**Question 1**

**(a) (5 points) Show that f(x) = x2 + 4x is O(x2).**

* x2 + 4x ≤ 𝑐 | x2 |
* x2 + 4x ≤ x2 + 4 x2 ∀x > 1
* x2 + 4x ≤ 5x2 ∀x > 1
* x0 = 1 𝑎𝑛𝑑 𝑐 = 5

Therefore, f(x)=O(x2).

**(b) (5 points) Show that f(x) = x2 is NOT O(√x).**

If x2 ≤ 𝑐√x;

x3/2 ≤ c;

As x →∞, x3/2→∞, but c is a constant.

Therefore, f(x) = x2 is NOT O(√x).

**Note:**

x2 ≤ x2 ∀x > 1

x2 ≤ x2 ∀x > 1

x0 = 1 𝑎𝑛𝑑 𝑐 = 1

**(c) (5 points) Show that f (x) = x is Ω(log x).**

x => c log x

2x => cx

x0 = 1, c = 2

Therefore, f (x) = x is Ω(log x).

**(d) (10 points) Show that f(x) = (2x2 − 3)/((3x4 + x3 − 2x2 − 1) is Θ(x−2).**

f(x) = (2x2 − 3)/((3x4 + x3 − 2x2 − 1)

2x2 – 3 is nearly equal to 2x2 ;

3x4 + x3 − 2x2 – 1 is nearly equal to 3x4 ;

f(x)≈2/3x2 ;

**Big-O:**

f(x) <= cg(x), x > x0

2/3x2 <= c x-2

Therefore, c = 1and x0 = 1, O(x−2);

**Big-Ω:**

f(x) => cg(x), x > x0

2/3x2 => c x-2

Therefore, c = 1/3 and x0 = 1, Ω(x−2);

Thus, that f(x) = (2x2 − 3)/((3x4 + x3 − 2x2 − 1) is Θ(x−2).

A white background with black text

Description automatically generated

Rank:

f3(x) -> f1(x) -> f5(x) -> f2(x) -> f4(x)

Explain my answer(Big-O):

f3(x) = √x <= c√x ∀x > 1 ⇒ c = 2; x0 = 1

Big-O Complexity is O(√x), so it is slower than x.

f1(x) = xlog2​x <= cx2 ∀x > 1 ⇒ c = 1; x0 = 1

Big-O Complexity is O(xlog2​x), so it is faster than x.

f5(x) and f2(x) belong to nx type.

f5(x) = 2x => c x2 ∀x > 4 ⇒ c = 1; x0 = 4

Therefore, Big- Ω for 2x  is x2.

Big-O Complexity is O(nx), so 3x is faster than 2x

f4(x) = x! => c3x  ∀x > 4 ⇒ c = 1; x0 = 7

Therefore, Big- Ω for x! is 3x.

And then,

Big-O

f4(x) = x! <= cx! ∀x > 1 ⇒ c = 2; x0 = 1

Result of Rank:

f3(x) -> f1(x) -> f5(x) -> f2(x) -> f4(x)

**Question 3 ：**

**(a) (15 points) Please describe an efficient algorithm in English using a data structure such as array / linked list / stack / queue to solve this problem.**

**1. Create a new Stack:**

* It can store opening parenthesis ( {, (, [) when they are encountered.

**2. Traverse the String**

* Use “for” to loop through each character into string.
* If it’s an opening parenthesis, “push” it into “stack”.
* If it’s a closing parenthesis:

First, check if the stack is empty.

Secondly, “pop” the top element and check if it matches the closing parenthesis.

Finally, check if the stack is empty.

**(b) (5 points) What is the asymptotic upper bound of complexity of running time for your algorithm?**

**A white paper with black text

Description automatically generated**

**Big-O:**

𝑓 (x) = n + 6k + 4 (k < n) ≈ n + 6n + 4 <= 7n+4n ∀n > 1 ⇒ c = 11; x0 = 1

𝑓(𝑛) 𝑔𝑟𝑜𝑤𝑠 𝑎𝑠 𝑂(𝑛)