1.5.1. Number of 1 bits

Problem Statement: Given a positive integer n, write a function that returns the number of set bits in its binary representation (also known as the Hamming weight).

Code:

```
class Solution {
public:
   int
hammingWeight(uint
32_t n) {
       int count = 0;
       while (n) {
          count += n &
1;
          n >>= 1;
       }
       return count;
   }
};
```

1.5.2. Maximum SubArray

Problem Statement: Given an integer array nums, find the Subarray with the largest sum, and return its sum.

Code:

```
class Solution {
public:
   int maxSubArray(vector<int>& nums) {
      return maxCrossingSum(nums, 0, nums.size() - 1);
   }
   int maxCrossingSum(vector<int>& nums, int low, int high) {
      if (low == high) return nums[low];
}
      int mid = low + (high - low) / 2;
      int leftMax = maxSubArraySum(nums, low, mid);

      int rightMax = maxSubArraySum(nums, mid + 1, high);

      int crossMax = maxCrossingSumHelper(nums, low, mid, high);
```

```
    return max(leftMax, max(rightMax, crossMax));
}

int maxSubArraySum(vector<int>& nums, int low, int high) {
    if (low == high) return nums[low];

    int mid = low + (high - low) / 2;

    return max(maxSubArraySum(nums, low, mid),
               maxSubArraySum(nums, mid + 1, high));
}

int maxCrossingSumHelper(vector<int>& nums, int low, int mid, int high) {
    int sum = 0;
    int leftSum = INT_MIN;
    for (int i = mid; i >= low; i--) {
        sum += nums[i];
        if (sum > leftSum) {
            leftSum = sum;
        }
    }

    sum = 0;
    int rightSum = INT_MIN;
    for (int i = mid + 1; i <= high; i++) {
        sum += nums[i];
        if (sum > rightSum) {
            rightSum = sum;
        }
    }

    return leftSum + rightSum;
```

```
    }
};
```

1.5.3. Longest Nice Substring

Problem Statement: Given a string s, return the longest substring of s that is nice. If there are multiple, return the substring of the earliest occurrence. If there are none, return an empty string.

Code:

```
class Solution {
public:
    int longestNiceSubarray(vector<int>& nums) {
        int maxLen = 0;
        int left = 0;
        unordered_set<int> sums;

        for (int right = 0; right < nums.size(); right++) {
            while (sums.count(nums[right])) {
                sums.erase(nums[left]);
                left++;
            }

            sums.insert(nums[right]);
            maxLen = max(maxLen, right - left + 1);
        }

        return maxLen;
    }
};
```