Sorting Algorithms and Leetcode Problems in C++

# 1. Radix Sort - Remaining Computation and Implementation

Radix Sort is a non-comparative sorting algorithm that sorts numbers by processing individual digits. The basic idea is to sort the input numbers based on each digit, starting from the least significant to the most significant. It uses a stable counting sort as a subroutine.  
Below is the implementation of Radix Sort in C++.

```cpp  
#include <iostream>  
#include <vector>  
#include <algorithm>  
using namespace std;  
  
// A utility function to get maximum value in arr[]  
int getMax(const vector<int>& arr) {  
 return \*max\_element(arr.begin(), arr.end());  
}  
  
// A function to do counting sort of arr[] according to the digit represented by exp.  
void countingSort(vector<int>& arr, int exp) {  
 int n = arr.size();  
 vector<int> output(n); // output array  
 int count[10] = {0};  
  
 // Store count of occurrences in count[]  
 for (int i = 0; i < n; i++)  
 count[(arr[i] / exp) % 10]++;  
  
 // Change count[i] so that count[i] contains the actual position of this digit in output[]  
 for (int i = 1; i < 10; i++)  
 count[i] += count[i - 1];  
  
 // Build the output array  
 for (int i = n - 1; i >= 0; i--) {  
 output[count[(arr[i] / exp) % 10] - 1] = arr[i];  
 count[(arr[i] / exp) % 10]--;  
 }  
  
 // Copy the output array to arr[], so that arr now contains sorted numbers according to the current digit  
 for (int i = 0; i < n; i++)  
 arr[i] = output[i];  
}  
  
// The main function to that sorts arr[] using Radix Sort  
void radixSort(vector<int>& arr) {  
 // Find the maximum number to know the number of digits  
 int max = getMax(arr);  
  
 // Do counting sort for every digit. Note that exp is 10^i where i is the current digit number  
 for (int exp = 1; max / exp > 0; exp \*= 10)  
 countingSort(arr, exp);  
}  
  
int main() {  
 vector<int> arr = {170, 45, 75, 90, 802, 24, 2, 66};  
 radixSort(arr);  
 for (int i : arr)  
 cout << i << " ";  
 return 0;  
}  
```

# 2. Bucket Sort Implementation

Bucket Sort works by distributing elements into several "buckets" and then sorting each bucket individually using another sorting algorithm (often insertion sort). It is mainly useful when input is uniformly distributed over a range.  
  
Below is the implementation of Bucket Sort in C++:

```cpp  
#include <iostream>  
#include <vector>  
#include <algorithm>  
using namespace std;  
  
void bucketSort(vector<float>& arr) {  
 int n = arr.size();  
 vector<vector<float>> buckets(n);  
  
 // Put array elements in different buckets  
 for (int i = 0; i < n; i++) {  
 int idx = n \* arr[i]; // Index in bucket  
 buckets[idx].push\_back(arr[i]);  
 }  
  
 // Sort individual buckets  
 for (int i = 0; i < n; i++)  
 sort(buckets[i].begin(), buckets[i].end());  
  
 // Concatenate all buckets into arr[]  
 int index = 0;  
 for (int i = 0; i < n; i++) {  
 for (int j = 0; j < buckets[i].size(); j++) {  
 arr[index++] = buckets[i][j];  
 }  
 }  
}  
  
int main() {  
 vector<float> arr = {0.897, 0.565, 0.656, 0.1234, 0.665, 0.3434};  
 bucketSort(arr);  
  
 for (float x : arr)  
 cout << x << " ";  
 return 0;  
}  
```

# 3. Cycle Sort Implementation

Cycle Sort is an in-place, non-comparative sorting algorithm that minimizes the number of writes to the array. It is optimal in terms of the number of writes, making it ideal for applications where writing operations are expensive.  
  
Below is the implementation of Cycle Sort in C++:

```cpp  
#include <iostream>  
#include <vector>  
using namespace std;  
  
void cycleSort(vector<int>& arr) {  
 int n = arr.size();  
 for (int cycle\_start = 0; cycle\_start < n - 1; cycle\_start++) {  
 int item = arr[cycle\_start];  
 int pos = cycle\_start;  
  
 // Find the position where we put the element  
 for (int i = cycle\_start + 1; i < n; i++) {  
 if (arr[i] < item)  
 pos++;  
 }  
  
 // Skip if the item is already in the right position  
 if (pos == cycle\_start)  
 continue;  
  
 // Put the item to the right position  
 while (item == arr[pos])  
 pos++;  
 swap(item, arr[pos]);  
  
 // Rotate rest of the cycle  
 while (pos != cycle\_start) {  
 pos = cycle\_start;  
 for (int i = cycle\_start + 1; i < n; i++) {  
 if (arr[i] < item)  
 pos++;  
 }  
 while (item == arr[pos])  
 pos++;  
 swap(item, arr[pos]);  
 }  
 }  
}  
  
int main() {  
 vector<int> arr = {20, 40, 50, 10, 30};  
 cycleSort(arr);  
  
 for (int x : arr)  
 cout << x << " ";  
 return 0;  
}  
```

# 4. Wiggle Sort (Leetcode)

Wiggle Sort rearranges an array such that nums[0] <= nums[1] >= nums[2] <= nums[3] and so on.  
Below is a C++ solution for Wiggle Sort from Leetcode:

```cpp  
#include <iostream>  
#include <vector>  
using namespace std;  
  
void wiggleSort(vector<int>& nums) {  
 for (int i = 0; i < nums.size() - 1; i++) {  
 if ((i % 2 == 0 && nums[i] > nums[i + 1]) || (i % 2 == 1 && nums[i] < nums[i + 1])) {  
 swap(nums[i], nums[i + 1]);  
 }  
 }  
}  
  
int main() {  
 vector<int> nums = {3, 5, 2, 1, 6, 4};  
 wiggleSort(nums);  
  
 for (int num : nums)  
 cout << num << " ";  
 return 0;  
}  
```

# 5. Dutch National Flag Problem (Sort Colors)

The Dutch National Flag problem is about sorting an array of 0s, 1s, and 2s. Below is the solution to sort the colors (Leetcode problem):

```cpp  
#include <iostream>  
#include <vector>  
using namespace std;  
  
void sortColors(vector<int>& nums) {  
 int low = 0, mid = 0, high = nums.size() - 1;  
 while (mid <= high) {  
 if (nums[mid] == 0) {  
 swap(nums[low++], nums[mid++]);  
 } else if (nums[mid] == 1) {  
 mid++;  
 } else {  
 swap(nums[mid], nums[high--]);  
 }  
 }  
}  
  
int main() {  
 vector<int> nums = {2, 0, 2, 1, 1, 0};  
 sortColors(nums);  
  
 for (int num : nums)  
 cout << num << " ";  
 return 0;  
}  
```