October 1

Problem: Maximum Subarray Sum

Problem Statement: Given an integer array nums, find the contiguous subarray (containing at least one number) which has the largest sum and return its sum.

Link to problem:

https://leetcode.com/problems/maximum-subarray/

Example 1:

Input: nums = [-2,1,-3,4,-1,2,1,-5,4]

Output: 6

Explanation: The subarray [4,-1,2,1] has the largest sum 6.

Example 2:

Input: nums = [1]

Output: 1

Explanation: The subarray [1] has the largest sum 1.

Example 3:

Input: nums = [5,4,-1,7,8]

Output: 23

Explanation: The subarray [5,4,-1,7,8] has the largest sum 23.

Solution:

```
class Solution {
   public int maxSubArray(int[] nums) {
      // Initialize the current sum and max sum
      int curSum = 0; // Stores the running sum of the current subarray
      int max = Integer.MIN_VALUE; // Tracks the maximum subarray sum found so far

      // Iterate over the array
      for(int i = 0; i < nums.length; i++) {
         // Calculate the possible new sum by adding the current element to curSum
         int temp = curSum + nums[i];

         // If the current element itself is larger than the new sum, reset curSum
         // This starts a new subarray at the current element
         if(temp < nums[i]) {
```

```
            curSum = nums[i];
        } else {
            curSum = temp;
        }

                // Update the max with the maximum value between the current sum and the
        previous max
        if(max < curSum) {
            max = curSum;
        }
    }

    // Return the maximum sum found
    return max;
  }
}
```

Explanation:

- We start with two variables: curSum (which keeps track of the current subarray sum) and max (to store the maximum subarray sum).
- As we iterate through the array, for each element, we decide whether to add it to the current subarray or start a new subarray. This is done by checking if the current element alone is greater than the sum of the current subarray plus the element.
- If it's better to start a new subarray, we reset curSum to the current element; otherwise, we continue adding to the existing subarray.
- After processing each element, we update max if the current subarray sum (curSum) is larger than the previous maximum sum.
- Finally, after iterating through all elements, we return max, which contains the largest sum of any subarray in the array.

Time Complexity:

- **O(n)**, where n is the number of elements in the array. We iterate through the array once.

Space Complexity:

- **O(1)**, as we are using only a few extra variables (curSum and max) for storage.
