Problem 1 Maximum Difference (10 pts)

Given an array of numbers $x_1,\ldots,x_n$ we are interested in finding

$D = \max\ (x_j - x_i)$ where $1 \le i \le j \le n$

Describe an efficient algorithm that calculates *D*. In addition to describing the algorithm, explain the efficiency of your algorithm clearly.

```java
public class P1 {
  // In order to find the max of D, we need to the find the largest number corresponding to each
  // element following this number.
  // Time complexity: O(n). Only a single pass is needed.
  // Space complexity: O(1). Only two variables are used.
    public int maxD(int[] values) {
      // initialize the current maximum of D, which is 0.
      // initialize the index of the smallest number so far.
      int maxD = 0;
      int left = 0;

      // make a single pass with pointer called right from index 0 to the end of array
      for (int right = 0; right < values.length; right++) {
        // first check whether we need to update the "valley" of array
        if (values[left] > values[right]) {
          // upstate the left pointer with the current right index.
          left = right;
        }
        // update max of D, we need to compare current maxD and new diff made by new position.
        maxD = Math.max(maxD, values[right] - values[left]);
      }
      return maxD;
    }
}
```

Problem 2 Minimum Number of Coins (10 pts)

Given is a list of K distinct coin denominations (V_1,...,V_k) and the total sum $S>0$. Find the minimum number of coins whose sum is equal to S (we can use as many coins of one type as we want), or report that it's not possible to select coins in such a way that they sum up to S. Justify your explanation

```
public int P2(int[] coins, int amount) {
    // Dynamic programming Bottom up method.

    // edge case
    if (coins.length == 0 || coins == null) {
        return 0;
    }
    // dp stores minimum number of coins needed to make change for amount i
    int[] dp = new int[amount + 1];
    // initialize the impossible number of coins that make up amount
    Arrays.fill(dp, val: amount + 1);
    dp[0] = 0;

    // for each iteration i, compute all minimum counts for amounts up to i
    for (int i = 1; i <= amount; i++) {
        // for each coin in array
        for (int coin : coins) {
            if (i - coin >= 0) {
                // check whether adding one more coins[j] will reduce number of coins
                dp[i] = Math.min(dp[i], 1 + dp[i - coin]);
            }
        }
    }
    // return 0 if impossible, else return number
    return dp[amount] != amount + 1 ? dp[amount] : 0;
}
```

Problem 3 Consecutive sums (5 + 5 = 10 pts)

Let $(a_1,...a_n)$ be a sequence of distinct numbers some of which maybe negative. For $\leq i \leq j \leq n$, consider the sum

$S_{ij} = a_i + ... + a_j$

a) What is the running time of a brute force algorithm to calculate *max* S_{ij}?

b) Give an efficient algorithm to find the above maximum. In addition to giving the algorithm, describe the efficiency of your algorithm clearly.

a) Running time of brute force is O(n^3)

We need to calculate all the possible combinations. We can choose any possible combination of i and j, as long as i is smaller or equal to j.

for j in range 1 to n:

for i in range 1 to j:

calculate Sij = a(i) + a(i+1) + ... + a(j)

step of calculating S1j = j

step of calculating S2j = j - 1

...

step of calculating Sjj = 1

So, the running time for inner loop is (1 + j) j / 2 = O(j^2)

So, the running time for total is O(1^2 + 2^2 + ... + n^2) = O(n^3)

So O(n(n-2)/2 * n) = O(n^3)

b) Time complexity: O(n), since we iterate through every element of array once. Space complexity O(1), we only have 2 variables.

```java
public int P3(int[] nums) {
    // edge case
    if (nums.length == 1) {
        return nums[0];
    }
    // initialize two variables sum and max
    int sum = 0;
    int max = Integer.MIN_VALUE;
    // iterate every element in array
    for (int n : nums) {
        // update sum and compare max with sum
        sum += n;
        max = Math.max(max, sum);
        // reinitialize sum to 0 is sum < 0
        if (sum < 0) {
            sum = 0;
        }
    }
    return max;
}
```