University of Toronto
Faculty of Arts and Science

CSC165H1S Midterm 2, Version 3

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 **7 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 binary representation of the decimal number 165. The representation should not have any leading zeros. HINT: $2^3 = 8$, $2^4 = 16$, $2^5 = 32$, $2^6 = 64$, $2^7 = 128$.

Solution

$165 = (10100101)_2$

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

Solution

$$\sum_{i=0}^{n-1} 1 \cdot 3^i = \frac{3^n - 1}{2}$$

(c) **[2 marks]** Let $f(n) = \dfrac{3n}{\log_2 n + 8}$ and $g(n) = n^{\log_2 n}$. For each statement below, check one box to indicate whether the statement is true or false.

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

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

```python
def f(n: int) -> None:
    """Precondition: n >= 0."""
    i = 3
    while i < n * n * n;
        i = i * i
```

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

Solution

$i_k = 3^{2^k}$

(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 $i_k \geq n^3$, i.e., $3^{2^k} \geq n^3$.
$k = \lceil \log_2(\log_3(n^3)) \rceil = \lceil \log_2(3 \log_3 n) \rceil$.

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

$$\forall n \in \mathbb{N},\ n \geq 1 \Rightarrow \sum_{i=1}^{n} \frac{1}{\sqrt{i}} > \sqrt{n} - 1$$

HINTS: for all $m \in \mathbb{Z}^+$, $\sqrt{m+1} - \sqrt{m} = \dfrac{1}{\sqrt{m+1} + \sqrt{m}}$, and $\dfrac{1}{\sqrt{m+1} + \sqrt{m}} < \dfrac{1}{\sqrt{m+1}}$.

Solution

Proof. **Base case**: let $n = 1$. We'll prove that $\sum_{i=1}^{n} \frac{1}{\sqrt{i}} > \sqrt{n} - 1$.

The left side is

$$\sum_{i=1}^{1} \frac{1}{\sqrt{i}} = \frac{1}{\sqrt{1}} = 1$$

The right side is $\sqrt{1} - 1 = 0$.

So the left side is greater than the right side.

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

We start with the left side of the target inequality:

$$\begin{aligned}
\sum_{i=1}^{k+1} \frac{1}{\sqrt{i}} &= \left(\sum_{i=1}^{k} \frac{1}{\sqrt{i}}\right) + \frac{1}{\sqrt{k+1}} \\
&> \sqrt{k} - 1 + \frac{1}{\sqrt{k+1}} && \text{(by I.H.)} \\
&> \sqrt{k} - 1 + \frac{1}{\sqrt{k+1} + \sqrt{k}} && \left(\text{since } \frac{1}{\sqrt{k+1}} > \frac{1}{\sqrt{k+1} + \sqrt{k}}\right) \\
&= \sqrt{k} - 1 + (\sqrt{k+1} - \sqrt{k}) && \text{(using the first HINT)} \\
&= \sqrt{k+1} - 1
\end{aligned}$$

□

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.

$$\forall a \in \mathbb{R}^+,\ a > 1 \Rightarrow a^n + 3 \in \Theta(2^n)$$

The next page is left blank for rough work and/or to continue your proof.

Solution

The negation is

$$\exists a \in \mathbb{R}^+,\ a > 1 \land a^n + 3 \notin \mathcal{O}(2^n) \lor a^n + 3 \notin \Omega(2^n)$$

Proof. Let $a = 3$. We'll prove that $a^n + 3 \notin \mathcal{O}(2^n)$.

Let $c, n_0 \in \mathbb{R}^{\geq 0}$. Let $n = \left\lceil \max(n_0, \log_{3/2} c) \right\rceil + 1$. We'll prove that $n \geq n_0$ and that $a^n + 3 > c2^n$.

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 $a^n + 3 > c2^n$.

Using the definition of n, we have:

$$\begin{aligned} n &> \log_{3/2} c \\ \left(\frac{3}{2}\right)^n &> c \\ 3^n &> c2^n \\ 3^n + 3 &> c2^n && \text{(since } 3^n + 3 > 3^n\text{)} \end{aligned}$$

$\square$

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

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

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

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 $m \in \mathbb{N}$):

$$\sum_{i=0}^{m} i = \frac{m(m+1)}{2}$$

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 $\geq i$, Loop 2 takes $\frac{i}{2}$ iterations (we're ignoring floor/ceiling here).

Next, we need to add up this number over all iterations of Loop 1. For Loop 1, i starts at 0 and increases by 1 at each iteration, until $i^2 \geq n$. This occurs when $i = \sqrt{n}$ (again, ignore floor/ceiling). So i takes on the values $0, 1, \dots, \sqrt{n} - 1$.

So the total number of iterations of Loop 2 is:

$$\begin{aligned}\sum_{i=0}^{\sqrt{n}-1} \frac{i}{2} &= \frac{1}{2} \sum_{i=0}^{\sqrt{n}-1} i \\ &= \frac{1}{2} \cdot \frac{(\sqrt{n}-1)\sqrt{n}}{2} \\ &= \frac{1}{4}n - \frac{1}{4}\sqrt{n}\end{aligned}$$

(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] > i:
            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 the summation formula from part (a).

HINT: when Loop 2 runs, all elements of `lst` from indexes `i+1` to `n-1` decrease by 1.

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 the inner loop takes constant time (1 step) and executes $n - i - 1$ times, which is $\leq n$. So the loop takes at most n steps.

Since the outer loop executes exactly n iterations, and each iteration takes time at most n steps for the inner loop, 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 `lst` $= [n + 1, n + 2, \ldots, 2n]$. Notice that at the start of the algorithm, `lst[j] > j + n` for every index j. Since `lst[j] = lst[j] - 1` is executed at most once for each index j during each iteration of the outer loop, this means that the condition `lst[j] > j` remains true during the entire execution of the algorithm, for each index j.

Then, for each iteration of the outer loop, the inner loop executes $n - i - 1$ iterations, each one taking constant time (1 step). Over all the iterations of the outer loop, the inner loop therefore executes a total of $\sum_{i=0}^{n-1}(n - i - 1) = n(n + 1)/2$ steps. Hence, the overall running time of `my_alg` on this input is $\Omega(n^2)$.