1. **Given:** A program P which computes the function f(n)=1 for all nEN and an arbitrary program Q

**To Prove:** There exists another program which R which determines whether P(n) = Q(n) for any input n.

**Proof:** We know that P(n) computes f(n) = 1. We can define program P as:

define P(n)

return 1

Let the program Q(n) be defined as follows:

define Q(n):

P(n)

return 1

Here, Program Q calls program P(n) and returns 1when P(n) halts for an input n. That is when Q(n) halts for an input.

If P(n) = Q(n), then Q(n) returns 1 for all inputs. This implies that P(n) halts for any given input which solves the halting problem.

But, since, halting problem is undecidable, our assumption is contradicting.

Therefore, we can’t determine conclusively whether two programs are equal or not.

5. For a given two functions f, g and h. Decide whether each of the following statements are correct and give a proof for each part.

1. If f(n) = Ω(g(n)) and g(n) = Ω(f(n)), then f(n) = Θ(g(n)).

Solution: The above statement is correct and the proof is as follows:

Given: f(n) = Ω(g(n))

g(n) = Ω (f(n))

To Prove: f(n) = Θ(g(n))

Proof: We have f(n) = Ω(g(n))

By definition of Big-Omega notation, it implies

∃*c1*>0, *n0*≥0: ∀*n*>*n0*:

*f*(*n*) ≥ *c1g*(*n*) …………. (Equation 1)

Also, g(n) = Ω (f(n))

i.e., ∃*c2*>0, *n0*≥0: ∀*n*>*n0*:

*g*(*n*) ≥ *c2f*(*n*)

i.e., 1/c2 \* (g(n)) ≥ *f*(*n*) [As c2 > 0]

Let 1/c2 = c (some constant)

Therefore, cg(n) ≥ *f*(*n*) .…..……..(Equation 2)

From Equations (1) and (2), we get

cg(n) ≥ *f*(*n*) ≥ *c1g*(*n*) …………..(Equation 3)

where c, c1>0

We know that, f (n) = Θ(g(n))

iff ∃c1,*c2*>0, *n0*≥0: ∀*n*>*n0*:

c1g(n) ≤ f(n) ≤ c2g(n) ……………..(Equation 4)

From equations (3) and (4), we conclude that given f(n) = Θ(g(n)) if f(n) = Ω(g(n)) and g(n) = Ω(f(n)).

Hence, proved.

1. If f(n) = o(g(n)), then g(n) ̸∈ O(f(n))

Solution: The above statement is correct and the proof is as follows:

Given: f(n) = o(g(n))

To Prove: g(n) ̸∈ O(f(n))

Proof by contraction:

We have, f(n) = o(g(n))

By definition of Little-Oh, it implies:

By definition of Big-Omega notation, it implies

∀*c1*>0: ∃*n0*≥0: ∀*n*>*n0*:

*f*(*n*) < *c1g*(*n*) …………. (Equation 1)

Let’s assume, g(n) ∈ O(f(n))

i.e., ∃c2>0, *n0*≥0: ∀*n*>*n0*:

g(n) ≤ c2f(n) …………..(Equation 2)

We can see that Equation (2) contradicts with Equation (1).

Hence, g(n) ̸∈ O(f(n)) if f(n) = o(g(n))

1. If f(n) = O(h(n)) and g(n) = O(h(n)), then f(n) + g(n) = O(h(n))

Solution: The above statement is correct and the proof is as follows:

Given: f(n) = O(h(n))

g(n) = O(h(n))

To Prove: f(n) + g(n) = O(h(n))

Proof:

We have f(n) = O(h(n))

i.e., by definition, ∃c1>0, *n0*≥0: ∀*n*>*n0*:

f(n) ≤ c1h(n) …………..(Equation 1)

Also, g(n) = O(h(n))

i.e., by definition, ∃c2>0, *n0*≥0: ∀*n*>*n0*:

g(n) ≤ c2h(n) …………..(Equation 2)

L.H.S. = f(n) + g(n)

≤ c1h(n) + c2h(n) …..……[From 1 and 2]

≤ (c1 + c2)h(n)

≤ ch(n) [where c = c1 + c2]

Therefore, f(n) + g(n) ≤ ch(n)

i.e., f(n) + g(n) = O(h(n))

1. If f(n) = O(h(n)) and g(n) = O(h(n)), then f(n) · g(n) = O(h(n))

Solution: The above statement is incorrect and the proof is as follows:

Given: f(n) = O(h(n))

g(n) = O(h(n))

To Prove: f(n) · g(n) = O(h(n))

Proof:

We have f(n) = O(h(n))

i.e., by definition, ∃c1>0, *n0*≥0: ∀*n*>*n0*:

f(n) ≤ c1h(n) …………..(Equation 1)

Also, g(n) = O(h(n))

i.e., by definition, ∃c2>0, *n0*≥0: ∀*n*>*n0*:

g(n) ≤ c2h(n) …………..(Equation 2)

L.H.S. = f(n) \* g(n)

≤ c1h(n) \* c2h(n) …..……[From 1 and 2]

≤ (c1 \* c2)[h(n)]2

≤ c[h(n)]2 [where c = c1 \* c2]

Therefore, f(n) \* g(n) ≤ c[h(n)]2

i.e., f(n) \* g(n) < O((h(n)2))

Hence, the given statement is false.

4. Arrange the following functions in ascending order of growth rate. (10 Points)

Solution:

Applying logarithms and simplifying given functions, we get:

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **g(x)** | **log(g(x))** |
| 1. | n^(101/100) | 1.01\*log n |
| 2. | n2^(n+1) | log n + (n+1)(log 2) |
| 3. | n(log n)^3 | log n + 3 \* log (log n) |
| 4. | n\*log n | log n + log log n |
| 5. | n^(log log n) | log log n \* log n |
| 6. | log(n^(2n)) | log log n^(2n) |
| 7. | n^(log n) | log n \* log n |
| 8. | 2^n | n \* log 2 |
| 9. | n2^n | log n + n \* log 2 |
| 10. | 2^(sqrt(log n)) | log 2 \* (sqrt (log n)) |
| 11. | 2^(2^(n+1)) | 2^(n+1) \* log 2 |
| 12. | e^e^n | e^n \* (log e) |
| 13. | log(n!) | log n + log log n |
| 14. | e^(log n) | log n \* log e |
| 15. | 2^(log(sqrt(n)) | 0.5 \* log 2 \* log n |
| 16. | (sqrt 2)^(log n) | 0.5 \* log 2 \* log n |
| 17. | 2^n^2 | n^2 \* log 2 |
| 18. | n! | n log n |
| 19. | (log n)! | log n \* log log n |
| 20. | log log n | log log log n |

By comparing log(g(x)) values for all given g(x) from the above table, we get can write the functions in ascending order of complexity as follows:

g20 < g10 < g15 = g16 < g14 < g1 < g19 <= g5 < g13 < = g4 < g6 < g3 < g7 < g8 < g9 < g2 < g18 < g17 < g11 < g12

i.e.,

log log n < 2^(sqrt (log n)) < 2^(log (sqrt n)) = (sqrt 2)^(log n) <

e^(log n) < n^(101/100) < (log n)! <= n^(log log n) < log (n!) <=

n log n < log n^(2n) < n(log n)^3 < n^(log n) < 2^n < n2^n < n2^(n+1) < n! < 2^n^2 < 2^2^(n+1) < e^e^n

-You will be using Blackboard for submissions to Part 1 and the website HackerRank for submissions to Part 2.

-single BlackBoard submission

The names and school email addresses of the members of your group

Your type set solutions to Part 1 of the assignment

The HackerRank username you submitted Part 2 under, exactly as it appears on the HackerRank website.

you do not need to submit any code on Blackboard.

single HackerRank username for your group

use their built in editor to submit your code in whichever language you are using.

We expect that the code will be reasonably commented, with variable and function names that make sense

If you have any doubts that what you did is correct, feel free to show me your code during office hours and I will give feedback. \*\*

Computation of lower bound:

Using guess and proof by induction method:

We assume that lower bound of T(n) = omega(2^n)

Proof: By mathematical induction:

Let P(n) : T(n) = 3T(n-1) >= 2^n

Base case: P(1) : 3T(0)>=2^1

P(1) : 3 >= 2

Inductive case:

P(n+1): 3T(n+1-1) >= 2^(n+1)

: 3T(n) >=2.2^n

: 3.2^n >=2.2^n

: 3>=2