Math Problem 1. Which of the following functions are increasing? eventually nondecreasing?

If you remember techniques from calculus, you can make use of those.

(1) $f(x) = -x^2$

(2) $f(x) = x^2 + 2x + 1$

(3) $f(x) = x^3 + x$

Soln:

1. $f(x) = -x^2$

   First derivative:

   $f'(x) = d/dx(-x^2) = -2x$

   For $x>0$, $f'(x) = -2x<0$
   For $x<0$, $f'(x) = -2x>0$
   At $x=0$, $f'(x) = 0$
   Since $f'(x)$ changes sign from positive to negative at $x=0$, $f(x)$ has a local maximum at $x=0$. $f(x)$ is increasing for $x<0$ and decreasing for $x>0$. Therefore, $f(x)$ is neither increasing nor eventually nondecreasing.

2. $f(x) = x^2 + 2x + 1$

   First derivative:

   $f'(x) = d/dx(x^2+2x+1) = 2x+2$

   To determine if $f(x)$ is increasing or decreasing, we need to analyze the sign of $f'(x)$.

   For $x > -1$, $f'(x) = 2x+2 > 0$

   For $x < -1$, $f'(x) = 2x + 2 < 0$

   At $x=-1$, $f'(x) = 0$

   Since $f'(x)$ changes sign from negative to positive at $x = -1$, $f(x)$ has a local minimum at $x = -1$. $f(x)$ is decreasing for $x < -1$ and increasing for $x > -1$. Therefore, $f(x)$ is not increasing but it is eventually nondecreasing for $x > -1$.

3. $f(x) = x^3 + x$

First derivative:

$f'(x) = d/dx(x^3+x) = 3x^2 + 1$

For all x, $f'(x)=3x^2+1 > 0$. Since $f'(x)$ is always positive, f(x) is strictly increasing for all x. Therefore, f(x) is increasing and also eventually nondecreasing.

Math Problem 2. Consider the following pairs and functions f, g. Decide if it is correct to say that, asymptotically, f grows no faster than g, g grows no faster than f, or both.

(1) $f(x) = 2x^2$, $g(x) = x^2 + 1$

(2) $f(x) = x^2$, $g(x) = x^3$

(3) $f(x) = 4x + 1$, $g(x) = x^2 – 1$

Soln:

To determine the asymptotic growth rates of the functions f(x) and g(x) in each pair, we can use Big-O notation. We will compare the growth rates by considering the dominant terms of each function.

1. $f(x) = 2x^2$, $g(x) = x^2 + 1$

   As x grows large, the dominant term in g(x) is x^2. The constant term +1 becomes negligible.

   $f(x) = 2x^2$ (Drop constant 2) $= x^2$
   $g(x) \approx x^2$

   f(x)=O(g(x)) and g(x)=O(f(x)), therefore, So, asymptotically, f and g grow at the same rate.

2. $f(x) = x^2$, $g(x) = x^3$

   As x grows large, the dominant terms are:

   $f(x) = x^2$

   $g(x) = x^3$

   Since x^2 grows slower than x^3 as x goes to infinity: f(x) = O(g(x)) But g(x) ≠ O(f(x))

   So, asymptotically, f grows no faster than g.

3. $f(x) = 4x + 1$, $g(x) = x^2 – 1$

As x grows large, the dominant terms are:

$f(x) \approx 4x$ (Drop constant) = x

$g(x) \approx x^2$

Since 4x grows slower than x^2 as x goes to infinity: $f(x) = O(g(x))$ but $g(x) \neq O(f(x))$

So, asymptotically, f grows no faster than g.

Problem 1: *GCD Algorithm.* Write a Java method int gcd(int m, int n) which accepts positive integer inputs m, n and outputs the greatest common divisor of m and n.

```java
private static int gcd(int m, int n){
    if (m == 0){
        return n;
    }
    int r;
    while (n != 0){
        r = m%n;
        m = n;
        n = r;
    }
    return m;
}
```

Problem 2: *Brute Force Solution.* Formulate your own procedure for solving the SubsetSum Problem. Think of it as a Java method.

```java
public static boolean subsetSum(int[] S, int k){
    int n = S.length;

    // Generate all subsets using a bit mask approach
    for (int i = 0; i < (1 << n); i++) {
        //List<Integer> subset = new ArrayList<>();
        int sum = 0;
        for(int j = 0; j < n; j++){
            // Check if the j-th bit in the i-th subset is set
            if((i & (1 << j)) != 0){
                //subset.add(S[j]);
                sum += S[j];
            }

        }
        // Check if the sum of the current subset is equal to k
        if(sum == k){
            return true;
        }
    }
    return false;
}
```

NB: Inefficient for large sets

- Exponential time complexity, $O(2^n)$
- However, it guarantees finding the solution if one exists.

Problem 3: *Greedy Strategies*. See if you can solve SubsetSum problems using the following *greedy* strategy. With a greedy strategy, at each step in an algorithm, a value that is optimal *at that time* is chosen.

i. Sort the set S in ascending order.
ii. Initialize an empty subset T.
iii. Iterate through each element s_i in S:
iv. If adding s_i to T does not exceed k, add s_i to T.
v. Otherwise, skip s_i.

Example 1:

Given S = {3, 5, 6, 2} and k = 10:

1. Sort S: {2,3,5,6}
2. Initialize T={}
3. Iterate through S:
   - $2 \leq 10$; Add 2 to T, so T={2}
   - $2+3 \leq 10$; Add 3 to T, so T={2,3}
   - $2+3+5 \leq 10$; Add 5 to T, so T={2,3,5}
   - $2+3+5+6 > 10$; Skip 6

Final T: {2, 3, 5} which sums to 10. This is correct.

Does Greedy strategy always works?

To determine if the greedy strategy always works, consider the following example.

Example 2:

Given S={1, 2, 5, 9} and k = 11

1. Sort S: {1, 2, 5, 9}
2. Initialize T={}
3. Iterate through SSS:
   - $1 \leq 11$; Add to T, so T={1}
   - $1+2 \leq 11$; Add 2 to T, so T={1, 2}
   - $1+2+5 \leq 11$; Add 5 to T, so T={1, 2, 5}
   - $1+2+5+9 > 11$; Skip 9

Final T: {1, 2, 5} which sums to 8. However, the correct subset that sums to 11 is {2, 9} which is missed by the Greedy strategy. Therefore, the Greedy strategy does not always work for the Subset Sum Problem.

Problem 4: You are given a solution T to a SubsetSum problem with a $S = \{s_0, s_1, \ldots, s_{n-1}\}$ and k some non-negative integer.

Set $S = \{s_0, s_1, \ldots, s_{n-1}\}$.
Sum k.
$T \subseteq S$ such that the sum of elements in T is k.

Given that s_{n-1} is in T:

i. The sum of elements in T is k.
ii. This implies $\sum_{t \in T} t = k$.
iii. Since s_{n-1} is in T, we can write $\sum_{t \in T} t = \sum_{t \in T - \{sn-1\}} t + s_{n-1}$.

Define T′:

i. Let $T' = T - \{s_{n-1}\}$
ii. Then $\sum_{t \in T'} t = k - s_{n-1}$

New set $S' = \{s_0, s_1, \ldots, s_{n-2}\}$.
New sum $k' = k - s_{n-1}$. We need to determine if $T' \subseteq S'$ and the sum of elements in T′ is k′.
Since T is a solution for the original problem, and s_{n-1} is included in T, removing s_{n-1} from T leaves us with T′, which is a subset of S′ with sum k′. Therefore, it is necessarily true that $T - \{s_{n-1}\}$ is a solution to the Subset Sum problem with inputs S′ and k′.