**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

The derivative f′(x) of a function f(x) tells us how the function behaves in terms of increasing or decreasing.

Derivative: f′(x)=−2x

This is negative for all x>0 and positive for x<0.

Since the function keeps decreasing for all large x, it is neither increasing nor eventually nondecreasing.

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

Derivative: f′(x)=2x+2

f′(x)=0 at x=−1

f′(x)<0 for x<−1x < -1 (decreasing) and f′(x)>0 for x>−1 (increasing).

Since f′(x)≥0 for x≥−1 the function is eventually nondecreasing.

1. f(x) = x3 + x

Derivative: f'(x) = 3x^2 + 1

Since 3x^2 + 1 > 0 for all x, the function is always increasing. Because it never decreases, it is 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

Leading term for both: x^2

Since constants and lower-order terms do not affect asymptotic growth: 2x^2 =Θ(x2)

There for both functions grow at an equal rate

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

Comparing growth rates: f(x)/g(x)=x^2/x^3=1/x →0 as x→∞.

Hence f(x) grows strictly slower than g(x).

1. f(x)=4x + 1, g(x) = x^2 – 1

Almost same as the previous question : f(x)/g(x)=4x+1/x^2-1=4/x →0 as x→∞.

Hence f(x) grows strictly slower than g(x).

**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

A computer screen with text and numbers

AI-generated content may be incorrect.

**Problem 2.** Brute Force Solution. Formulate your own procedure for solving the SubsetSum Problem. Think of it as a Java method subsetsum that accepts as input S, k, and outputs a subset T of S with the property that the sum of the si in T is k if such a T exists, or null if no such T can be found. (A non-null return value can be thought of as a return of “true” and a null return value signifies “false.”) Implement your idea in Java code.

Ans:

My plan is to start with an empty subset. Iterate through each number in S. Create new subsets by adding the number to each existing subset. Check if any new subset sums to k. If yes, return it. Return null if no valid subset is found.

A computer screen shot of a program

AI-generated content may be incorrect.

**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 1 that time is chosen. Decide whether the following greedy strategy works: Begin by sorting the input set S; assume that S in sorted order is as follows: {s0, s1,...,sn−1}. Initialize an empty set T; we will add elements to T as we scan S. As you scan S, if s0 ≤ k, put s0 in T; otherwise leave it out. Then, if the sum of the elements of T together with s1 is ≤ k, then put s1 in T; otherwise leave it out. Then, if the sum of the elements of T together with s2 is ≤ k, put s2 in T; otherwise, leave it out Continue this way until every number in S has been checked. To illustrate the algorithm, consider the following example: Suppose S = {3, 5, 6, 2} and k = 10. After sorting, we have S = {2, 3, 5, 6}. We notice 2 ≤ 10, so we put 2 into T. Now T = {2}. Next we check the number 3 in S. Since 2 + 3 ≤ 10, we also place 3 into T. Now T = {2, 3}. Next we check the number 5. Since 2 + 3 + 5 = 10, we include 5 in T as well. Now T = {2, 3, 5}. Finally we check 6. Since 2 + 3 + 5 + 6 > 10, we reject 6. Our final value for T is {2, 3, 5}, and this is a correct solution. For this problem, decide if this strategy always works. If not, give an example of a SubsetSum problem for which the algorithm gives an incorrect result. If you think it does work, give an argument to support your idea.

Ans:

No, it doesn’t work. The greedy strategy fails because prioritizing smaller numbers can exclude correct subsets, and once a number is skipped, it isn't reconsidered. Additionally, the approach doesn't explore all possible combinations, unlike more comprehensive methods like backtracking or dynamic programming, which are able to find the correct solution in all cases.

For example, For S={3,7,8,2} and k=10:

* Sorting gives S={2,3,7,8}.
* The algorithm picks {2,3}, summing to 5 (incorrect).
* The correct subset is {3,7} (sums to 10).

**Problem 4.** You are given a solution T to a SubsetSum problem with a S = {s0, s1,...,sn−1} and k some non-negative integer. (Recall that T is a solution if it is a subset of S the sum of whose elements is equal to k.) Suppose that sn−1 belongs to T. Is it necessarily true that the set T −{sn−1} is a solution to the SubsetSum problem with inputs S0 , k0 where S0 = {s0, s1,...,sn−2} and k0 = k − sn−1? Explain. Hint. The sum of an empty set of integers is (by convention) equal to 0.

Ans:

Yes, it is necessarily true. Since T is a solution to the SubsetSum problem, the sum of its elements equals k. Given that sn−1​ is in T, removing it results in a subset whose sum is k−sn−1​. This subset consists only of elements from S0={s0,s1,...,sn−2}, making it a valid solution to the new Subset Sum problem with target k0=k−sn−1​. Therefore, T−{sn−1} is a correct solution for the modified problem.