}$):

$$\sum_{i=1}^{m} i^2 = \frac{m(m+1)(2m+1)}{6}$$

Note: make sure to explain your work in English, rather than writing only calculations.

Solution

First, we analyse the number of iterations of Loop 2 for a *single* iteration of Loop 1:

- Since j starts at 0 and takes on the values $0, 2, 4, \dots$ until its value is $\geq i^2$, Loop 2 takes $\frac{i^2}{2}$ iterations (we're ignoring floor/ceiling here).

Next, we need to add up this number over all iterations of Loop 1. Loop 1 runs n iterations, for $i = n, n-1, \dots, 1$. So the total number of iterations of Loop 2 is:

$$\begin{aligned}\sum_{i=1}^{n} \frac{i^2}{2} &= \frac{1}{2}\sum_{i=1}^{n} i^2 \\ &= \frac{1}{2} \cdot \frac{n(n+1)(2n+1)}{6} \\ &= \frac{1}{6}n^3 + \frac{1}{4}n^2 + \frac{1}{12}n\end{aligned}$$

(b) [**6 marks**] Consider the following algorithm, which takes as input a list of integers.

```python
def my_alg(lst: List[int]) -> None:
    n = len(lst)
    for i in range(n):                  # Loop 1
        if lst[i] % 2 == 0:
            for j in range(i + 1, n):   # Loop 2
                lst[j] = lst[j] + 1
```

Prove matching upper (Big-Oh) and lower (Omega) bounds on the worst-case running time of `my_alg`. Clearly label which part of your solution is a proof of the upper bound, and which part is a proof of the lower bound. You may use properties of divisibility (e.g., about even and odd numbers) in your analysis without proving them. You may also use the summation formula $\sum_{i=0}^{m} i = \frac{m(m+1)}{2}$.

HINT: when Loop 2 runs, all the elements of `lst` from indexes `i+1` to `n-1` switch from even to odd, or vice versa.

Solution

Upper bound on worst-case running time.

Let $n \in \mathbb{N}$, and let `lst` be an *arbitrary* list of integers of length n. The body of Loop 2 takes constant time (1 step) and executes at most $i \leq n$ iterations, for a total of at most n steps.

Since the outer loop executes exactly n iterations, and each iteration takes time at most n steps, the total time is at most $n^2 \in \mathcal{O}(n^2)$.

Lower bound on worst-case running time.

Let $n \in \mathbb{N}$ and let `lst` $= [0, -1, \ldots, -n+1]$.

Then for each iteration of the outer loop, `lst[i]` is even, which causes the inner loop to execute $n-i$ iterations, each in constant time (1 step). This adds 1 to every element in `lst[i+1:n]`, causing `lst[i+1]` to become equal to 0, which ensures that the `if` condition will be true at the next iteration of the outer loop.

Since this happens at each iteration, the total number of iterations of the inner loop is $(n-1)+(n-2)+\cdots+0 = n(n-1)/2$. Since each of these iterations counts as 1 step, the overall running time of `my_alg` on this input is $\Omega(n^2)$.

Use this page for rough work. If you want work on this page to be marked, please indicate this clearly *at the location of the original question.*