HW2

1) n^2

(a) $(2n)^2=4n^2$: It becomes 4 times slower.

(b) $(n+1)^2=n^2+2n+1$: Factor of increase is $(n^2+2n+1)/n^2 \approx 1+2/n+1/n^2$ for large n.

2)n^3

(a) $(2n)^3=8n^3$: It becomes 8 times slower.
(b) $(n+1)^3=n^3+3n^2+3n+1$: Factor of increase is $(n^3+3n^2+3n+1)/n^3$ aprox $1 +3/n + 3/n^2 + 1/n^3$.

3)$100n^2$

(a) $100(2n)^2 = 400n^2$: It becomes 4 times slower.

(b) $100(n+1)^2=100(n^2+2n+1)$: Factor of increase is $100(n^2+2n+1)/100n^2$ aprox $1+2/n+1/n^2$

4)n log n

(a) 2n log(2n)=$2n(\log 2 + \log^2 n)=2n \log n + 2n \log 2$:Aproxmately doubles but increase by 2 log 2times due to the log 2 term.

- **(b) (n+1)log(n+1):T**his is a bit more complex,but increases proportionally by a factor approaching log(n+1)-log aprox 1/n for large n.

5)2^n

(a)$2^{2n} =(2^n)^2 = 2^n .2^n=4^n$: Exponential increase,precisely sqaured.

(b)$2^{n+1} = 2^n .2$:exactly two times slower.

3)

To arrange the given functions in ascending order of growth rate, we'll analyze each function's asymptotic behavior as n becomes large. The function that grows slowest as \(n \) increases should be listed first, and the one that grows fastest should be listed last. Here's a breakdown of each function's growth:

1. $f_2(n)$ = sqrt{2n})

   This function grows as Theta sqrt{n}, which is slower than linear growth.

2. $f_3(n) = n + 10$

   This is a linear function Theta(n) , and grows faster than sqrt{n}.

3. $f_1(n) = n^{2.}$

   This polynomial function grows faster than linear but slower than cubic functions. It is Theta($n^{2.5}$).

4. $f_6(n)$ = n2 log n

   This function grows faster than any polynomial n^k for $k \leq 2$ but slower than n^3. It is super-quadratic due to the logarithmic factor.

5. $f_4 = 10^n$

   This exponential function grows very fast, significantly faster than polynomial or polynomial-logarithmic functions.

6.$f_5 = 100^n$

   This function, also exponential, grows even faster than 10^n to a larger base.

Arranging them in order of ascending growth rate gives:

- $f_2(n)$ = sqrt{2n} (smallest growth rate)
- $f_3(n) = n + 10$
- $f_1(n) = n^{2.}$

- $f_6(n) = n2 \log n$

- $f_4 = 10^n$

- $f_5 = 100^n$ (largest growth rate)

This list ensures that each function is O(g(n)) of the function that follows it, reflecting an increasing order of growth rates.

5)

Given the statement that f(n) is O(g(n)), let's analyze each sub-statement to decide whether they are true or false, providing proofs or counterexamples accordingly:

(a) $\log_2 f(n)$ is $O(\log_2 g(n))$.

True. Since f(n) = O(g(n)), by definition, there exists a positive constant c and a value n_0 such that

f(n) <= c * g(n) for all n >= n_0. Taking logarithms of both sides:

$\log_2 f(n) <= \log_2 (c * g(n)) = \log_2 c + \log_2 g(n)$

Here, $\log_2 c$ is a constant. Hence, $\log_2 f(n)$ can be bounded by a constant plus $\log_2 g(n)$, which means $\log_2 f(n) = O(\log_2 g(n))$.

(b) $2^{(f(n))}$ is $O(2^{g(n)})$.

False. This statement can be false because exponential growth is sensitive to the exponent's rate of increase. Consider f(n) = g(n) + $\log_2$ n. Clearly, f(n) = O(g(n)) since the logarithmic term grows much slower than any polynomial, and thus does not affect the overall class. However:

$2^{(f(n))} = 2^{(g(n)} + \log_2 n = 2^{(g(n))} * n$

This is not $O(2^{(g(n))})$ because the factor of n makes $2^{(f(n))}$ grow faster than any constant multiple of $2^{(g(n))}$.

c) n^2 is $O(g(n)^2)$.

True. Again, since f(n) = O(g(n)), there exists some constant c such that f(n) <= c * g(n) for all

n >= n_0. Squaring both sides yields:

$f(n)^2 <= (c * g(n))^2 = c^2 * g(n)^2$

Thus, $f(n)^2 = O(g(n)^2)$ because we can find a constant c^2 that bounds $f(n)^2$ in terms of $g(n)^2$.

These proofs and counterexamples show the nuanced effects of different mathematical operations on growth rates and asymptotic behaviors.

"How to Design Algorithms"

As of my understanding on how to design algorithms So far is that Read the problem and Understand it well because it will help us in designing a better solution with existing resources(programing languages and data structures) in hand.

When designing an algorithm I will try to approach with the worst scenario first and optimize it according to its efficency. I will break it down into smaller pieces to help frame my solution in a better way.

It is always important to analyze an algorithm and check its efficency because it has no value if it takes infinite amount of time and space, so keeping them in mind and designing algorithm is very crucial

for it to work as desired and also we need check how it is reacting to the input size and keep a track of it.

We also need to learn the existing techniques and methods and use them accordingly to problem that we encounter.

Designing algorithms cannot be accomplished in one day it takes a lot of effort and logical thinking

algorithms are software machines that helps us smooth our way and lead a easy life