Assignment 1

Course: COMPSCI 2C03
Date: 09/26/2025
Author: Sulagna Nandi
MacID: nandis
Student Number: 400575118
Professor: Dr. Vincent Maccio

Problem 1.1: Ordering Functions

Big-O Definition

$$f(n) \in O(g(n)) \iff \exists c, n_0 > 0 \ \big(\forall n \geq n_0,\ 0 \leq f(n) \leq c \cdot g(n)\big)$$

Big-Ω Definition

$$f(n) \in \Omega(g(n)) \iff \exists c, n_0 > 0 \ \big(\forall n \geq n_0,\ f(n) \geq c \cdot g(n) \geq 0\big)$$

Big-Θ Definition

$$f(n) \in \Theta(g(n)) \iff \big(f(n) \in O(g(n))\big) \wedge \big(f(n) \in \Omega(g(n))\big)$$

a) $f(n) = \log_5(n) = \frac{\log(n)}{\log(5)}$

Set $c = \frac{1}{\log(5)}$ and $n_0 = 1$.

$$\forall n \geq n_0, \quad f(n) \leq c \cdot \log(n) \quad \Rightarrow \quad f(n) \in O(\log(n))$$

$$\forall n \geq n_0, \quad f(n) \geq c \cdot \log(n) \quad \Rightarrow \quad f(n) \in \Omega(\log(n))$$

$$\therefore f(n) \in O(\log(n)) \quad \wedge \quad f(n) \in \Omega(\log(n))$$

$$\boxed{f(n) \in \Theta(\log(n))}$$

b) $f(n) = n^{1/2}$

Set $c = 1$ and $n_0 = 1$.

$$\forall n \geq n_0, \quad f(n) \leq c \cdot n^{1/2} \quad \Rightarrow \quad f(n) \in O(n^{1/2})$$

$$\forall n \geq n_0, \quad f(n) \geq c \cdot n^{1/2} \quad \Rightarrow \quad f(n) \in \Omega(n^{1/2})$$

$$\therefore f(n) \in O(n^{1/2}) \ \wedge \ f(n) \in \Omega(n^{1/2})$$

$$\boxed{f(n) \in \Theta(n^{1/2})}$$

c) $f(n) = n^2$

Set $c = 1$ and $n_0 = 1$.

$$\forall n \geq n_0, \quad f(n) \leq c \cdot n^2 \quad \Rightarrow \quad f(n) \in O(n^2)$$

$$\forall n \geq n_0, \quad f(n) \geq c \cdot n^2 \quad \Rightarrow \quad f(n) \in \Omega(n^2)$$

$$\therefore f(n) \in O(n^2) \ \wedge \ f(n) \in \Omega(n^2)$$

$$\boxed{f(n) \in \Theta(n^2)}$$

d) $f(n) = \sum_{i=1}^{n} n(\frac{1}{2})^i = 2n(1 - (\frac{1}{2})^n)$

$f(n) = 2n(1 - (\frac{1}{2})^n)$

$\leq 2n(1)$

$= 2n$

Set $c = 2$ and $n_0 = 1$.

$$\forall n \geq n_0, \quad f(n) \leq c \cdot n \quad \Rightarrow \quad f(n) \in O(n)$$

$f(n) = 2n(1 - (\frac{1}{2})^n)$

$\geq 2n(\frac{1}{2})$

$= n$

Set $c = 1$ and $n_0 = 1$.

$$\forall n \geq n_0, \quad f(n) \geq c \cdot n \quad \Rightarrow \quad f(n) \in \Omega(n)$$

$$\therefore f(n) \in O(n) \ \wedge \ f(n) \in \Omega(n)$$

$$\boxed{f(n) \in \Theta(n)}$$

e) $f(n) = \prod_{i=0}^{7} i^2 = 0$

Set $c = 1$ and $n_0 = 1$.

$$\forall n \geq n_0, \quad f(n) \leq c \cdot (1) \quad \Rightarrow \quad f(n) \in O(1)$$

Since $f(n) = 0$, it is not an element of any Ω family.

$$f(n) = 0 \implies \neg\exists c, n_0, g(n) > 0 \ \big(\forall n \geq n_0, \ f(n) \geq c \cdot g(n)\big)$$

$$\boxed{f(n) \in O(1)}$$

f) $f(n) = n^{-2}$

Set $c = 1$ and $n_0 = 1$.

$$\forall n \geq n_0, \quad f(n) \leq c \cdot n^{-2} \quad \Rightarrow \quad f(n) \in O(n^{-2})$$

$$\forall n \geq n_0, \quad f(n) \geq c \cdot n^{-2} \quad \Rightarrow \quad f(n) \in \Omega(n^{-2})$$

$$\therefore f(n) \in O(n^{-2}) \ \wedge \ f(n) \in \Omega(n^{-2})$$

$$\boxed{f(n) \in \Theta(n^{-2})}$$

g) $f(n) = n\log(n)$

Set $c = 1$ and $n_0 = 1$.

$$\forall n \geq n_0, \quad f(n) \leq c \cdot n\log(n) \quad \Rightarrow \quad f(n) \in O(n\log(n))$$

$$\forall n \geq n_0, \quad f(n) \geq c \cdot n\log(n) \quad \Rightarrow \quad f(n) \in \Omega(n\log(n))$$

$$\therefore f(n) \in O(n\log(n)) \ \wedge \ f(n) \in \Omega(n\log(n))$$

$$\boxed{f(n) \in \Theta(n\log(n))}$$

h) $f(n) = n^3$

Set $c = 1$ and $n_0 = 1$.

$$\forall n \geq n_0, \quad f(n) \leq c \cdot n^3 \quad \Rightarrow \quad f(n) \in O(n^3)$$

$$\forall n \geq n_0, \quad f(n) \geq c \cdot n^3 \quad \Rightarrow \quad f(n) \in \Omega(n^3)$$

$$\therefore f(n) \in O(n^3) \quad \wedge \quad f(n) \in \Omega(n^3)$$

$$\boxed{f(n) \in \Theta(n^3)}$$

i) $f(n) = \frac{1}{n^{-2}} = n^2$

Set $c = 1$ and $n_0 = 1$.

$$\forall n \geq n_0, \quad f(n) \leq c \cdot n^2 \quad \Rightarrow \quad f(n) \in O(n^2)$$

$$\forall n \geq n_0, \quad f(n) \geq c \cdot n^2 \quad \Rightarrow \quad f(n) \in \Omega(n^2)$$

$$\therefore f(n) \in O(n^2) \quad \wedge \quad f(n) \in \Omega(n^2)$$

$$\boxed{f(n) \in \Theta(n^2)}$$

j) $f(n) = \log_n(n \cdot 2^n) = 1 + n \cdot \frac{\log(2)}{\log(n)} = 1 + \frac{n}{\log(n)}$

$f(n) = 1 + \frac{n}{\log(n)}$

$\leq \frac{n}{\log(n)} + \frac{n}{\log(n)}$

$= \frac{2n}{\log(n)}$

Set $c = 2$ and $n_0 = 1$.

$$\forall n \geq n_0, \quad f(n) \leq c \cdot \frac{n}{\log(n)} \quad \Rightarrow \quad f(n) \in O(\frac{n}{\log(n)})$$

$f(n) = 1 + \frac{n}{\log(n)}$

$\geq \frac{n}{\log(n)}$

Set $c = 1$ and $n_0 = 1$.

$$\forall n \geq n_0, \quad f(n) \geq c \cdot \frac{n}{\log(n)} \quad \Rightarrow \quad f(n) \in \Omega(\frac{n}{\log(n)})$$

$$\therefore f(n) \in O(\frac{n}{\log(n)}) \ \wedge \ f(n) \in \Omega(\frac{n}{\log(n)})$$

$$\boxed{f(n) \in \Theta(\frac{n}{\log(n)})}$$

k) $f(n) = 2^n$

Set $c = 1$ and $n_0 = 1$.

$$\forall n \geq n_0, \quad f(n) \leq c \cdot 2^n \quad \Rightarrow \quad f(n) \in O(2^n)$$

$$\forall n \geq n_0, \quad f(n) \geq c \cdot 2^n \quad \Rightarrow \quad f(n) \in \Omega(2^n)$$

$$\therefore f(n) \in O(2^n) \ \wedge \ f(n) \in \Omega(2^n)$$

$$\boxed{f(n) \in \Theta(2^n)}$$

l) $f(n) = \sqrt[3]{n^6} = n^2$

Set $c = 1$ and $n_0 = 1$.

$$\forall n \geq n_0, \quad f(n) \leq c \cdot n^2 \quad \Rightarrow \quad f(n) \in O(n^2)$$

$$\forall n \geq n_0, \quad f(n) \geq c \cdot n^2 \quad \Rightarrow \quad f(n) \in \Omega(n^2)$$

$$\therefore f(n) \in O(n^2) \ \wedge \ f(n) \in \Omega(n^2)$$

$$\boxed{f(n) \in \Theta(n^2)}$$

m) $f(n) = n^{\log_3(2)}$

Set $c = 1$ and $n_0 = 1$.

$$\forall n \geq n_0, \quad f(n) \leq c \cdot n^{\log_3(2)} \quad \Rightarrow \quad f(n) \in O(n^{\log_3(2)})$$

$$\forall n \geq n_0, \quad f(n) \geq c \cdot n^{\log_3(2)} \quad \Rightarrow \quad f(n) \in \Omega(n^{\log_3(2)})$$

$$\therefore f(n) \in O(n^{\log_3(2)}) \ \wedge \ f(n) \in \Omega(n^{\log_3(2)})$$

$$\boxed{f(n) \in \Theta(n^{\log_3(2)})}$$

Note: $\frac{1}{2} < \log_3(2) < 1$

n) $f(n) = \prod_{i=1}^{n} \sqrt{4} = 2^n$

Set $c = 1$ and $n_0 = 1$.

$$\forall n \geq n_0, \quad f(n) \leq c \cdot 2^n \quad \Rightarrow \quad f(n) \in O(2^n)$$

$$\forall n \geq n_0, \quad f(n) \geq c \cdot 2^n \quad \Rightarrow \quad f(n) \in \Omega(2^n)$$

$$\therefore f(n) \in O(2^n) \ \wedge \ f(n) \in \Omega(2^n)$$

$$\boxed{f(n) \in \Theta(2^n)}$$

Groups in Order of Increasing Time Complexity

1. $\prod_{i=0}^{n} i^2 \in O(1) \quad (f(n) = 0 \text{ so nothing smaller})$
2. $\frac{1}{n^2} \in \Theta(n^{-2})$
3. $\log(n \cdot 2^n) \in \Theta(\frac{n}{log(n)})$
4. $\log_5(n) \in \Theta(\log(n))$
5. $n^{1/2} \in \Theta(n^{1/2})$
6. $n^{\log_2(3)} \in \Theta(n^{\log_2(3)})$
7. $\sum_{i=1}^{n} n \left(\frac{1}{2}\right)^i \in \Theta(n)$
8. $n\log(n) \in \Theta(n\log(n))$
9. $n^2,\ \frac{1}{n^{-2}},\ \sqrt[3]{n^6} \in \Theta(n^2)$
10. $n^3 \in \Theta(n^3)$
11. $2^n,\ \prod_{i=1}^{n} \sqrt{4} \in \Theta(2^n)$

This order can be confirmed by taking adjacent functions $g_1(n) < g_2(n)$ and showing (using L'Hôpital's rule when needed):

$$\lim_{n\to\infty} \frac{g_2(n)}{g_1(n)} = \infty \quad \Rightarrow \quad g_1(n) \in O(g_2(n))$$

or

$$\lim_{n\to\infty} \frac{g_1(n)}{g_2(n)} = 0 \quad \Rightarrow \quad g_2(n) \in \Omega(g_1(n))$$

Problem 1.2: Induction Proof for Recurrence

Consider the recurrence

$$T(n) = \begin{cases} 1 & \text{if } n = 0, \\ 4 & \text{if } n = 1, \\ 4T(n-1) - 4T(n-2) & \text{if } n > 1. \end{cases}$$

We want to prove that

$$T(n) = f(n) = 2^n(n+1).$$

Base Cases

For $n = 0$:

$$T(0) = 1, \quad f(0) = 2^0(0+1) = 1 \quad \checkmark$$

For $n = 1$:

$$T(1) = 4, \quad f(1) = 2^1(1+1) = 4 \quad \checkmark$$

Inductive Step

Assume the formula holds for n and $n-1$:

$$T(n) = 2^n(n+1), \quad T(n-1) = 2^{n-1} \cdot n.$$

Now compute $T(n+1)$:

$$\begin{aligned} T(n+1) &= 4T(n) - 4T(n-1) \\ &= 4 \cdot \big(2^n(n+1)\big) - 4 \cdot \big(2^{n-1}n\big) \\ &= 2^n(4n+4) - 2^n(2n) \\ &= 2^n(2n+4) \\ &= 2^{n+1}(n+2). \end{aligned}$$

This matches the closed form: $f(n+1) = 2^{n+1}(n+2)$.

$$\boxed{\therefore \text{ By induction, } T(n) = 2^n(n+1) \quad \forall n \geq 0.}$$

Algorithm 1 IsSorted(L)

Pre: L is an *array*.

$i, result := 0, \text{true}$
while $i \neq |L| - 1$ **do**
  if $L[i] > L[i+1]$ **then**
    $result := \text{false}$
  end if
  $i := i + 1$
end while
return $result$

Post: return *true* if the list L is sorted.

Problem 2.1: Invariant for IsSorted

Predicate for whether the section of the list that has been traversed so far up to and including index i is sorted:

$$P(i) := \forall j \in [0, i),\ (L[j] \leq L[j+1])$$

Invariant which will evauluate to true if and only if ”result = true” and P(i) (which indicates whether the sub-list traversed thus far is sorted) have the same truth value:

$$\boxed{I(i, \text{result})\ :=\ \ 0 \leq i \leq |L| - 1, \quad \big(P(i)\ \iff\ \text{result} = \text{true}\big)}$$

Problem 2.2: Bound Function for IsSorted

The loop will run while $i > 0$ and $i \leq |L| - 1$, and i is incremented by 1 each iteration. We can define a function for the number of iterations remaining as:

$$\boxed{f(i) := (|L| - 1) - i.}$$

Since $|L|, i \in \mathbb{N}$ and $0 \leq i \leq |L| - 1$, $f(i) \in \mathbb{N}$. Since i is strictly increasing, $f(i)$ must be strictly decreasing. Thus, we know $f(i)$ will eventually reach 0, which means the loop will terminate.

Problem 2.3: Proving IsSorted is Correct

Base case: Prove invariant holds before the loop. Input: L is an array.

$$i = 0, \quad \text{result} = \text{true}$$

$I(0, \text{true})$ is vacuously true since there's no j such that $0 \leq j < i = 0$.

Inductive step: Assume $I(i_k, \text{result}_k) = \text{true}$. This is the same as stating $P(i_k) \iff \text{result}_k = \text{true}$.

Case 1: $L[i_k] > L[i_k + 1]$

- The code changes `result` to false. So $\text{result}_{k+1} = \text{false}$.
- By the end of the iteration, i is incremented by 1. Let $i_{k+1} = i_k + 1$.
- This means that, by the end of the iteration, $\exists j \in [0, i_{k+1})$ such that $L[j] > L[j+1]$. The witness is $j = i_k$. This means $P(i_{k+1}) \equiv \text{false}$.
- $P(i_{k+1}) \equiv \text{false}$ and $(\text{result}_{k+1} = \text{false})$, it can be concluded that $I(i_{k+1}, \text{result}_{k+1}) \equiv \text{true}$.

$$\therefore \text{ For Case 1, } I(i_k, \text{result}_k) \Rightarrow I(i_{k+1}, \text{result}_{k+1}).$$

Case 2: $L[i_k] \leq L[i_k + 1]$

- The code maintains the value of result. So $\text{result}_{k+1} = \text{result}_k$.

Subcase 2a: $\text{result}_k = \text{true}$

- By the induction hypothesis, $P(i_k) \equiv \text{true}$.
- By the end of the iteration, i is incremented, so $i_{k+1} = i_k + 1$.
- Since $L[i_k] \leq L[i_k + 1]$ and $P(i_k) \equiv \text{true}$ by the induction hypothesis,
$$\forall j \in [0, i_k + 1)\ (L[j] \leq L[j+1]) \ \equiv \ \text{true}$$
- This means $P(i_{k+1}) \equiv \text{true}$. Since $\text{result}_{k+1} = \text{result}_k = \text{true}$, it can be concluded that $I(i_{k+1}, \text{result}_{k+1}) \equiv \text{true}$.

$$\therefore \text{ For Subcase 2a, } I(i_k, \text{result}_k) \Rightarrow I(i_{k+1}, \text{result}_{k+1}).$$

Subcase 2b: $\text{result}_k = \text{false}$

- By the induction hypothesis, $P(i_k) \equiv \text{false}$.
- By the end of the iteration, i is incremented, so $i_{k+1} = i_k + 1$.
- Since $P(i_k) \equiv \text{false}$ by the induction hypothesis,
$$\forall j \in [0, i_k + 1)\ (L[j] \leq L[j+1]) \ \equiv \ \text{false}$$
- This means $P(i_{k+1}) \equiv \text{false}$. Since $\text{result}_{k+1} = \text{result}_k = \text{false}$, it can be concluded that $I(i_{k+1}, \text{result}_{k+1}) \equiv \text{true}$.

$$\therefore \text{ For Subcase 2b, } I(i_k, \text{result}_k) \Rightarrow I(i_{k+1}, \text{result}_{k+1}).$$

$I(i, \text{result})$ holds for the base case, $I(i_k, \text{result}_k) \Rightarrow I(i_{k+1}, \text{result}_{k+1})$ for all possible cases past the base case, and $f(i)$ is a bound function that guarantees the loop will terminate (as explained in Problem 2.2). Therefore, the IsSorted algorithm is correct.

Problem 2.4: Time and Memory Complexity

Time Complexity

The while loop in the IsSorted function will run exactly $|L|-1$ times. This is because i is initialized as $i := 0$, i is incremented by 1 in each iteration, and the loop will terminate if and only if i has reached the value $i = |L|-1$. This totals to $|L|-1$ iterations regardless of whether the list is sorted and up until which index it is sorted. If we let $n = |L| - 1$ and $\text{IsSortedTime}(n)$ be the amount of time it take the algorithm to run, the techniques from Problem 1.1 can be used to show $\text{IsSortedTime}(n) \in O(n)$ and $\text{IsSortedTime}(n) \in \Omega(n)$, which means $\text{IsSortedTime}(n) \in \Theta(n)$.

Memory Complexity

Let $n = |L|$ and the memory used by the IsSorted function be $\text{IsSortedMemory}(n)$. Since the array L is not copied in this algorithm and only two variables are created, namely i and result, the memory used by the IsSorted algorithm does not depend on n and will be constant. Only the memory taken and used by i and result should be considered. This means $\text{IsSortedMemory}(n) \in O(1)$ and $\text{IsSortedMemory}(n) \in \Omega(1)$, meaning $\text{IsSortedMemory}(n) \in \Theta(1)$.

Problem 2.5: FasterIsSorted Algorithm

Algorithm 2 FasterIsSorted(L)

Pre: L is an *array*.

$i := 0$
while $i \neq |L| - 1$ **do**
    if $L[i] > L[i+1]$ **then**
        return false
    end if
    $i := i + 1$
end while
return true

Post: return *true* if the list L is sorted.

Let $i = p$ be the first index in L where $L[i] > L[i+1]$ and $n := |L|$. The while loop in FasterIsSorted will run (including the iteration when $L[i] > L[i+1]$) exactly $p+1$ times. This means that the runtime complexity of FasterIsSorted only depends on p and not n (and we already know that $p < n$). Let FasterIsSortedTime(p) be the amount of time the function takes to run. Using the techniques from Problem 1.1, it can be shown that FasterIsSortedTime(p) $\in O(p)$ and FasterIsSortedTime(p) $\in \Omega(p)$, which means IsSortedTime(p) $\in \Theta(p)$.